Malleable Zero-Knowledge Proofs and Applications

Mikhail Volkhov

Doctor of Philosophy

Institute for Language, Cognition and Computation
School of Informatics
University of Edinburgh

2023

Abstract

In recent years, the field of privacy-preserving technologies has experienced considerable expansion, with zero-knowledge proofs (ZKPs) playing one of the most prominent roles. Although ZKPs have been a well-established theoretical construct for three decades, recent efficiency improvements and novel privacy applications within decentralized finance have become the main drivers behind the surge of interest and investment in this area. This momentum has subsequently sparked unprecedented technical advances. Non-interactive ZKPs (NIZKs) are now regularly implemented across a variety of domains, encompassing, but not limited to, privacy-enabling cryptocurrencies, credential systems, voting, mixing, secure multi-party computation, and other cryptographic protocols.

This thesis, although covering several areas of ZKP technologies and their application, focuses on one important aspect of NIZKs, namely their malleability. Malleability is a quality of a proof system that describes the potential for altering an already generated proof. Different properties may be desired in different application contexts. On the one end of the spectrum, non-malleability ensures proof immutability, an important requirement in scenarios such as prevention of replay attacks in anonymous cryptocurrencies. At the other end, some NIZKs enable proof updatability, recursively and directly, a feature that is integral for a variety of contexts, such as private smart contracts, compact blockchains, ZK rollups, ZK virtual machines, and MPC protocols generally.

This work starts with a detailed analysis of the malleability and overarching security of a popular NIZK, known as Groth16. Here we adopt a more definitional approach, studying certain properties of the proof system, and its setup ceremony, that are crucial for its precise modelling within bigger systems. Subsequently, the work explores the malleability of transactions within a private cryptocurrency variant, where we show that relaxing non-malleability assumptions enables a functionality, specifically an atomic asset swap, that is useful for cryptocurrency applications. The work culminates with a study of a less general, algebraic NIZK, and particularly its updatability properties, whose applicability we present within the context of ensuring privacy for regulatory compliance purposes.

Acknowledgements

No one walks the path truly alone. I am grateful to many people who motivated and supported me on a long journey of which this thesis is a significant milestone.

To my parents and family, treating me with kindness and patience, but at the same time cultivating and always supporting my curiosity.

To my friends and partners, providing equal amount of friendly peer pressure and warm support when it was most needed.

To Markulf Kohlweiss, my PhD supervisor, for the valuable advice during the course of this degree and the growth opportunity that it constitutes.

To my teachers Dmitry Schtukenberg and Jan Malachowski for the character-defining inspiration during my undergrad years.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Mikhail Volkhov*)

Dedication:

To Π, comparable to Ï in certain ways, but fundamentally superior in most.

Table of Contents

Chapter 1

Introduction

It becomes increasingly hard to find a space in this world that is free of technology, yet is meaningfully connected to the comforts, physical and psychological, that the modern day life is associated with. Even in the corners of our planet that are more disconnected from the economic and financial hubs it is always present, looming, and reaching from the outside: through the politics, through the goods, through the news and hearsay. Technology can be enlightening, but it can be also invasive; it can be liberating, but at the same it can be brittle and unreliable. In the age of rapid advancements in machine learning, that rather often come, quite justifiably, under the label of artificial intelligence, we cannot afford treating technology as something that is given us for free. It is not, it never was, and it becomes only more obvious over time. And while we must keep aside the question of developing a human culture that approaches computers with a proper understanding of the many caveats that often come without an appropriate warning, the responsibility of professionals in the field is very much within the scope of this work.

Cryptography as a field is, arguably, more often viewed as an inhibitor of this process, making protocols more secure, robust, reliable; more private; more user-friendly. Often, cryptographic trends tend to be reactions to external challenges. One recent example is the novel emphasis on the technical area of zero-knowledge machine learning (ZKML [awe, 2023]), that seeks to make machine learning models more aligned and user-friendly, trying to counter their tendency to abuse data and disregard individual or group rights over general performance. Differential privacy [Desfontaines and Pejó, 2020] is a different, more established approach, which is nevertheless also quite similar in its end goal. However, it is also absolutely undeniable that the steady growth of cryptocurrencies and decentralized finance, that attracted a significant amount of attention and capital to the field, was rather disruptive for certain markets. This being just one of the prime examples, the others — such as digital identity schemes, privacy-oriented instant messenger protocols, cloud computations, private machine learning, just to mention a few — are also changing both the landscape of technology and experience of human interaction with it.

But, vaguely paraphrasing Anne Morrow Lindbergh [Lindbergh, 2011], one cannot solve all the beautiful technological problems of the modern world; and I am convinced that many of them are not computer science problems at all. However one must focus on just a few, and in this way they will be more impressive, if they are only few. In this work we will focus on how different malleabilities (think: moving pieces) of cryptographic primitives, and in particular so-called zero-knowledge proofs (ZKPs) affect the privacy and security of cryptographic protocols. Undoubtedly, ZKPs will become much more central in many user-facing protocols due to the practical advancements in the field, and their power to control privacy in a much more granular way than it is often possible otherwise. Although zero-knowledge proofs find applications in many areas, the problems we are focusing on in this thesis are mostly relevant to cryptocurrencies, or more generally decentralized ledger protocols, which we will therefore cover first.

1.1 Decentralization, Privacy, Accountability

Sparked by the success of cryptocurrencies in the early 2010s, the term *Web3* came to be associated with the new generation of web protocols that highlight decentralization and token-based economics. Surely, decentralized protocols existed before that; however, the growing popularity of blockchain-based protocols is indeed more salient than could have been estimated, and this becomes only more obvious when web3 is contrasted with the concurrent growth of the big IT companies, and the all-engulfing centralized services and infrastructures that they provide.

The first half-decade of active cryptocurrency development was perhaps best characterised by research and experiments with the basic protocols, where Bitcoin [Nakamoto, 2008] undoubtedly was occupying the most visible place. Later, however, the focus shifted towards diverse financial instruments that can be realized with them. Decentralized finance (DeFi [Werner et al., 2021]), an umbrella term that covers such financial instruments in the cryptocurrency community, is one of the raison d'être of Ethereum [Wood et al., 2014], a cryptocurrency popularized by the wide applicability of its smart contract toolchain. One of the foundational pieces of DeFi is the ability (of the distributed ledger) to create user-defined tokens, and trade or exchange them directly on-chain. Ethereum, by providing Turing complete smart contracts and the ERC tokens standards (e.g. ERC20 or non-fungible ERC721), allows building tools such as automated exchanges [Xu et al., 2021], investment platforms[1,2], bidding platforms, insurance tools, NFT marketplaces, etc. Many more DeFi platforms [Gilad et al., 2017, Kiayias et al., 2017, Goodman, 2014] and cross-chain solutions [Wood, 2016, Kwon and Buchman, 2019] exist nowadays, presenting different economic and usability trade-offs.

Privacy in cryptocurrencies and DeFi is not merely a niche political or academic question. Even though human privacy as a basic right is not always a priority in web3, e.g., most transactions on Bitcoin or Ethereum are done pseudonymously (which, for most users, is not a concern until it *is*), privacy also has deep economic implications. Another reason for this seeming unpopularity is that combining privacy with DeFi tools is often technically challenging.

To solve the problem, many privacy-preserving cryptocurrencies, like Monero [Noether et al., 2016] and Zcash [Hopwood et al., 2022] to mention the most popular ones, have been developed. These solutions are undoubtedly practical, and ample academic research in the area is available [Alonso and Joancomartí, 2018, Ben-Sasson et al., 2014a, Fauzi et al., 2019, Bünz et al., 2020a, Fuchsbauer et al., 2019, Kerber et al., 2019]. The task they solve is basic and important — a private, yet secure, exchange of basic tokens. However, this task is also minimal, and most DeFi protocols that attract market attention cannot be captured by their limited functionality.

Other, often more involved solutions try to tackle this question, such as private smart contracts [Kosba et al., 2016, Kerber et al., 2021b] and privacy-friendly decentralized exchanges [Baum et al., 2021], but often find it hard to balance practical applicability with the privacy guarantees they provide. Flexibility, so desirable for

[1] `https://compound.finance/markets`
[2] `https://polygon.market.xyz/`

DeFi, is at odds with privacy, since solutions that achieve both generally requires heavier cryptographic primitives, like general NIZKs, or multi-party computation (MPC).

For example, one of the active research areas is *private exchange* of assets [Engelmann et al., 2021, Chu et al., 2020, Gao et al., 2019] — quite an important basic DeFi functionality, which we focus on in Chapter 5. The relative lack of attention to it is unfortunate, as privacy on the blockchain, such as transaction anonymity or transfer amount secrecy, is not only interesting per se (for the end user), but also changes the financial landscape of the ecosystem. This can be advantageous, for example, if restricting the adversarial view prevents certain harmful behaviour that results from gaming the market (e.g. frontrunning, Miner Extractable Value). Consider a miner which observes a buy-order for an asset that is either from a notorious investor or has an exceptionally large transfer amount. The miner can buy that asset itself early and cheaply before the order drives up the price of the asset. Private swaps of assets can mitigate such attacks. In Chapter 5 we suggest a cryptocurrency mechanism that has both support for owning multiple assets, and an embedded functionality to perform non-interactive atomic assets swaps (as well as regular transfers).

Another important aspect of DeFi is its relation to the traditional finance. While DeFi is only starting to gain traction, most value is still concentrated within the traditional mechanisms, that run in compliance to the laws imposed by the international agreements, governments, and centralised financial institutions. At this point it is, arguably, not clear, how certain aspects of traditional finance can be decentralized effectively. Certain concepts, especially trust and real-world, physical accountability, are quite effectively monetized in traditional finance, while lacking good analogues in DeFi that tends to be more anonymous and transient. Moreover, the law itself is complex, convoluted, often vague, and therefor hard to digitalize. One example of this are anti-money-laundering policies, which are sometimes implemented in the form of huge legacy codebases, run within the bank, that analyse transactions for suspiciousness based on certain enforced by law, but also privately developed and obscure algorithms.

This problem of balancing the need for accountability with the decentralized system's tendency towards privacy is definitely not an easy one; this does not mean, however, that there is necessary a hard limitation. Many technical solutions exist and are being developed; some of them are still simplistic in nature, but there is hope that scaling is not impossible. Approaching from the traditional side, centralized cryptocurrencies (CBDC [Danezis and Meiklejohn, 2016, Wüst et al., 2019, Barki and Gouget, 2020, Lipp et al., 2021, Kiayias et al., 2022]) aim to provide banks with more cryptocurrency-like protocols for monetary control; thus providing more transparency and privacy to the end users. On the other side, there exist blockchain protocols, decentralized in nature, that offer certain kind of accountability (transaction limits, or partial disclosure of otherwise anonymous data on investigation), that are supposed to make these protocols more appealing for traditional policy makers [Chen et al., 2020, Bogatov et al., 2021, Badertscher et al., 2021]. The protocol that we develop in Chapter 6 finds a direct application in the private AML case; more precisely, it can be used for a variety of *private score tracking*, credit score tracking being one of the examples.

1.2 Zero-Knowledge Arguments

A zero-knowledge proof [Goldwasser et al., 1985] (ZKP) is a cryptographic primitive that allows to prove knowledge of a statement that depends on a secret data, without disclosing the data itself. More technically, assuming $\mathcal{R}(\mathsf{x}, \mathsf{w})$ is a certain public relation (potentially, NP), encoded as a computer program, where x is public and w is private (called *instance* and *witness* correspondingly), a ZKP can prove that there exists a witness w such that $\mathcal{R}(\mathsf{x}, \mathsf{w}) = 1$, while keeping w secret. The primitive's powerful functionality often comes as surprising to the people outside of the cryptographic community; and indeed, ZKPs can be used to build novel protocols, such as private cryptocurrencies as we mentioned earlier.

The practicality of ZKPs has not always been a given. The last decade have seen rapid advancements in the development of the so-called Succinct Non-interactive Arguments of Knowledge (SNARKs) [Groth, 2010, Lipmaa, 2012, Parno et al., 2013, Danezis et al., 2014, Groth, 2016] — a type of ZKPs with very attractive real-world properties, most importantly *succinctness*, which implies constant-sized (or logarithmic) proofs. This has enabled the use of zero-knowledge proofs in practical systems, especially in the context of blockchains [Ben-Sasson et al., 2014a, Kosba et al., 2016, Steffen et al., 2019, Bowe et al., 2020, Kerber et al., 2021b]. The ready availability of cryptographic libraries implementing SNARKs has also inspired numerous other applications [Naveh and Tromer, 2016, Delignat-Lavaud et al., 2016].

But practical ZKPs are not limited by SNARKs only; many other works, notably based on the so-called Sigma protocols [Bünz et al., 2018, Attema and Cramer, 2020, Lee, 2021, Arun et al., 2022] have found wide practical applications. One of the important distinguishing factors between the two categories is that often SNARKs need to rely on a *common reference string* — a piece of data that must be generated using an MPC (colloquially called a “ceremony”) before the SNARK can be used; thus they need a form of a *trusted setup*. Although Sigma-protocol based approaches are less efficient, they often do not need this step and are therefore *transparent*. Our construction in Chapter 6 crucially depends on a NIZK by Couteau and Hartmann [Couteau and Hartmann, 2020], that is a variant of a Sigma-protocol offering novel properties when executed in the so-called bilinear group setting.

One zk-SNARK enjoys a particular focus in this work, namely Groth16 [Groth, 2016]. Due to its exceptional performance, simplicity, and near optimal proof size, until very recently it was one of the most deployed SNARKs; and even though the trends in ZK research are changing quite fast, Groth16 is likely to remain an important milestone in the cryptographic literature, maybe comparable to ElGamal encryption [ElGamal, 1985] and Schnorr [Schnorr, 1991] or DSA [PUB, 1993] signature schemes. We analyse different aspects of Groth16 in Chapters 3 and 4; the latter focuses exactly on the security of the ceremony for Groth16, which is quite characteristic of this type of NIZKs.

1.3 Malleability in Cryptography

From a practical perspective, cryptography is a set of engineering practices that are concerned with building secure tools for specialists in other areas, such as software engineers, protocol designers, and policymakers. But unlike construction tools, a failure in a cryptographic primitive has widespread, long-lasting and often devastating consequences. While Ford can recall a series of faulty trucks, the danger caused is often localised (only a few individuals will suffer; this poses no danger to international security), gradual (cars will rarely explode; most will get into minor incidents), and it is comparably easy to fix (albeit expensive). This is why cryptography as a field first focused on the primitives that satisfy the basic functionality, and *provably nothing more* — such as IND-CCA encryption [Rackoff and Simon, 1992], EUF-CMA signatures [Goldwasser et al., 1988], or strongly simulation-extractable ZKPs [Sahai, 1999] — before the question of anything "moving" within the primitive became practically relevant.

But things have changed, and in the search for innovative functionalities the field moved towards accepting more multifunctional and exotic, as traditionally viewed, primitives: e.g. fully homomorphic encryption [Gentry, 2009, Cheon et al., 2017], malleable signatures [Chase et al., 2014, Hanser and Slamanig, 2014, Crites and Lysyanskaya, 2019], reusable MPC [Benhamouda and Lin, 2020], updatable garbled circuits [Ananth et al., 2017], extendable ring signatures [Aranha et al., 2022], and malleable NIZKs.

Malleable NIZKs are particularly interesting because they are complex enough to be practically interesting, but also well-understood enough to be feasible for certain applications. The term *malleable* in the context of NIZKs means that, most generally, the NIZK can be transformed into another one. We will say that a malleable NIZK is *updatable* to highlight this transformation action as a desired functionality, especially if this malleability is not trivial, e.g. the new NIZK attests to an updated instance x'.

There are several different ways to achieve NIZK malleability. The most prominent nowadays is using *recursive techniques* that allow reasoning about verifiability relation of the proof system *within* the proof relation (or circuit) itself [Bitansky et al., 2013, Ben-Sasson et al., 2014b, Chase et al., 2014, Bünz et al., 2021a, Kothapalli et al., 2022]. The approach is sometimes called "proof-of-a-proof", because it suggests creating proofs *about* proofs. This direction of research gives rise to constant-size blockchain protocols [Bonneau et al., 2020b], efficient proof batching, PCDs, one-layer recursion for function privacy, and other applications. The functionality of recursive proofs is powerful, and the scope of application is wide; this, naturally, comes with certain practical limits on the circuit size.

But recursion is not the only avenue for applied ZKP malleability, and it is not the one we focus on in this thesis. Some zero-knowledge proofs are not tailored for circuit satisfiability, and thus it is hard to make them verify their own proofs recursively; and nevertheless these ZKPs can be malleable in critical ways. Often these ways are subtle; malleable proofs for small algebraic languages can be more practical for designing malleable signature schemes or delegatable credentials [Belenkiy et al., 2009, Crites and Lysyanskaya, 2019]. An example for such application is the protocol in Chapter 6.

1.4 Thesis Outline

In the following **Chapter 2, Background**, we review most of the common cryptographic notions necessary for understanding the rest of the thesis. We will include not only existing results, but also minor common-knowledge results, lemmas, and a literature overview of the related areas.

The rest of the chapters contain novel results that correspond to individual papers:

- **Chapter 3: Extraction and Malleability in Groth16.** We start investigating the notion of NIZK malleability by examining Groth16. Formally, we focus on the security property called Simulation Extractability (SE), a necessary security property for a NIZK argument to achieve Universal Composability (UC), a common requirement for such protocols.

  Most of the works that investigate SE focus on its strong variant which implies proof non-malleability. In this chapter we focus on a relaxed weaker notion, that we call weak-SE. It *allows proof randomization*, while guaranteeing statement non-malleability, which we argue to be more natural a security property in many practical contexts. The main result of the chapter is that we show that it is already achievable by Groth16, which was not observed before (Section 3.2). As an important corollary, we formally analyse randomizability of Groth16 in Section 3.3. The rest of the chapter deals with the practical applications of weak-SE: we show that Groth16 can be *efficiently* transformed into a *black-box weak SE NIZK*, the stronger notion that is sufficient for UC protocols.

- **Chapter 4: Secure Non-Malleable Ceremonies for SNARKs.** In this chapter we investigate ceremonies for zk-SNARKs, again targeting Groth16. Our intention is to close the empirical gap that existed between the ceremony protocol and the NIZK itself, for which we present a holistic security framework. We then revisit the BGM ceremony protocol [Bowe et al., 2017b] for a variant of Groth16. We show that the original construction can be simplified and optimized, and then prove its security in our new framework. Importantly, our construction avoids the random beacon model used in the original work.

Here, we focus on the *(non-)malleability* of the CRS, as our intention is to show that the resulting reference string is valid despite the complicated nature of updates during the ceremony. Even though the resulting NIZK is the same Groth16 we saw in Chapter 3, this section tackles a hard problem of modelling updatability algebraically. The techniques we develop and use are generalizable to other algebraic settings.

- **Chapter 5: Multi-Asset Swaps from SNARKs.** Here we focus on private cryptocurrencies that employ SNARKs, and investigate the malleability of transactions, given SNARKs. By combining insights and security properties from Zcash and SwapCT ([Engelmann et al., 2021], an atomic swap system for Monero), we present a simple zk-SNARK-based transaction scheme, called Zswap, which is somewhat malleable to allow the merging of transactions, while preserving anonymity and security. Our protocol enables multiple assets and atomic exchanges by making use of sparse homomorphic commitments with aggregatable open randomness, together with Zcash-friendly simulation-extractable NIZK proofs. This results in a provably secure privacy-preserving transaction protocol, with efficient swaps, and overall performance close to that of existing deployed private cryptocurrencies. Practically, it is similar to Zcash Sapling and benefits from existing code bases and implementation expertise.

  The investigations in this chapter started initially with examining whether updatable signatures, playing central role in SwapCT, can be replaced by updatable NIZKs, with, potentially, a functional improvement in the application. The temporary conclusion was that transaction malleability can be achieved by simpler means; we show that in this case we merely need a rerandomizable SNARK together with a homomorphic encryption scheme.

- **Chapter 6: Exploding Commitments and Applications to AML.** This chapter develops and positively answers the question of applicability of malleable NIZKs asked in the previous chapter. Here we focus on updatability of simpler, algebraic NIZKs, and the applications of such updatability.

  First, we examine the NIZK by Couteau and Hartmann [Couteau and Hartmann, 2020], which has specific updatability properties, not previously covered in the literature. We specify and elaborate on these properties in Section 6.1. The main contribution of the chapter is exemplifying the practical relevance of this strong NIZK malleability.

Therefore in Section 6.3 we introduce the notion of *Exploding Commitment Scheme* (ECS) and provide its realization which relies non-trivially on the NIZK while being concretely optimal and realistically useful. The main functionality of the ECS is to enable a set of users to maintain and homomorphically update a commitment to a value, while allowing a regulator party to learn only a pre-agreed predicate on it (whether the aggregated value *"explodes"* or not?). In our construction, the commitment maintained by the users hides the value x, but provides the regulator with a verifiable escrow of the predicate value $P(t, x)$ for a pre-selected regulator secret t. Updatable proofs are combined with standard additively homomorphic Pedersen commitments to prove soundness of homomorphic updates; we argue that this application of updatable proofs is practically optimal, compared to the existing alternatives. The ECS model can be used for a variety of anti-money laundering applications in both decentralized and traditional finance, such as maintaining credit score, enforcing transaction spending limits, or implementing generic rating mechanisms.

Chapter 2

Background

2.1 Basic Notions and Notation

Following the standard notions of complexity theory, computational cryptographic definitions are most commonly defined w.r.t. Turing machines (TM) that take 1^λ as an input parameter, explicitly or implicitly. Here, $\lambda \in \mathbb{N}$ is the so-called *security parameter*, and 1^λ is a unary representation of it (i.e. a string of length λ containing only character 1). Often these algorithms will take elements that already depend on λ — e.g. for a λ-bit sized group $\mathbb{G}$ and $g, h \in \mathbb{G}$, $\mathcal{A}(g, h)$ takes λ implicitly. Most algorithms we encounter are either (uniform) deterministic polynomial time (DPT), or (uniform) probabilistic polynomial-time (PPT), which are defined as TMs that run in polynomial time in λ.

We say that a function $f : \mathbb{N} \to \mathbb{R}$ is *negligible*, if for all big enough λ, $f(\lambda) < 1/p(\lambda)$ for all polynomials $p(\lambda)$. For simplicity we write $g(\lambda) = \mathsf{negl}(\lambda)$ or $g(\lambda) \leq \mathsf{negl}(\lambda)$ to denote that $g(\lambda)$ is some negligible function; similarly $g(\lambda) = \mathsf{poly}(\lambda)$ denotes that $g(\lambda)$ is some polynomial function. The terms *overwhelming*, as in "with overwhelming probability" or "w.o.p.", applied to $g(\lambda)$ similarly means that $1 - g(\lambda) = \mathsf{negl}(\lambda)$.

We write $y \overset{r}{\leftarrow} \mathcal{A}(x)$ or $y \leftarrow \mathcal{A}(x; r)$ when a PPT algorithm $\mathcal{A}$ outputs y on input x and uses random coins r. Often we drop r for simplicity, and just write $y \overset{\$}{\leftarrow} \mathcal{A}(x)$. Similarly, $x \overset{\$}{\leftarrow} D$ denotes random sampling from a distribution D, and when this notation is used with a finite set S, $x \overset{\$}{\leftarrow} S$ denotes *uniform* sampling from S.

A *view* of an algorithm $\mathcal{A}$ is a list denoted by $\mathsf{view}_\mathcal{A}$ which contains the data that fixes $\mathcal{A}$'s execution trace: (private) random coins, its inputs (including ones from the oracles), and outputs. We refer to the "transcript" implying only the public part of the view: that is input and output interactions of $\mathcal{A}$ with oracles and the challenger. Any algorithm $\mathcal{B}$ running any PPT $\mathcal{A}$ within itself will have access to the transcript, since it is always possible to wrap all external communication channels of $\mathcal{A}$ to record what they return. Access to the view, additionally containing random coins, usually implies that the algorithm $\mathcal{A}$ cannot sample private coins secretly from the party running it.

All our sets are ordered multisets by default, more resembling data structures than mathematical sets. The set minus operation $A \setminus B$ removes as many values in A as there are in B: if B has more values of the same type than in A, the number of elements in the resulting set is 0. The function $\mathsf{ToSet}(S)$ removes all duplicate entries in the multiset S and sorts the result according to some predefined ordering, essentially converting S to a "proper" set.

For two positive integers $m < n$, we denote discrete intervals of the form $\{m, m+1, \dots, n\}$ as $[m, n]$; in the special case of $m = 1$, we write is as $[n]$. We write $|S|$ for the cardinality of the set S, $|\vec{x}|$ for the length of the vector $\vec{x}$, thus $[|S|]$ means $\{1 \dots |S|\}$.

When indexing families of values, we commonly use comma to separate subscripts, e.g. $\{G_{\alpha,j,k}\}_{j,k}$ can denote a (j,k) indexed family of α-tagged elements. In this case α is merely a tag, a label for a collection. Sometimes we use semicolon for readability, e.g. $\{G_{\beta x:i}\}_{i=0}^{n}$ is a vector $G_{\beta x}$ indexed by i, where βx is a tag. Where subscripts are indices and where they are tags is not uniform, but in most cases clear from the context.

We use notation $\overset{\mathsf{p}}{\approx}, \overset{\mathsf{s}}{\approx}, \overset{\mathsf{c}}{\approx}$ to denote perfect, statistical, and computational indistinguishability correspondingly. The first one is equality of distributions; for definitions of the second and the latter one see e.g. [Goldreich, 2001][Sec 3.2.2].

We write vectors in bold and with an arrow on top interchangeably, e.g. $\boldsymbol{a}$ or $\vec{a}$. We write $\vec{a} \cdot \vec{b}$ for the inner product of two vectors $\vec{a}$ and $\vec{b}$.

When working with multi-variable polynomials, we generally use upper case letters for indeterminates as X, Y, Δ, X_γ, and lower case for concrete values x, y, δ, γ. We use vector notation to denote a list of formal variables, so for $\boldsymbol{X} = X_1, \dots, X_n$, we write $P(\boldsymbol{X}) \in \mathbb{F}[X_1 \dots X_n] = \mathbb{F}[\boldsymbol{X}]$ for a polynomial in these variables, and for a $\boldsymbol{x} \in \mathbb{F}^n$, $P(\boldsymbol{x})$ will denote the polynomial evaluation $P(x_1, \ldots, x_n)$. Let $\vec{a}$ and $\vec{b}$ be vectors of length n. We say that the vector $\vec{c}$ of length $2n-1$ is a *convolution* of $\vec{a}$ and $\vec{b}$ if

$$c_k = \sum_{\substack{(i,j)\in[n]\times[n]:\\ i+j=k+1}} a_i b_j \quad \text{for } k \in \{1, \ldots, 2n-1\}$$

In particular, multiplying the polynomial $\sum_{i=1}^{n} a_i X^{i-1}$ with $\sum_{i=1}^{n} b_i X^{i-1}$ produces $\sum_{i=1}^{2n-1} c_i X^{i-1}$.

We will denote languages and relations interchangeably in the following manner: $\mathcal{L}_{\mathcal{R}} = \{\mathsf{x} \mid \exists\ \mathsf{w}.\,(\mathsf{x},\mathsf{w}) \in \mathcal{R}_{\mathcal{L}}\}$, where $\mathsf{x} \in X$ is a (public) instance, $\mathsf{w} \in W$ is a (secret) witness, and the relation $\mathcal{R}_{\mathcal{L}}$ is a subset of $X \times W$. Defining $\mathcal{R}$ also determines $\mathcal{L}_{\mathcal{R}}$ uniquely; and similarly languages $\mathcal{L}$ we work with in most cases uniquely define $\mathcal{R}_{\mathcal{L}}$ (most of our $\mathcal{L}$ are **NP**).

Algorithms in this work will be written in a pseudocode with the following conventions. We use pairs like $\mathsf{acc}/\mathsf{rej}$ or $\mathsf{expl}/\mathsf{under}$ for readability as aliases for $1/0$ or $\top/\bot$. The equality-question-mark combination $\stackrel{?}{=}$ denotes the boolean result of an equality check between the expressions surrounding the symbol, while $\neq$ denotes the negation of $\stackrel{?}{=}$. Abort ("**abort**") by default means returning $\bot$ immediately, and assertions ("**assert** P") for a boolean value P abort when $\neg P$. The operator $\leftarrow$ denotes creation of a new value or variable, while $:=$ denotes re-assignment of the existent variable. Whenever we do not explicitly use one output of an algorithm, we denote the unspecified output place with $\cdot$, e.g. "$(x, \cdot) \leftarrow \mathsf{Alg}(y)$" binds the first output to variable x and leaves the second output unbound to any variable name. The symbol @ denotes temporary variable assignment in the pattern-matching cases, e.g. "$(a@(x,y),z) \leftarrow \mathsf{Foo}(1^\lambda)$" means "$((x,y),z) \leftarrow \mathsf{Foo}(1^\lambda); a \leftarrow (x,y)$". Parsing or pattern-matching always implicitly asserts that the returned elements belong to the correct domain or type: when the result of an assignment cannot be properly pattern-matched, e.g. $(a,b) \leftarrow \mathsf{Foo}(1^\lambda)$ where function Foo returns $\bot$, non-tuple value, wrong type of a or b, etc., the execution aborts.

2.2 Bilinear Groups

In this work we will consider Type III asymmetric bilinear groups [Galbraith et al., 2006], with $\mathbb{G}_1 \neq \mathbb{G}_2$ and without any efficiently computable isomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$; we will formalise them in the following way. Let $\mathsf{BGen}(1^\lambda)$ be a bilinear group generator that takes in a security parameter and outputs a bilinear group description $\mathsf{bp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e}, G, H)$ where $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are groups of prime order p, G is a generator of $\mathbb{G}_1$, H is a generator of $\mathbb{G}_2$, and $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a non-degenerate and efficient bilinear map. Sometimes we will denote the generators G_1, G_2 instead of G, H, depending on the context. For readability, we will switch between additive and multiplicative notation where needed; however $\mathbb{G}_1$ and $\mathbb{G}_2$ will always have same type of notation in the same context. That is, $\hat{e}(G, H)$ is a generator of $\mathbb{G}_T$ and for any $a, b \in \mathbb{Z}_p$, we can write $\hat{e}([a]G, [b]H) = \hat{e}(G, H)^{ab}$, where square brackets are used to merely separate field elements in additive notation.

It will be convenient to additionally use square brackets notation to represent group elements by specifying their exponents: $[a]_\iota := [a]G_\iota \in \mathbb{G}_\iota, \iota \in \{1, 2, T\}$. We will denote the (exponent-level) pairing for the square brackets notation as $[a]_1 \bullet [b]_2 := \hat{e}([a]G, [b]H) = [ab]_T$. When $\boldsymbol{a}$ is a vector of values $a_i \in \mathbb{Z}_p$, we will overload the square brackets notation, and denote a vector of $[a_i]_\iota$ by $[\boldsymbol{a}]_\iota$. In the same way we will overload $[\{a, b, c, \ldots\}]_\iota = \{[a]_\iota, [b]_\iota, [c]_\iota, \ldots\}$ for sets. When a set or vector A contains elements from several groups, we will denote it by combining all the group indices in the subscript, e.g. $[A]_{1,2,T}$ if A contains elements from all the three groups, e.g. $\{[A_1]_1, [A_2, A_3]_2, [A_4]_T\}$. In some protocols, especially in Chapter 6, we will primarily work in $\mathbb{G}_1$; thus whenever clear from the context, we will drop the index and just write $\mathbb{G}$ for $\mathbb{G}_1$, and G for its generator. The bracket notation also applies to the non-bilinear, single group scenario, e.g. in Chapter 5; in this case the ι group index is omitted.

2.3 Models and Assumptions

Cryptographic security statements are often formulated in terms of certain asymptotic probabilistic statements involving protocol algorithms. Cryptographic proofs in our work will be done in the standard game-transition style [Shoup, 2004], where a probabilistic statement about a certain algorithm, called a *game*, will be transformed step by step, until we reach a different statement that we can easily reason about. The transitions between games will be either perfect, statistical, or computational. In the last, computational case, the proof of a valid transition is by showing a *reduction* — generally, such an algorithm $\mathcal{R}$, that if two games are not computationally indistinguishable by PPT $\mathcal{A}$, $\mathcal{R}(\mathcal{A})$ will break an assumed property called a *cryptographic assumption*.

Most important assumptions in this work will be quite standard. For example, we will commonly employ the discrete logarithm (DL or DLOG) assumption in a finite group, or variants of the Diffie-Hellman (DH) assumption. The computational DLOG asumption states that it is computationally hard to compute exponents of randomly sampled elements in the given group:

Definition 2.3.1 (Computational Discrete Logarithm Assumption). *Let $\mathbb{G}$ be a cyclic group with a generator $G \in \mathbb{G}$. We say that the* computational discrete logarithm *assumption holds in $\mathbb{G}$, if for all PPT $\mathcal{A}$,*

$$\mathsf{Adv}^{DL}_{\mathbb{G},\mathcal{A}} := \left|\Pr\left[H \xleftarrow{\$} \mathbb{G}; c \xleftarrow{\$} \mathcal{A}(H) : G^c = H\right]\right| = \mathsf{negl}(\lambda)$$

The decisional DDH assumption claims that even given $[x], [y] \in GG$ for uniformly sampled $x, y \in \mathbb{Z}_p$, it is hard to distinguish between a uniformly sampled $[z] \in \mathbb{Z}_p$ and the product $[xy] \in GG$. In the following experiments the bit b switches between these two cases; remember that $xy + z$ is distributed uniformly, similarly to z itself.

Definition 2.3.2 (Decisional Diffie-Hellman Assumption). *Let $\mathbb{G}$ be a cyclic group with a generator $G = [1]$. We say that the* decisional Diffie-Hellman *assumption holds in $\mathbb{G}$, if for all PPT $\mathcal{A}$,* $\mathsf{Adv}^{DDH}_{\mathbb{G},\mathcal{A}} := |\varepsilon_0 - \varepsilon_1| = \mathsf{negl}(\lambda)$*, where*

$$\varepsilon_b := \Pr\left[x, y, z \xleftarrow{\$} \mathbb{Z}_p^* : \mathcal{A}([x], [y], [xy + bz]) = 1\right]$$

Both DL and DDH belong to the category of *falsifiable* assumptions — intuitively, this means that there is always a way to provably falsify that the assumption does not hold In the case of DL and DDH that is because they are formulated with respect to an interactive PPT challenger $\mathcal{C}$. This is somewhat implicit, but the winning condition is on the probability of the game which interacts with $\mathcal{A}$; this is exactly the challenger machine. If a falsifiable assumption does not hold, $\mathcal{C}(\mathcal{A})$ is an *explicit* test proving that the assumption is broken. For more details, see [Goldwasser and Kalai, 2016][Definition 5].

In this work we will often employ the term *standard model*. The term is quite overloaded and does not always have a well-defined meaning, but within this work it will say that we are in the standard model, if we are using only these well-understood falsifiable assumptions. Sometimes, “standard model” is used in a weaker sense to imply that no other assumptions are made; that is "under assumption X in the standard model" means that only X is assumed, and nothing else implicitly.

A non-falsifiable assumption is an assumption that does not match the definition of falsifiability. A common template example is a statement of the form $\forall \mathcal{A}.\, \exists \mathcal{E}.\, \Pr(\cdots) = \mathsf{negl}(\lambda)$, which cannot be falsified. This is because in order to break this assumption we would need to present $\mathcal{A}$ for which *no* extractor $\mathcal{E}$ "works"; however it is hard to prove statements about non-existence of certain algorithms without additional assumptions. An example of this non-falsifiable template are the so-called *knowledge assumptions* [Damgård, 1992] and various group models (AGM and GGM) that we discuss next.

The order of quantifiers imposes not only limits on falsifiability, but also on the way machines *access* each other. We say that a PPT $\mathcal{E}$ (extractor) has *black-box* access to a PPT $\mathcal{A}$, if it can only "run $\mathcal{A}$ externally" — that is, examine its outputs on certain data. In terms of quantification, a black-box extractor is introduced before the program it extracts from: e.g. $\exists \mathcal{E}.\, \forall \mathcal{A}.\, P(\mathcal{E}(\mathcal{A}(\cdot), \ldots), \ldots) = 1$ is a statement that makes $\mathcal{E}$ explicitly take $\mathcal{A}$ as an input in a black-box way. However, in many falsifiable assumptions we will often encounter *white-box* access, which is an umbrella term for modelling anything *beyond* black-box access; e.g. white-box access can model having access to the code of $\mathcal{A}$, or to its private coins or "hardcoded" secrets, even when $\mathcal{E}$ cannot run $\mathcal{A}$ directly. This ultimately is a result of $\forall \mathcal{A}.\, \exists \mathcal{E}$ quantification order — since $\mathcal{E}$ is introduced after $\mathcal{A}$, its existence depends on all the potential aspects of $\mathcal{A}$. For example, for each adversary $\mathcal{A}$ with a "hardcoded" secret key inside there exists a machine $\mathcal{E}$ that returns that secret key; the opposite black-box analogue ($\exists \mathcal{E}.\, \forall \mathcal{A}$) is not always true, because it deals with computational ability of $\mathcal{E}$ to understand what the secret key is from $\mathcal{A}$'s behaviour.

The generic group model (GGM), proposed by Shoup [Shoup, 1997], is an encompassing way of modelling cryptographic assumptions, that explicitly assumes a strong property of an adversary — that it can only interact with the group in the black-box way, in line with its interface. The algebraic group model [Fuchsbauer et al., 2018] (AGM), which we will use in this work, is an extension and relaxation of GGM, which also assumes that the adversary has a certain *form*, but instead of interacting with the group directly as in GGM, an AGM adversary is assumed to explain the group elements it returns as linear combinations of the elements it saw. The AGM is therefore located between the GGM and the standard model. We discuss the AGM in the following subsection 2.3.

Lastly, we will make use of the so-called *random oracles* — a class of special oracles that map their inputs to the randomly sampled elements. When used in the standard model, we refer to the setup as ROM (random oracle model); when used within AGM, we will explicitly refer to it as “AGM with RO”. Random oracles were introduced into applied cryptography in [Bellare and Rogaway, 1993], and they are used to model hash functions, heuristically, mimicking hash function’s behaviour producing random-looking outputs based on their input, in a deterministic manner. However it must be said that actual security definitions for hash functions capturing the aforementioned pseudorandomness are much more complex. Therefore, security of the RO in this context is *both* well-established and questionable, depending on the application. In most applications RO does lead to more efficient algorithms that we do not know attacks for; but at the same time there exist examples clearly illustrating unsoundness of ROM. Different variants of RO exist (read-only, programmable, etc), depending on how much power does the algorithm or reduction have over the oracle, or in which context or model it is used (e.g. GGM [Dent, 2002] or universal composability [Canetti et al., 2014]). For more discussion on ROM security in general, see e.g. [Koblitz and Menezes, 2015]. We elaborate on how we use RO in this work in subsection 2.3.1.1.

2.3.1 Algebraic Modelling

In Chapter 5 and Chapter 6 we will be working primarily in the standard model, sometimes with RO. In Chapter 3 and Chapter 4 that deal with Groth16, however, our proofs will be very AGM-like; however we will use a *variant* of AGM which we explain in this section.

Following [Fuchsbauer et al., 2018, Lipmaa, 2019], we say that a PPT algorithm $\mathcal{A}^{\mathsf{alg}}$ is *algebraic*, if there is a way to represent any group element it returns using elements it has seen before as a linear combination of these seen elements. More precisely, when $\mathbb{G}$ is a cyclic group of prime order p, and $\mathcal{A}^{\mathsf{alg}}$ has so far received group elements $G_1, \dots, G_n \in \mathbb{G}$ and output a group element $G_{n+1} \in \mathbb{G}_p$, then it has to also provide a vector of integer coefficients $\vec{K} = (k_1, \dots, k_n) \in \mathbb{Z}_p^n$ such that $G_{n+1} = \prod_{i=1}^{n} G_i^{k_i}$. Security against algebraic adversaries can be formalized either as a white-box knowledge-extraction assumption [Boneh and Venkatesan, 1998, Paillier and Vergnaud, 2005, Lipmaa, 2019], or by defining a separate cryptograpic model as done in the algebraic group model (AGM) [Fuchsbauer et al., 2018]. We are following the extraction assumption style from [Lipmaa, 2019], without considering the stronger hashed version that additionally allows $\mathcal{A}$ to sample random elements in $\mathbb{G}$ without knowing their exponents. Formally, the set of algebraic coefficients $\vec{K}$ is obtained by calling the algebraic extractor $\vec{K} \leftarrow \mathsf{Ext}^{\mathsf{alg}}_{\mathcal{A}}(\mathsf{view}_{\mathcal{A}})$ that is guaranteed to exist for any algebraic adversary $\mathcal{A}$. This extractor is white-box and requires $\mathcal{A}$'s view to run, not just the transcript — this includes $\mathcal{A}$'s private coins, and without it the definition is too demanding, since it is not reasonable to assume that it is possible to definitely extract linear coefficients from $\mathcal{A}$ it its private coins are unknown.

Definition 2.3.3 (Algebraic Algorithm, [Lipmaa, 2019])**.** *A PPT algorithm $\mathcal{A}$ is* algebraic *with respect to a cyclic group $\mathbb{G}_\iota$ of prime order p, if there exists a polynomial time extractor $\mathsf{Ext}^{\mathsf{alg}}_{\mathcal{A}}$ returning a coefficients matrix K, such that for all m and all efficiently sampleable distributions $\mathcal{D}$ over $(\mathbb{Z}_p^*)^m$,*

$$\Pr\left[\boldsymbol{\sigma} \xleftarrow{\$} \mathcal{D}_\lambda; \boldsymbol{e} \xleftarrow{\$} \mathcal{A}([\boldsymbol{\sigma}]_\iota); K \leftarrow \mathsf{Ext}^{\mathsf{alg}}_{\mathcal{A}}(\mathsf{view}_{\mathcal{A}}) : \boldsymbol{e} \neq [K\boldsymbol{\sigma}]_\iota\right] = \mathsf{negl}(\lambda).$$

The definition covers only the case of a single group; however it is easy to see how it can be extended to asymmetric bilinear groups — $\mathsf{Ext}^{\mathsf{alg}}_{\mathcal{A}}$ should return K with $m_1 + m_2$ rows, and the condition in the probabilistic statement is changed to $(\boldsymbol{e}_1\ \boldsymbol{e}_2)^T = \left[K(\boldsymbol{\sigma}_1\ \boldsymbol{\sigma}_2)^T\right]_{1,2}$. Similarly, it is trivially extended to the case when $\mathcal{A}$ obtains elements from an oracle (where $\mathsf{view}_{\mathcal{A}}$ captures communication with it) — we assume that the inputs to the oracle must be similarly "explained" in terms of linear combinations; and outputs of oracle queries are counted as additional group elements that $\mathcal{A}$ sees.

The discrete logarithm assumption (Definition 2.3.1) is central to secure groups, but especially in the AGM case. In proofs with algebraic adversaries, we use the following variant of the discrete logarithm assumption [Fuchsbauer et al., 2018].

Definition 2.3.4 (**(q_1, q_2)-dlog**)**.** *Let $\mathsf{bp} := (p, \mathbb{G}_1, \mathbb{G}_2, \ldots) \xleftarrow{\$} \mathsf{BGen}(1^\lambda)$ be a Type III bilinear group. We say that (q_1, q_2)-Discrete Logarithm Assumption holds in bp if for all PPT $\mathcal{A}$,*

$$\Pr\left[x \xleftarrow{\$} \mathbb{Z}_p^*; z \xleftarrow{\$} \mathcal{A}(\mathsf{bp}, [x, \ldots, x^{q_1}]_1, [x, \ldots, x^{q_2}]_2) : x = z\right] = \mathsf{negl}(\lambda).$$

The real-world security of q-**dlog** assumptions is analysed in [Blake and Garefalakis, 2004, Cheon, 2006, Kozaki et al., 2007] suggesting an attack taking about $O(\sqrt{p/q} + \sqrt{q})$ evaluations, where $p = |\,\mathbb{G}\,|$.

We also state two lemmas that will be useful in our AGM proofs.

Lemma 2.3.1 ([Bauer et al., 2020])**.** *Let Q be a non-zero polynomial in $\mathbb{Z}_p[X_1, \ldots, X_n]$ of total degree $d \geq 0$. Define $Q'(Z) := Q(R_1 Z + S_1, \ldots, R_n Z + S_n)$ in the ring $(\mathbb{Z}_p[R_1, \ldots, R_n, S_1, \ldots, S_n])[Z]$. Then the coefficient of the highest degree monomial in $Q'(Z)$ is a degree d polynomial in $\mathbb{Z}_p[R_1, \ldots, R_n]$.*

Proof. See Lemma 2.1 in [Bauer et al., 2020]. □

Lemma 2.3.2 (Schwartz-Zippel)**.** *Let P be a non-zero polynomial in $\mathbb{Z}_p[X_1, \ldots, X_n]$ of total degree $d \geq 0$. Let $S \subset \mathbb{Z}_p$, then*

$$\Pr\left[x_1, \ldots, x_n \xleftarrow{\$} S : P(x_1, \ldots, x_n) = 0\right] \leq \frac{d}{|S|}$$

Proof. See e.g. [Moshkovitz, 2010] or [Bünz and Fisch, 2022]. □

2.3.1.1 Random Oracle for Algebraic Adversaries

Random oracle in this work will me mostly used in Chapter 4 within the AGM context, but also to some degree in Chapter 5 to prove certain property of a commitment scheme. In this section we will focus on the former, more important use case; the latter one can be seen as a sub-case.

In [Fuchsbauer et al., 2018] it is also shown how to integrate the AGM with the random oracle model. In particular, we are interested in a RO that outputs group elements. These elements, as usual, are sampled lazily: when first sampled, they are added to the "history" set Q_{RO}, and on repeated queries they are taken from Q_{RO} instead of sampling a response afresh.

In Chapter 4 however we will need a weakening of the programmable RO that we refer to as a *transparent RO*, presented in Fig. 2.1. For convenience we will denote $\mathsf{RO}(\cdot) := \mathsf{RO}_0(\cdot)$. Actual protocol algorithms and the adversary $\mathcal{A}$ in all security definitions will only have access to the restricted oracle $\mathsf{RO}_0(\cdot)$. However, now with $\mathsf{RO}_1(\cdot)$ the simulator can learn the discrete logarithm r of group elements as long as it knows the input x. In the *programmable RO*, to learn the discrete logarithm,

$\underline{\mathsf{RO}_t(x):}$

$\texttt{\% Initially } Q_{\mathsf{RO}} = \emptyset$

if $Q_{\mathsf{RO}}[x] \neq \perp$ **then** $r \leftarrow Q_{\mathsf{RO}}[x]$

else $r \xleftarrow{\$} \mathbb{Z}_p;\ Q_{\mathsf{RO}}[x] \leftarrow r$

return $(\mathbf{if}\ t = 1\ \mathbf{then}\ r\ \mathbf{else}\ G^r)$

Figure 2.1: The transparent random oracle $\mathsf{RO}_0(\cdot) : \{0,1\}^* \to \mathbb{G}_1$, $\mathsf{RO}_1(\cdot) : \{0,1\}^* \to \mathbb{Z}_p$. We write $\mathsf{RO}(x)$ for the interface $\mathsf{RO}_0(x)$ provided to protocols.

the simulator $\mathcal{S}$ would program the oracle to return G^r where r is chosen by $\mathcal{S}$; with our transparent RO $\mathcal{S}$ can learn the discrete logarithm r by querying $\mathsf{RO}_1(x)$, and r is chosen by the oracle itself, thus no programming is needed. Also note that when one has access to the full $\mathsf{RO}_t(\cdot)$, it is always possible to pass only the restricted version $\mathsf{RO}_0(\cdot)$ of it to the subroutine[1], as done w.r.t $\mathcal{A}$ in all the relevant security definitions.

When working with algebraic adversaries $\mathcal{A}$ with RO, one remarkable detail with the white-box access that is already assumed by the AGM is that $\mathsf{view}_{\mathcal{A}}$ (introduced in Section 2.1) includes the RO transcript (but not RO randomness), since it contains all requests and replies $\mathcal{A}$ exchanges with the oracles it has access to, including RO. Thus access to $\mathsf{view}_{\mathcal{A}}$ is often sufficient for our proofs, even though we do not otherwise explicitly model any extractor's access to the RO history besides the aforementioned post-run view of $\mathcal{A}$.

2.4 Public Key Encryption

Public key encryption is among the most widely known cryptographic primitives, and it allows a user to encrypt certain data to another party's public key, which that party can decrypt with their secret key. We formulate the standard notion of a public key encryption scheme as follows.

Definition 2.4.1 (Public Key Cryptosystem)**.** *A public key cryptosystem* PKC *is a triple of efficient algorithms* $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *with the following functionality:*

$\mathsf{KeyGen}(1^\lambda) \xrightarrow{\$} (\mathsf{pk}, \mathsf{sk})$**:** *generates a public key* pk *and a secret key* sk*.*

[1] Colloquially, $\mathcal{A}$ is a subroutine of $\mathcal{B}$ if $\mathcal{B}$ runs $\mathcal{A}$ as part of its own execution. To not be confused with UC subroutines.

$\mathsf{Enc}(\mathsf{pk}, m; r) \to \mathsf{C}$**:** *takes as an input a public key* pk*, a message* m*, and randomness* r*, and outputs a ciphertext* C*.*

$\mathsf{Dec}(\mathsf{sk}, \mathsf{C}) \to m$**:** *takes in the secret key* sk *together with a ciphertext* C*, and outputs a message* m*.*

In this work, it will be convenient to define PKC more generally for n-vectors of messages, so assume that the message space is always a vector space. Every non-vectorised PKC can be trivially converted to the vector PKC by repeating all the algorithms n times.

The standard *completeness* property requires that all valid ciphertext decrypt to the message that was used to create it.

Definition 2.4.2 (PKE Completeness)**.** *A* $\mathsf{PKC} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is (perfectly)* complete *if for all* $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$*, and* m, r *in the message and randomness spaces correspondingly,* $\Pr[\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m; r)) = m] = 1$*.*

The standard privacy property for public cryptosystems is INDistinguishability under Chosen Plaintext Attacks which says that an efficient adversary has a negligible advantage of distinguishing $\mathsf{C}_0 \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}, m^{(0)})$ from $\mathsf{C}_1 \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}, m^{(1)})$ where $m^{(0)}, m^{(1)}$ are chosen by the adversary:

Definition 2.4.3 (IND-CPA)**.** *A* $\mathsf{PKC} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *satisfies* indistinguishability under chosen plaintext attacks *if for all* $\lambda \in \mathbb{N}$*, and all stateful PPT* $\mathcal{A}$*, the following holds:*

$$\Pr\left[\begin{array}{l} (\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda) \\ (\{m_i^{(0)}\}_{i=0}^n, \{m_i^{(1)}\}_{i=0}^n) \xleftarrow{\$} \mathcal{A}(\mathsf{pk}) \\ b \xleftarrow{\$} \{0,1\}, r \xleftarrow{\$} \mathbb{R} \\ \mathsf{C} \leftarrow \mathsf{Enc}(\{m_i^{(b)}\}_{i=1}^n, r) \\ b' \xleftarrow{\$} \mathcal{A}(\mathsf{C}) \\ \textbf{return } b' = b \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

where $\mathbb{R}$ *is the randomness space defined implicitly by* PKC*.*

A common variation of the IND-CPA definition where $\mathcal{A}$ gets several ciphertexts instead of just one (still in the left-or-right fashion) is equivalent to IND-CPA.

We will only tangentally use more powerful PKE security notions, such as IND-CCA and IK-CCA, in Theorem 5.3.6, where we will directly link the necessary definitions.

2.4.1 ElGamal Encryption Scheme

We describe a common variant of ElGamal cryptosystem [ElGamal, 1984] that is used to encrypt a vector of group elements; we will use it in Chapter 3 and Chapter 6. Let $\mathbb{G}$ be a finite group of prime order p, and $n \geq 1 \in \mathbb{N}$; the message space for the cryptosystem will then be $\mathbb{G}^n$. The vector ElGamal cryptosystem is defined as follows:

<u>$\mathsf{KeyGen}(1^\lambda)$</u>:
1. Sample $s_1, \ldots, s_n \xleftarrow{\$} \mathbb{Z}_p$
2. **return** $(\mathsf{pk} := [s_1, \ldots, s_n], \mathsf{sk} := \{s_i\}_{i=1}^n)$

<u>$\mathsf{Enc}(\mathsf{pk}, \{M_i\}_{i=1}^n \in \mathbb{G}^n; r)$</u>:
1. **if** $r = \perp$ **then** $r \xleftarrow{\$} \mathbb{Z}_p$
2. **return** $\boldsymbol{c} := ([r], r[s_1] + M_1, \ldots, r[s_n] + M_n)$

<u>$\mathsf{Dec}(\mathsf{sk}, (C_0, C_1, \ldots, C_n) \in \mathbb{G}^{n+1})$</u>:
1. **for** $i \in [1, n]$ **do** $M_i \leftarrow C_i - s_i \cdot C_0$
2. **return** $\{M_i\}_{i=1}^n$

Another variant of this scheme, that we will return to as "exponent ElGamal", works with $\mathbb{Z}_p^n$ as a message space, that is Enc receives $\{m_i\}_{i=1}^n \in \mathbb{Z}_p^n$ as input and not $\{M_i\}_{i=1}^n \in \mathbb{G}^n$. The encryption proceeds as before after converting each m_i into $[m_i] = M_i$. But this modification has an inherent limitation: the messages have to be small enough so that it is efficient to compute the discrete logarithm, since Dec algorithm will still return $[m_i] \in \mathbb{G}$ and not $m_i \in \mathbb{Z}_p$; thus the decrypting party will have to invert dlog which is only efficient for poly-sized message space ranges.

Both variants of ElGamal are known to be IND-CPA secure [Kurosawa, 2002] under the DDH assumption.

Note that ElGamal is homomorphic: if $\vec{C}_1, \vec{C}_2$ are ciphertexts of $\vec{M}_1, \vec{M}_2$ correspondingly w.r.t. a fixed pk, then the pairwise group-product $\vec{C}_1 \cdot \vec{C}_2$ is an encryption of the message $\vec{M}'$ where $\vec{M}'_i = \vec{M}_{1,i}\vec{M}_{2,i} \in \mathbb{G}$, also with respect to the same pk: In the case of exponent ElGamal, the message is a field-sum of messages $\vec{m}_{1,i} + \vec{m}_{2,i} \in \mathbb{Z}_p$. We will use ElGamal homomorphism in Chapter 6.

2.5 Non-Interactive Commitments

Commitment scheme allows a party to convert a message to a receipt (called a commitment), without revealing the message itself; and to later prove that the commitment was constructed with exactly this value "inside". In this paper we will mostly work with variants of the well-known Pedersen commitment scheme [Pedersen, 1992], which we will formalise in this section as a *sparse homomorphic commitment* (SHC) scheme.

An SHC scheme, or just SHC, is additively homomorphic for an exponential number of possible domains, also called types, however each commitment is sparsely populated, i.e. few domains have a non-zero value. Between domains, there exists no homomorphic property. Constructions were proposed by [Poelstra et al., 2019] and formalized by [Campanelli et al., 2021] which resemble vector Pedersen commitments with ad-hoc generators.

Definition 2.5.1 (Sparse Homomorphic Commitment)**.** *An* SHC *scheme with message space* $\mathbb{V}$*, type space* $\mathbb{T}$*, and randomness space* $\mathbb{R}$*, consists of algorithms* $\mathsf{ComSetup}$ *and* Commit *defined as follows:*

$\mathsf{ComSetup}(1^\lambda) \to \mathsf{pp}$: *takes the security parameter* λ *and outputs the public parameters* pp *implicitly provided to* Commit*.*

$\mathsf{Commit}(\{(\mathsf{ty}_i, a_i)\}_{i=1}^n, \mathsf{rc}) \to \mathsf{com}$: *takes a set of* distinct *types* $\mathsf{ty}_i \in \mathbb{T}$*, corresponding values* $a_i \in \mathbb{V}$*, together with the single randomness* $\mathsf{rc} \in \mathbb{R}$*, and outputs a commitment* com*.*

In addition, the SHC *scheme must be* homomorphic*:* $\mathbb{V}$ *must be a group, and there must exist an efficient operation* $\oplus$*, such that for any* $\{\mathsf{ty}_i\}_{i=1}^n \in \mathbb{T}^n$*,* $\{(a_i, a'_i)\}_{i=1}^n \in (\mathbb{V} \times \mathbb{V})^n$ *and any* $\mathsf{rc}, \mathsf{rc}' \in \mathbb{R}$ *it must hold that*

$$\begin{aligned}\mathsf{Commit}\left(\{(\mathsf{ty}_i, a_i)\}_1^n, \mathsf{rc}\right) \oplus \mathsf{Commit}\left(\{(\mathsf{ty}_i, a'_i)\}_1^n, \mathsf{rc}'\right)& \\ = \mathsf{Commit}\left(\{(\mathsf{ty}_i, a_i + a'_i)\}_1^n, \phi(\mathsf{rc}, \mathsf{rc}')\right)&\end{aligned}$$

for some function ϕ*, where zero values must not affect the input set:* $\mathsf{Commit}((\mathsf{ty}, 0), \mathsf{rc}) = \mathsf{Commit}(\emptyset, \mathsf{rc})$*.*

In other words, homomorphic SHC is additive on the same type, but acts like a vector commitment on distinct types.

We write $\mathsf{Commit}(\mathsf{ty}, a, \mathsf{rc})$ as a shorthand for $\mathsf{Commit}(\{(\mathsf{ty}, a)\}, \mathsf{rc})$. When we write $\mathsf{Commit}(a, \mathsf{rc})$, we implicitly assume that SHC in scope is defined w.r.t. the singleton type space $\mathbb{T} = \{\mathsf{ty}_0\}$, and that commitment is created for this default type. Unless specified otherwise "commitment scheme" always refers to this definition.

A *secure* commitment scheme must satisfy the standard binding and hiding properties. Binding guarantees that commitment cannot be opened to two distinct messages.

Definition 2.5.2 (Commitment Binding). *An* SHC *w.r.t.* $(\mathbb{V}, \mathbb{T}, \mathbb{R})$ *is* binding *if for all (big enough)* $\lambda \in \mathbb{N}$ *and all PPT adversaries* $\mathcal{A}$*, it holds that:*

$$\Pr\left[\begin{array}{l}
\mathsf{pp} \leftarrow \mathsf{ComSetup}(1^\lambda) \\
\left\{\left(\{(\mathsf{ty}_i^{(b)}, a_i^{(b)})\}_{i=1}^{n_b}, \mathsf{rc}^{(b)}\right)\right\}_{b=0}^{1} \xleftarrow{\$} \mathcal{A}(\mathsf{pp}) \\
\mathsf{com}^{(0)} \leftarrow \mathsf{Commit}\left(\{(\mathsf{ty}_i^{(0)}, a_i^{(0)})\}_{i=1}^{n_1}, \mathsf{rc}^{(0)}\right) \\
\mathsf{com}^{(1)} \leftarrow \mathsf{Commit}\left(\{(\mathsf{ty}_i^{(1)}, a_i^{(1)})\}_{i=1}^{n_2}, \mathsf{rc}^{(1)}\right) \\
\textbf{return } \mathsf{com}^{(0)} = \mathsf{com}^{(1)} \wedge \\
\qquad \left(\{(\mathsf{ty}_i^{(0)}, a_i^{(0)})\}_{i=1}^{n_1}, \mathsf{rc}^{(0)}\right) \neq \left(\{(\mathsf{ty}_i^{(1)}, a_i^{(1)})\}_{i=1}^{n_2}, \mathsf{rc}^{(1)}\right) \wedge \\
\qquad \left(\forall b.\ \{\mathsf{ty}_i^{(b)}\}_{i=1}^{n_b} \textit{ distinct} \wedge \left(\forall i.\ a_i^{(b)} \neq 0\right)\right)
\end{array}\right] = \mathsf{negl}(\lambda)$$

where (n_0, n_1) *is implicitly returned by* $\mathcal{A}$ *and therefore each* n_i *must be poly-sized.*

Hiding guarantees that it is impossible to distinguish between two fresh commitments, no matter what their content is.

Definition 2.5.3 (Commitment Hiding). *An* SHC *w.r.t.* $(\mathbb{V}, \mathbb{T}, \mathbb{R})$ *is* hiding *if for all (big enough)* $\lambda \in \mathbb{N}$ *and all stateful PPT adversaries* $\mathcal{A}$*, it holds that:*

$$\Pr\left[\begin{array}{l}
\mathsf{pp} \leftarrow \mathsf{ComSetup}(1^\lambda) \\
\left\{\left(\{\mathsf{ty}_i^{(j)}, a_i^{(j)}\}_{i=0}^{n_j}\right)\right\}_{j=0}^{1} \xleftarrow{\$} \mathcal{A}(\mathsf{pp}) \\
b \xleftarrow{\$} \{0,1\}, \mathsf{rc} \xleftarrow{\$} \mathbb{R} \\
\mathsf{com} \leftarrow \mathsf{Commit}\left(\{(\mathsf{ty}_i^{(b)}, a_i^{(b)})\}_{i=1}^{n_b}, \mathsf{rc}\right) \\
b' \leftarrow \mathcal{A}(\mathsf{com}) \\
\textbf{return } b' = b
\end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

Both definitions are given here in the computational flavour for illustrative purpose. Perfect binding would contradict perfect hiding since for latter Commit must be highly non-injective, so unbounded adversary can always find a collision; so only one property can be perfect. In our work we will *assume commitment hiding to be perfect*; in this case it can be formulated as perfect indistinguishability of distributions of com_i (as given to $\mathcal{A}$), for *all* possible inputs $\{(\{\mathsf{ty}_i^{(j)}, a_i^{(j)}\}_{i=0}^{n})\}_{j=0}^{1}$.

2.5.1 Pedersen Commitment Scheme

Pedersen commitment scheme [Pedersen, 1992] is a simple and one of the most commonly used commitment schemes. With a slight modification to support multiple bases, derived from [Poelstra et al., 2019], it becomes a valid multi-base SHC.

Let $\mathbb{G}$ be a finite group of prime order p with a generator G. We present the Pedersen commitment scheme with two possible instantiations. In static mode we pre-sample a finite number of generators and use them as bases assuming $\mathbb{T} = [1, n_{\mathsf{max}}]$. In the dynamic mode each base is chosen as the output of a random oracle call on the type ty. The difference is only in the way the bases are mapped to types which we express through the base mapping function $\mathsf{bm} : \mathbb{T} \to \mathbb{G}$; given bm, the Commit algorithm is the same in both modes.

$\underline{\mathsf{ComSetup}_{\mathsf{static}}(1^\lambda, n_{\mathsf{max}})}$:

1. Sample $H_1, \ldots, H_{n_{\mathsf{max}}} \xleftarrow{\$} \mathbb{G}$
2. $\mathsf{bm} \leftarrow (i \mapsto \textbf{if } i \in [1, n_{\mathsf{max}}] \textbf{ then } H_i \textbf{ else } \bot)$
3. **return** bm

$\underline{\mathsf{ComSetup}_{\mathsf{dynamic}}(1^\lambda, \mathsf{RO})}$:

1. **return** $\mathsf{bm} := (\mathsf{ty} \mapsto \mathsf{RO}(\mathsf{ty}))$

$\underline{\mathsf{Commit}(\{(\mathsf{ty}_i, a_i)\}_{i=1}^{n}, \mathsf{rc})}$:

1. **assert** $\forall i \in [n].\ \mathsf{bm}(\mathsf{ty}_i) \neq \bot$
2. **return** $G^{\mathsf{rc}} \cdot \prod_{i=1}^{n} \mathsf{bm}(\mathsf{ty}_i)^{a_i}$

In practice, the dynamic mode is instantiated by replacing RO by a cryptographic hash function $H : \mathbb{T} \to \mathbb{G}$. Then $\mathsf{Commit}(\{(\mathsf{ty}_i, a_i)\}, \mathsf{rc}) := (\prod H(\mathsf{ty}_i)^a)\, G^{\mathsf{rc}}$ with $a_i, \mathsf{rc} \in \mathbb{Z}_p$. Sometimes we will refer to the dynamic mode Pedersen scheme as “sparse Pedersen commitment scheme”, highlighting the fact that it supports polynomial number of types without pre-processing.

Assuming DLOG in $\mathbb{G}$, in the static mode Pedersen commitment scheme is secure in the standard model, and in the dynamic mode it is secure in the ROM.

2.6 Zero-Knowledge Arguments

Zero-knowledge argument systems enable a prover to convince a verifier of the validity of a statement, without revealing any additional information beyond the truth of the statement itself. We introduce the primitive gradually.

An *argument of knowledge* (AoK) is a two-party protocol, potentially interactive, that can be formalised as a tuple of algorithms $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ with the following semantics. The $\boldsymbol{\sigma} \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$ algorithm is run before any interaction occurs and it returns a *common reference string* $\boldsymbol{\sigma}$, which is used by $\mathsf{Prove}(\boldsymbol{\sigma}, \mathsf{x}, \mathsf{w})$ and $\{0,1\} \leftarrow \mathsf{Verify}(\boldsymbol{\sigma}, \mathsf{x}, \pi)$. The interaction is then between the prover party P which runs Prove, and the verifier party V which runs Verify, where the latter outputs the decision bit — whether verification passed or failed.

A *non-interactive zero-knowledge argument of knowledge* NIZK for the relation $\mathcal{R}$ can be seen as a *non-interactive* AoK that is in addition *zero-knowledge* [Rackoff and Simon, 1992]. Non-interactivity merely means that P does not expect any response from V, and Prove *only* produces a single proof object π; and that Verify similarly only works with its inputs and π obtained from P. Zero-knowledge, informally, means that π does not "leak" any information about w; it is formalized through the so-called simulator algorithm Sim.

Definition 2.6.1. *A* NIZK *for a relation* $\mathcal{R}$ *is a tuple of algorithms* $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{Sim})$ *with the following semantics:*

$\mathsf{Setup}(\lambda) \xrightarrow{\$} (\boldsymbol{\sigma}, \mathsf{td})$**:** *generates a common reference string* $\boldsymbol{\sigma}$ *and a trapdoor* td*. In real-world setups* td *must be discarded. Sometimes* Setup *will be taking relation* $\mathcal{R}$ *as an input: by this we merely highlight that* NIZK *can work with a* class *of relations, but a concrete one needs to be "fixed" in the setup phase.*

$\mathsf{Prove}(\boldsymbol{\sigma}, \mathsf{x}, \mathsf{w}) \xrightarrow{\$} \pi$**:** *produces a proof for* $(\mathsf{x}, \mathsf{w}) \in \mathcal{R}$*;*

$\mathsf{Verify}(\boldsymbol{\sigma}, \pi, \mathsf{x}) \rightarrow 0/1$**:** *verifies* π *w.r.t. the instance* x*, checking whether* $\mathsf{x} \in \mathcal{L}_{\mathcal{R}}$*;*

$\mathsf{Sim}(\boldsymbol{\sigma}, \mathsf{td}, \mathsf{x}) \xrightarrow{\$} \pi$**:** *using the trapdoor* td *for* $\boldsymbol{\sigma}$*, creates a simulated proof* π *for* $\mathsf{x} \in \mathcal{L}_{\mathcal{R}}$ *without the corresponding witness* w*. Simulation is only used in security definitions, since* td *must not be available in real world deployments.*

For conciseness, we will sometimes assume $\boldsymbol{\sigma}$ *is passed to all the algorithms implicitly.*

It is important to note that the common reference string is necessary in the standard model for achieving non-interactive zero-knowledge [Oren, 1987][Theorems 4,5]. However, CRS is not required for certain interactive protocols proven secure in the ROM, and converted into non-interactive model using Fiat-Shamir heuristic [Fiat and Shamir, 1987] that replaces the RO with a hash function. This includes basic Σ-protocols — three-round public coin proofs of knowledge. We discuss ROM-based NIZKs in more details later.

One important class of NIZKs that we will be working with are SNARKs, short for *succinct non-interactive arguments of knowledge*. A SNARK (or a zk-SNARK, used in this work interchangeably) is a NIZK that possesses a certain practical compactness property called *succinctness* [Kilian, 1992]. Most importantly, the proof has to be $O(\mathsf{polylog}(|\mathsf{x}|+|\mathsf{w}|))$ sized for any fixed λ, which means that Prove of a SNARK is, information-theoretically, quite compressing — in practice the proofs are often either constant or $\log$-sized. A well-known impossibility result [Gentry and Wichs, 2011] states that SNARKs cannot be proven secure under falsifiable assumptions. Sometimes the notion of succinctness is taken more liberally to also imply some restrictions on the proving and verifying time, such as e.g. $O(|\mathsf{x}|)$ verifier time. The practicality of novel SNARKs was perhaps the main driving force behind the success of NIZKs in the last decade [Gennaro et al., 2013, Ben-Sasson et al., 2013, Ben-Sasson et al., 2014c, Groth, 2016, Maller et al., 2019, Bowe et al., 2019, Gabizon et al., 2019].

The bare minimum security definitions a NIZK must satisfy is the classical triple of completeness, soundness, and zero-knowledge, which we present next.

Definition 2.6.2 (Completeness)**.** *A* NIZK *for a relation* $\mathcal{R}$ *is* perfectly complete*, if for any* $\boldsymbol{\sigma} \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$*, and all* $(\mathsf{x},\mathsf{w}) \in \mathcal{R}$*,*

$$\Pr\left[\mathsf{Verify}(\boldsymbol{\sigma}, \mathsf{Prove}(\boldsymbol{\sigma},\mathsf{x},\mathsf{w}),\mathsf{x}) = 1\right] = 1$$

where the randomness is over the random coins of Prove*.*

Definition 2.6.3 (Soundness)**.** *A* NIZK *for a relation* $\mathcal{R}$ *is* computationally sound*, if for all* $\mathsf{x} \notin \mathcal{L}_\mathcal{R}$ *and all PPT* $\mathcal{A}$ *playing a role of a malicious prover,*

$$\Pr\left[\begin{array}{l}\boldsymbol{\sigma} \xleftarrow{\$} \mathsf{Setup}(1^\lambda) \\ \pi \xleftarrow{\$} \mathcal{A}(\boldsymbol{\sigma})\end{array} : \mathsf{Verify}(\boldsymbol{\sigma},\pi,\mathsf{x}) = 1\right] = \mathsf{negl}(\lambda)$$

where the randomness is over the random coins of Setup *and* $\mathcal{A}$*.*

In many cases, we use a stronger definition, *knowledge soundness* (KS), that holds if from every verifying proof we can *extract* a witness. Knowledge soundness implies soundness, and it implicitly models "knowledge" — not only the witness exists, but it can be computationally extracted, if given a trapdoor td.

Definition 2.6.4 (Knowledge-Soundness)**.** *A* NIZK *for a relation* $\mathcal{R}$ *is* knowledge-sound (KS) *if for any PPT adversary* $\mathcal{A}$ *there exists a polynomial time extractor* $\mathsf{Ext}_{\mathcal{A}}$ *such that*

$$\Pr\left[\begin{array}{l}(\boldsymbol{\sigma}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{x}, \pi) \xleftarrow{\$} \mathcal{A}(\boldsymbol{\sigma}) \\ \mathsf{w} \leftarrow \mathsf{Ext}_{\mathcal{A}}(\boldsymbol{\sigma}, \mathsf{td}, \mathsf{x}, \pi)\end{array} : \begin{array}{l}\mathsf{Verify}(\boldsymbol{\sigma}, \mathsf{x}, \pi) = 1 \wedge \\ (\mathsf{x}, \mathsf{w}) \notin \mathcal{R}\end{array}\right] = \mathsf{negl}(\lambda)$$

This definition is provided here in a weaker white-box fashion, where Ext is introduced after $\mathcal{A}$; however the black-box variant of extractability is easily achieved by swapping the order of quantifiers. In this case the extractor does not even necessarily need to depend on td.

Both soundness and knowledge-soundness were given in a computational flavour. But it is possible to strengthen them by requiring perfect indistinguishability, that is requiring $\Pr[\ldots] = 0$ instead of $= \mathsf{negl}(\lambda)$. In this case the system is called a *proof* of knowledge, while computational definitions only give rise to an *argument*. In this work, however, we will sometimes use these two terms interchangeably, and instead focusing on which security definition a NIZK alone achieves.

Note, that as we mentioned before, td must be discarded in the real-world setups, so the ability to extract the witness by KS does not contradict zero-knowledge, which states that proofs should not "leak" the witness.

We recall the standard definition of zero-knowledge in two flavours explicitly since we will use both of them.

Definition 2.6.5 (Computational Zero-Knowledge)**.** *A* NIZK *for a relation* $\mathcal{R}$ *is* computationally zero-knowledge*, if for any PPT adversary* $\mathcal{A}$*,* $|\varepsilon_0 - \varepsilon_1| = \mathsf{negl}(\lambda)$*, where*

$$\varepsilon_b = \Pr\left[\ (\boldsymbol{\sigma}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda) : \mathcal{A}^{\mathcal{S}_{b,\boldsymbol{\sigma},\mathsf{td}}}(\boldsymbol{\sigma}) = 1\ \right].$$

The simulation oracle $\mathcal{S}_{b,\boldsymbol{\sigma},\mathsf{td}}$ *is the first variant as defined in Fig. 2.2.*

Definition 2.6.6 (Perfect Zero-Knowledge)**.** *A* NIZK *for a relation* $\mathcal{R}$ *is* perfectly

$\underline{\mathcal{S}_{b,\boldsymbol{\sigma},\mathsf{td}}(\mathsf{x},\mathsf{w})}$:	$\underline{\mathcal{S}_{\boldsymbol{\sigma},\mathsf{td}}(\mathsf{x})}$:
assert $(\mathsf{x},\mathsf{w}) \in \mathcal{R}$	$\pi \xleftarrow{\$} \mathsf{Sim}(\boldsymbol{\sigma},\mathsf{td},\mathsf{x})$
if $b = 0$ **then** $\pi \xleftarrow{\$} \mathsf{Prove}(\boldsymbol{\sigma},\mathsf{x},\mathsf{w})$	% $Q_{\mathcal{S}} \leftarrow \emptyset$ on init
else $\pi \xleftarrow{\$} \mathsf{Sim}(\boldsymbol{\sigma},\mathsf{td},\mathsf{x})$	$Q_{\mathcal{S}} := Q_{\mathcal{S}} \cup \{(\mathsf{x},\pi)\}$
return π	**return** π

Figure 2.2: Simulation oracles for a NIZK for a relation $\mathcal{R}$. The first variant is used for zero-knowledge game. The second variant is mostly used for simulation-extractability variants.

zero-knowledge *if for all* $(\mathsf{x},\mathsf{w}) \in \mathcal{R}$

$$\left\{ (\boldsymbol{\sigma},\pi) \;\middle|\; \begin{array}{c} (\boldsymbol{\sigma},\mathsf{td}) \xleftarrow{\$} \mathsf{Setup}(1^\lambda) \\ \pi \xleftarrow{\$} \mathsf{Sim}(\boldsymbol{\sigma},\mathsf{td},\mathsf{x}) \end{array} \right\} \overset{p}{\approx} \left\{ (\boldsymbol{\sigma},\pi) \;\middle|\; \begin{array}{c} (\boldsymbol{\sigma},\cdot) \xleftarrow{\$} \mathsf{Setup}(1^\lambda) \\ \pi \xleftarrow{\$} \mathsf{Prove}(\boldsymbol{\sigma},\mathsf{x},\mathsf{w}) \end{array} \right\}$$

2.6.1 Updatable and Malleable NIZKs

Malleability of cryptographic primitives can be treated both as a disadvantage and as functional feature. There exist many malleable cryptographic primitives that find different real-life applications: many basic primitives like encryption schemes, commitments, and signatures exhibit homomorphic properties, but also more complicated primitives, such as proof systems, and in particular NIZKs can be made malleable. Randomisability is commonly treated as a form of weak malleability, and it is arguably more widespread than elaborate data-involving transformations. We start by discussing the notion of NIZK malleability from a broader perspective. We then state the malleability definitions that we will be using in this work.

Groth-Sahai Proofs. One of the first examples of NIZK malleability which was successfully applied in practice is the Groth-Sahai (GS) proof system [Groth and Sahai, 2008, Ghadafi et al., 2010]. GS proofs are constructed for a specific (**NP**-complete) language of pairing equations (and for two other less powerful sublanguages). Their size is linear in the number of instance and witness elements forming this equation. Assuming additive notation for the target group $\mathbb{G}_T$ in the bilinear setting, each pairing equation is a zero-sum linear combination of all witness (and constant) elements, where equation coefficients are defined by an instance matrix. As mentioned before, the malleability of GS proofs has been investigated and applied practically. The work by Belenkiy et al. [Belenkiy et al., 2009] studies malleability and randomizability of GS proofs to construct delegatable anonymous credentials, [Dodis et al., 2010, Acar and Nguyen, 2011, Fuchsbauer, 2011] investigate homomorphic properties of GS proofs to build, respectively, continuous-leakage resilient signature scheme, revocation mechanism for DACs, and commuting signatures. The work of CKLM [Chase et al., 2012] generalises transformations on GS proofs. More on the application side, GS proofs are also used to construct signatures of randomizable ciphertexts [Blazy et al., 2011], as well as to build scalable mix-nets [Hébant et al., 2020].

The fact that GS proofs are malleable is partially a result of the specificity of the pairing equations language. There are two types of transformations one can apply to them. First, both commitments to the values and the proof values are randomizable, in a derivation-private sense – that is, it is hard to distinguish the randomized proof from a freshly created one. Second, the proof itself can be mauled: one can change its instance and modify the language description without knowing the witness that was originally used to construct it. There are six transformations (as described in [Chase et al., 2012]), that introduce, change or remove variables or equations, combining the proofs correspondingly.

Recursion vs direct malleability. There exist two distinct paradigms for updating proofs: changing or composing them directly, or creating a recursive proof-of-a-proof.

The latter approach is widely used in practice these days. The basic scheme is taking a proof system with a flexible language, such as what most SNARKs support, and specializing the NIZK for a recursive circuit: a circuit that can encode verifiability of the proof system itself. This allows creating a so-called "proof-of-a-proof", giving rise to other powerful primitives such as incrementally verifiable computations (IVC [Valiant, 2008]) or proof-carrying data (PCD [Chiesa and Tromer, 2010]). In practice though, recursive proof systems have to be both generally optimized and sometimes tailored to the domain. Examples of such systems include [Bitansky et al., 2013, Ben-Sasson et al., 2014b, Chase et al., 2014, Bowe et al., 2019, Bünz et al., 2020c, Bünz et al., 2020b, Bünz et al., 2021a, Kothapalli et al., 2021].

The direct approach to malleability is, roughly, everything that is not using the recursion technique. Groth-Sahai proofs fall into this category, since updating the proof amounts to direct computation on the proof elements. The malleable proof system by Couteau and Hartmann [Couteau and Hartmann, 2020] that we will focus on in Chapter 6 is, too. To give another interesting example of direct malleability, Ananth, Deshpande, Kalai and Lysyanskaya [Ananth et al., 2019] present a construction that they call fully-homomorphic NIZK (FH-NIZK), for circuit satisfiability. The fully homomorphic malleability property of this NIZK is expressed in the transformation $T : (\pi_1, \ldots, \pi_n) \mapsto \pi'$ that allows to combine n proofs $\{\pi_i\}_{i=1}^n$ attesting to $\exists w_i.\, P_i(x_i, w_i) = b_i$ where $b_i \in \{0, 1\}$, into one combined proof π' that attests to the existence of all combined witnesses, where the resulting circuit is a custom transformation P of the base circuits: $\exists w_1 \ldots w_n.\, P(P_1(x_1, w_1), \ldots, P_n(x_n, w_n))$. Moreover, π' is indistinguishable from a random proof on the combined circuit $P(P_1(\cdot\,,\cdot), \ldots, P_n(\cdot\,,\cdot))$, that is the transformation is derivation-private.

Advanced malleability. Going a bit further, it is important to mention that some works that present even more advanced versions of malleability. Ananth, Cohen, and Jain [Ananth et al., 2017] introduce a cryptographic primitive named *updatable randomized encoding* (URE), that allows to build a generic *updatable* NIZK (u-NIZK). URE is a very powerful cryptographic primitive. For bounded polynomial number of updates it can be built from OWFs, for unbounded updates it can be built from functional encryption (FE), and assuming LWE, URE implies FE. Moreover, output-compact URE implies exponentially-efficient indistinguishability obfuscation. Intuitively, the update mechanics of these more powerful proofs includes ability of the updater to compute (potentially computationally hiding) functions of the witness, and use them to create new proofs. This is while in the more basic notions the updater only sees the witness in a particularly hidden way. The analogy with functional encryption is clear.

In this work we will mostly study basic, direct, non-recursive malleability.

2.6.1.1 Definitions

Updatability for NIZKs is a general concept within malleability that covers its constructive side, like updates and their properties. This does not include *limits* of malleability, a concept that we discuss later. In this section we elaborate on the basic concepts, starting by examining languages and what updatability means for them.

Definition 2.6.7 (Updatable Language)**.** *A language $\mathcal{L}$ is* updatable *w.r.t. the class of transformations $\mathcal{T}$ if for all $T \in \mathcal{T}$, $T = (T_{\mathsf{x}}, T_{\mathsf{w}})$, and for all $(\mathsf{x}, \mathsf{w}) \in \mathcal{R}_{\mathcal{L}}$ it holds that $(T_{\mathsf{x}}(\mathsf{x}), T_{\mathsf{w}}(\mathsf{w})) \in \mathcal{R}_{\mathcal{L}}$. We call such T* valid transformations *for $\mathcal{L}$.*

Note that the functions $T_{\mathsf{x}} : X \to X, T_{\mathsf{w}} : W \to W$ are defined independently of any particular instance and witness, i.e. in T_{x} the symbol "x" is only used as a label. All the relations and functions we consider can be evaluated in polynomial time in the security parameter.

In terms of the interface, updatable NIZKs are merely standard NIZKs that additionally allows transforming a proof for x into a proof for $T_{\mathsf{x}}(\mathsf{x})$ using a routine called Update:

Definition 2.6.8 (Updatable NIZKs)**.** *A* NIZK *is* updatable *for a relation $\mathcal{R}$ and a set of transformations $\mathcal{T}$ if, additionally to Definition 2.6.1 there exists an algorithm $\mathsf{Update}(\boldsymbol{\sigma}, \pi, \mathsf{x}, T) \xrightarrow{\$} \pi'$, that updates the proof π for x into π' for $T_{\mathsf{x}}(\mathsf{x})$.*

In addition, a secure updatable NIZK must satisfy two properties. First, update completeness states that the updated proof must verify for the updated instance:

Definition 2.6.9 (Update Completeness)**.** *A non-interactive proof system for $\mathcal{R}$ satisfies* update completeness *w.r.t. a set of transformations $\mathcal{T}$, if given $(\boldsymbol{\sigma}, \cdot) \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$, for all x, π such that $\mathsf{Verify}(\boldsymbol{\sigma}, \pi, \mathsf{x}) = 1$, and all $T = (T_{\mathsf{x}}, \cdot) \in \mathcal{T}$ it holds that $\Pr\left[\mathsf{Verify}(\boldsymbol{\sigma}, \mathsf{Update}(\boldsymbol{\sigma}, \pi, \mathsf{x}, T), T_{\mathsf{x}}(\mathsf{x})) = 1\right] = 1$.*

Second, derivation privacy, states that updated proofs are distributed similarly to fresh proofs for the new instance.

Definition 2.6.10 (Derivation Privacy)**.** *A non-interactive proof system for $\mathcal{R}$ satisfies* derivation privacy *w.r.t. a set of transformations $\mathcal{T}$, if given $(\boldsymbol{\sigma}, \cdot) \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$, for all $(\mathsf{x}, \mathsf{w}) \in \mathcal{R}$, all π such that $\mathsf{Verify}(\boldsymbol{\sigma}, \pi, \mathsf{x}) = 1$, and all $T = (T_{\mathsf{x}}, T_{\mathsf{w}}) \in \mathcal{T}$ it holds that*

$$\left\{\mathsf{Update}(\boldsymbol{\sigma}, \pi, \mathsf{x}, T)\right\} \overset{p}{\approx} \left\{\mathsf{Prove}(\boldsymbol{\sigma}, T_{\mathsf{x}}(\mathsf{x}), T_{\mathsf{w}}(\mathsf{w}))\right\}$$

Because updated proofs are distributed as fresh ones, they can be simulated using the standard simulator guaranteed by zero-knowledge; therefore transformed proofs are also ZK. This property is inspired by derivation privacy in [Chase et al., 2012].

We call a NIZK *randomizable*, if it can be nontrivially updated w.r.t. identity transformation T_{id} with the update being derivation private. Conceptually, this means that the new π' is distributed as a fresh proof. We will denote $\mathsf{Rand}(\boldsymbol{\sigma}, \pi, \mathsf{x}) := \mathsf{Update}(\boldsymbol{\sigma}, \pi, \mathsf{x}, T_{\mathsf{id}})$.

Definition 2.6.11 (Proof Rerandomization)**.** *A* NIZK *for* $\mathcal{R}$ *is* rerandomizable *if it is updatable w.r.t. the identity transformation* T_{id}*, and furthermore for all* $(\mathsf{x}, \mathsf{w}) \in \mathcal{R}$*, all* $\boldsymbol{\sigma}$ *output by* $\mathsf{Setup}(\mathcal{R}_\lambda)$ *and all* π *such that* $\mathsf{Verify}(\boldsymbol{\sigma}, \mathsf{x}, \pi) = 1$ *it holds that*

$$\left\{\mathsf{Rand}(\boldsymbol{\sigma}, \pi, \mathsf{x})\right\} \overset{p}{\approx} \left\{\mathsf{Prove}(\boldsymbol{\sigma}, \mathsf{x}, \mathsf{w})\right\}$$

where the randomness is over the random variables used in Prove *and* Rand*.*

Note that this definition is just derivation-privacy for T_{id} with different semantics for convenience. The notion of proof rerandomization we use is similar to [Belenkiy et al., 2009] and the ciphertext rerandomization in [Lee et al., 2019]:

2.6.2 Simulation-Extractability and Limits of Malleability

In Chapter 3, we consider an important perspective on the security analysis of Groth16, namely the limits of its malleability (and non-malleability). The Groth16 NIZK is only randomizable, therefore only trivially updatable according to the definitions from the previous section ($T = T_{\mathsf{id}}$). However, these definitions we presented so far only model which updates are *possible*, but they do not capture which ones are *impossible*. In this section we capture this subtle yet important middle-ground.

Strong Simulation-Extractability. Arguably, the strongest soundness and non-malleability property for NIZKs is (strong) *simulation-extractability* (SE) [Sahai, 1999, De Santis et al., 2001], a security notion that extends knowledge-soundness (KS) by giving the adversary access to the simulation oracle. One of the important properties of this notion is that its straight-line extractable, black-box variant seems necessary to achieve universally composable (UC) security [Canetti, 2001] for non-interactive zero-knowledge (NIZK) proof systems, as shown by [Canetti et al., 2002, Groth et al., 2006, Groth, 2006]. This is an important practical concern since applications employing NIZKs often use the UC framework due to its flexibility and expressive power [Kosba et al., 2016, Kerber et al., 2019, Kerber et al., 2021b]. Second, SE is needed in game-hopping style proofs [Shoup, 2004] in which one game hop introduces the simulator and a subsequent game hop relies on extraction [Kosba et al., 2016, Camenisch et al., 2017]. Finally, the SE property allows us to capture non-malleability which is not possible with just knowledge-soundness.

Definition 2.6.12 (Simulation-Extractability). *A* NIZK *for* $\mathcal{R}$ *is* (strongly) simulation-extractable (SE) *if for any PPT adversary* $\mathcal{A}$ *there exists a PT extractor* $\mathsf{Ext}_{\mathcal{A}}$ *such that:*

$$\Pr\left[\begin{array}{l}(\boldsymbol{\sigma},\mathsf{td})\leftarrow\mathsf{Setup}(1^{\lambda})\\ (\mathsf{x},\pi)\leftarrow\mathcal{A}^{\mathcal{S}_{\boldsymbol{\sigma},\mathsf{td}}}(\boldsymbol{\sigma})\\ \mathsf{w}\leftarrow\mathsf{Ext}_{\mathcal{A}}(\mathsf{x},\pi)\end{array}\;:\;\begin{array}{l}\mathsf{Verify}(\boldsymbol{\sigma},\mathsf{x},\pi)=1\;\wedge\\ (\mathsf{x},\mathsf{w})\notin\mathcal{R}\wedge(\mathsf{x},\pi)\notin Q_{\mathcal{S}}\end{array}\right]=\mathsf{negl}(\lambda)$$

where $\mathcal{S}_{\boldsymbol{\sigma},\mathsf{td}}(\mathsf{x})$ *is the second variant in Fig. 2.2, a simulator oracle that calls* $\mathsf{Sim}(\boldsymbol{\sigma},\mathsf{td},\mathsf{x})$ *internally, and also records* (x,π) *into* $Q_{\mathcal{S}}$*.*

Let us examine what exactly does the property mean. First, it is easy to see that it implies KS, since in SE we only give $\mathcal{A}$ more freedom by providing it with oracle access. The main change is the winning condition: if the proof verifies, then either the extractor succeeds, or the proof was simulated (in which case extractor cannot succeed if the language is hard, because simulated proofs do not “contain” a witness).

Second, SE guarantees that a NIZK is in a certain sense “instance-binding”. Assume a certain game produces proofs $\{\pi_i\}$ for a language with a hard sublanguage, e.g. $\mathsf{x} = (\mathsf{x}_1, \mathsf{x}_2)$ and $\mathsf{x}_1 = f(\mathsf{w}_1)$ where f is a one-way function. Assume further that by zero-knowledge we give the adversary simulated proofs instead of the real ones. Then, if the NIZK is SE, $\mathcal{A}$ can only return either these simulated proofs for the *same* instance, or we can extract w from them (in which case the reduction essentially inverts f). This means that $\mathcal{A}$ cannot change the x_2 part in $\{\pi_i\}$; In practice this means that if the honest π hides certain secrets (a key, or a secret random value; something we do not expect $\mathcal{A}$ to obtain), then $\mathcal{A}$ cannot maul π into proofs for a “partially modified” instance.

A NIZK is called a *signature of knowledge* (SoK) [Chase and Lysyanskaya, 2006, Groth and Maller, 2017] if the proof possesses this instance-binding property. SE NIZKs are therefore automatically SoKs. By including m into the statement without asserting anything about m in the relation, we can turn every NIZK into, simultaneously, a signature (of knowledge) on m.

Third, strong SE proofs are non-malleable. Note that even if a proof π is merely randomisable, $\mathcal{A}$ could take a simulated proof π, randomise it into π', and pass it back to the game. In this case, it would (1) still verify, (2) extractor will fail (we cannot extract from simulated proofs), (3) $(x, \pi') \notin Q_{\mathcal{S}}$ holds. So $\mathcal{A}$ will win the game; thus randomisable NIZKs are not strong SE.

Simulation-extractability is relevant for both CRS-based and Random-Oracle (RO) based NIZKs. Faust et al. [Faust et al., 2012] show that NIZKs obtained from Σ-protocols using the Fiat-Shamir heuristic satisfy (strong) simulation-extractability in the ROM. This non-malleability result extends to other ROM NIZKs [Kohlweiss and Zając, 2021, Ganesh et al., 2022a], essentially implying that directly malleable NIZKs are only available in the standard/CRS model.

Weak Simulation-Extractability. In this work and particularly in Chapter 3, we focus on the weaker flavor [Kosba et al., 2015], that allows limited malleability of proofs in the form of randomizability. We achieve this by requiring the adversary to produce a proof for a statement that differs from any of the statements queried from the simulator.

Definition 2.6.13 (Weak Simulation-Extractability)**.** *A* NIZK *for* $\mathcal{R}$ *is* (weakly) simulation-extractable (weak SE) *if for any PPT adversary* $\mathcal{A}$ *there exists a PT extractor* $\mathsf{Ext}_{\mathcal{A}}$ *such that:*

$$\Pr\left[\begin{array}{l}(\boldsymbol{\sigma}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda)\\ (\mathsf{x}, \pi) \leftarrow \mathcal{A}^{\mathcal{S}_{\boldsymbol{\sigma},\mathsf{td}}}(\boldsymbol{\sigma})\\ \mathsf{w} \leftarrow \mathsf{Ext}_{\mathcal{A}}(\mathsf{x}, \pi)\end{array} \quad : \quad \begin{array}{l}\mathsf{Verify}(\boldsymbol{\sigma}, \mathsf{x}, \pi) = 1 \land\\ (\mathsf{x}, \mathsf{w}) \notin \mathcal{R} \land (\nexists \pi'.\, (\mathsf{x}, \pi') \in Q_{\mathcal{S}})\end{array}\right] = \mathsf{negl}(\lambda)$$

where $\mathcal{S}_{\boldsymbol{\sigma},\mathsf{td}}(\mathsf{x})$ *is the second variant in Fig. 2.2.*

The only thing we change is the $(\nexists \pi'.\, (\mathsf{x}, \pi') \in Q)$ condition, which we will sometimes write as $(\mathsf{x}, \cdot) \notin Q_{\mathcal{S}}$ or even $\mathsf{x} \notin Q_{\mathcal{S}}$

Weak SE and strong SE of *proof systems* can be seen as analogous to chosen message attack (CMA) and strong CMA unforgeability of *signatures*. Note that despite its name, weak SE is still a tremendously useful notion, since it models simulation-based security (and thus allows the NIZK to be used in the simulation-setting) without requiring non-malleability. In practice, many NIZKs are naturally exactly randomizable without allowing more fundamental instance malleability. In some cases, as we will show in Chapter 5, this randomizability is a desirable feature. Therefore, one *has to require weak SE* if one wants to use such NIZKs in the simulation-based setting.

We can also have a corresponding notion of weak simulation soundness, used in Section 3.4, which is implied by white-box and black-box weak simulation extractability.

Definition 2.6.14 (Weak Simulation-Soundness)**.** *A* NIZK *for* $\mathcal{R}$ *is* weakly simulation-sound *if for any PPT adversary* $\mathcal{A}$ *it holds that*

$$\Pr\left[\begin{array}{l}(\boldsymbol{\sigma}, \cdot) \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{x}, \pi) \leftarrow \mathcal{A}^{\mathcal{S}_{\boldsymbol{\sigma},\mathsf{td}}}(\boldsymbol{\sigma})\end{array} : \begin{array}{c}\mathsf{Verify}(\boldsymbol{\sigma}, \mathsf{x}, \pi) = 1 \wedge \\ \mathsf{x} \notin \mathcal{L}_{\mathcal{R}} \wedge \mathsf{x} \notin Q_{\mathcal{S}}\end{array}\right] = \mathsf{negl}(\lambda)$$

where $\mathcal{S}_{\boldsymbol{\sigma},\mathsf{td}}(\mathsf{x})$ *is the second variant in Fig. 2.2.*

Another important parameter of a SE notion especially in the context of SNARKs is whether it supports white-box (WB) or black-box (BB) extraction. In practice, the non-falsifiability of the assumptions used for SNARKs comes from their white-box nature; that is, they imply some knowledge of the adversary's internals. This prevents proving black-box extraction (and black-box SE), which requires extracting from the adversary only using its "input/output" interface. Since precisely this notion is required for standard UC security, in practice it is necessary to use a compiler lifting a zk-SNARK to black-box SE [Kosba et al., 2015, Atapoor and Baghery, 2019, Baghery, 2019], and, crucially, efficiency of these compilers can benefit from a stronger (white-box) property of the input SNARK as we show in Chapter 3. Note that assuming a *global random oracle* one can prove a SNARK secure in UC without losing its succinctness as shown by compiler of Ganesh et al. [Ganesh et al., 2022b].

Although black-box *strong* SE is sometimes a desirable property, (black-box) *weak* SE is sufficient for many UC applications, for instance in Hawk [Kosba et al., 2016], as argued in [Kosba et al., 2015]. Hawk uses SE NIZKs directly as a raw primitive (without employing a functionality), and it suggests to use a non-succinct strong SE NIZK, since no other candidates were known at that time. Kosba et al. [Kosba et al., 2015] point out that a weak SE NIZK can be used instead.

The black-box variant of weak SE specifies the existence of a single extractor that works for all adversaries.

Definition 2.6.15 (Black-box Weak Simulation-Extractability, [Kosba et al., 2015])**.** *A* NIZK *for* $\mathcal{R}$ *is* black-box weak SE *if there is an extractor* Ext *such that for any*

PPT adversary $\mathcal{A}$ and $\mathcal{R}_\lambda$,

$$\Pr\left[\begin{array}{c}(\boldsymbol{\sigma}, (\mathsf{td}, \mathsf{td}_{\mathsf{ext}})) \leftarrow \mathsf{Setup}(\mathcal{R}_\lambda) \\ (\mathsf{x}, \pi) \leftarrow \mathcal{A}^{\mathcal{S}_{\boldsymbol{\sigma},\mathsf{td}}}(\boldsymbol{\sigma}) \\ \mathsf{w} \leftarrow \mathsf{Ext}(\boldsymbol{\sigma}, \mathsf{td}_{\mathsf{ext}}, \mathsf{x}, \pi)\end{array} : \begin{array}{c}\mathsf{Verify}(\boldsymbol{\sigma}, \mathsf{x}, \pi) = 1 \wedge \\ (\mathsf{x}, \mathsf{w}) \notin \mathcal{R}_\lambda \wedge \mathsf{x} \notin Q\end{array}\right] = \mathsf{negl}(\lambda),$$

where $\mathcal{S}_{\boldsymbol{\sigma},\mathsf{td}}(\mathsf{x})$ is a simulator oracle that calls $\mathsf{Sim}(\boldsymbol{\sigma}, \mathsf{td}, \mathsf{x})$ internally, and also records x into Q.

Here we further split the trapdoor into two components, to highlight the fact that the extractor does not have to use the whole trapdoor. In practice our black-box compiled NIZKs will have an additional trapdoor, different from the original one; which $\mathsf{td}_{\mathsf{ext}}$ exactly models.

We also note that weak SE is sufficient for the SNARKs to signatures of knowledge (SoK) compiler of [Groth and Maller, 2017] that embeds a hash of the message into the statement proven. Thus applications employing SoK, such as [Bonneau et al., 2020a], can also benefit from our work. Note that in weak SE it is the statement rather than the proof that cannot be mauled. The resulting SoK satisfies CMA unforgeability.

Finally, it is important to note that the notion of simulation extractability can be significantly weakened to the case of arbitrary transformations. The two works of CKLM [Chase et al., 2012, Chase et al., 2013b], first of which was mentioned before, investigate the question of *controlled malleability*. This notion, first defined in [Chase et al., 2012], extends simulation-extractability by assuming the existence of an extractor that additionally to the witness may return if not the original instance, but another instance and a transformation that was applied to the proof. In other words, this definitions is a "controlled" variant of SE that allows the adversary to maul the proof by applying a transformation from a predefined set. A NIZK satisfying this condition is called controlled-malleable (cm-NIZK). Chase et al. show that it is possible to build a cm-NIZK for a set of transformation that are CM-friendly (essentially, that can be expressed using pairing equations), and this is by using GS malleability to perform the transformations over a specific, augmented variant of the original language. A recent work by Faonio et al. [Faonio et al., 2023] takes this approach even further. While investigating malleability of interactive oracle proof (IOP) based SNARKs and underlying commitments, it defines a generalised notion of Φ-flexible SE that captures malleability up to the predicate Φ, which strictly generalizes our weak SE notion.

2.6.3 SNARKs and Groth16

In terms of the interface, SNARKs are regular NIZKs; but in terms of instantiation and implementation — details that we will be focusing on in the first two main chapters — they require the introduction of additional background.

First, most SNARKs are general-purpose, and work with any NP languages; since general programs have many representations, we must talk about which one is actually used in practice. Within this work, we will focus on Groth16, and its arithmetisation called quadratic arithmetic program (or QAP for short).

Let $\mathcal{R}$ be a relation for an NP language $\mathcal{L}$. When $\mathcal{R}$ is implemented as an arithmetic circuit $\mathcal{C}$, we assume it to be of the following form. The input wires are split into: l public input wires corresponding to $\mathsf{x}_1, \ldots, \mathsf{x}_l$, and l_w private input wires, corresponding to $\mathsf{w}_1, \ldots, \mathsf{w}_{l_\mathsf{w}}$. We denote the total number of wires by m, and thus the remaining $m - l - l_\mathsf{w}$ wires are called intermediate — they can be computed from x and w.

A *quadratic arithmetic program* (QAP, [Gennaro et al., 2013]) for the circuit $\mathcal{C}$ is described by a tuple $\mathsf{QAP} = \big(\mathbb{Z}_p, \{u_i(X), v_i(X), w_i(X)\}_{i=0}^m, t(X)\big)$, which consists of the quotient polynomial $t(x)$ of degree n, and three sets of polynomials $\{u_i(X)\}_{i=0}^m$, $\{v_i(X)\}_{i=0}^m$ and $\{w_i(X)\}_{i=0}^m$ of degree $n-1$. A particular QAP assignment $\{\mathsf{a}_i\}_{i=0}^m$ contains assignments to the circuit wires, and $\mathsf{a}_0 = 1$ is a fixed parameter. We will refer to the sets $\{\mathsf{x}_i\} \cup \{\mathsf{w}_i\}$ and $\{\mathsf{a}_i\}$ interchangeably when there is no risk of confusion, with x_0 corresponding to a_0. The assignment $\{\mathsf{a}_i\}$ satisfies the QAP if and only if

$$\left(\sum_{i=0}^m \mathsf{a}_i u_i(X)\right)\left(\sum_{i=0}^m \mathsf{a}_i v_i(X)\right) - \left(\sum_{i=0}^m \mathsf{a}_i w_i(X)\right) = h(X)t(X)$$

for some $h(X)$ of degree $n-2$. That is, $t(x)$ divides the left hand side of the equation. Summing up, we can define the following *satisfiability* relation for QAP:

$$\mathcal{R}_{\mathsf{QAP}} = \left\{ (\mathsf{x}, \mathsf{w}) \;\middle|\; \begin{array}{l} \mathsf{x} = (\mathsf{a}_0 = 1, \mathsf{a}_1, \ldots, \mathsf{a}_\ell) \in \mathbb{Z}_p^{1+\ell}, \\ \mathsf{w} = (\mathsf{a}_{\ell+1}, \ldots, \mathsf{a}_m) \in \mathbb{Z}_p^{m-\ell}, \\ \exists h(X) \in \mathbb{Z}_p[X] \text{ of degree} \leq n-2 \text{ such that} \\ \left(\sum_{i=0}^m \mathsf{a}_i u_i(X)\right)\left(\sum_{i=0}^m \mathsf{a}_i v_i(X)\right) = \sum_{i=0}^m \mathsf{a}_i w_i(X) + h(X)t(X) \end{array} \right\}$$

As QAP relations are defined over a finite field that determines suitable bilinear groups, they need to be compatible with the desired security level λ. Our asymptotic security notions are all quantified over λ-compatible relations $\mathcal{R}$. In practice SNARK systems use very specific pre-defined groups for a fixed security level. For these reasons we elide most of these details in our formal modelling and typically write $\mathcal{R}$ instead of $\mathcal{R}_\lambda$.

Groth16 [Groth, 2016] is the SNARK with the smallest proof size and fastest verifier in the literature, and it is also competitive in terms of prover time. We present Groth16 in Fig. 2.3. The QAP arithmeticised relation is an implicit input to Setup, and CRS is circuit-dependent and linear in the QAP size. Proving is linear in the size of the QAP, verification is linear in instance size, and proof size is just three group elements. The intuition behind soundness of Groth16 is quite complex to be presented here. The proof ultimately relies on the fact that CRS has a concrete structured form, and certain elements (e.g. cross-products between trapdoor variables) are not available; this forces proof elements A, B, C to only be in a certain form, like in the honest proof, and thus to encode parts of the QAP equation. We refer to the original paper for more details.

Beyond efficiency, Groth16 has several other useful properties. It is rerandomizable, as we will elaborate on in Section 3.3, which is a desirable property for achieving receipt-free voting [Lee et al., 2019]. Simultaneously, it also has a weak form of simulation extractability which guarantees that even if the adversary has seen some proofs before, it cannot prove a new statement without knowing the witness. The prover and verifier use only algebraic operations and thus proofs can be aggregated [Bünz et al., 2021b]. Furthermore, Groth16 is attractive to practitioners due to the vast quantity of implementation and code auditing it has already received.

2.7 Distributed Ledgers

In Section 1.1 we gave a general account of distributed ledger technologies, including privacy-oriented solutions. In this section we will elaborate further, in particular highlighting Zcash [Hopwood et al., 2022], which our ideas in Chapter 5 are closely related to.

To simplify notation we denote $q_i(\alpha, \beta, x) := \beta u_i(x) + \alpha v_i(x) + w_i(x)$ and $y_i(\alpha, \beta, \gamma, x) := q_i(\alpha, \beta, x)/\gamma$, and use it as $q_i(x)$ and $y_i(x)$ omitting other variables when it is clear from the context.

$\underline{\mathsf{Setup}(1^\lambda)}$:

1. $\mathsf{td} := x, \alpha, \beta, \gamma, \delta \xleftarrow{\$} \mathbb{Z}_p^*$
2. $\boldsymbol{\sigma}_1 \leftarrow \left[\alpha, \beta, \delta, \{x^i\}_{i=0}^{n-1}, \left\{\frac{x^i t(x)}{\delta}\right\}_{i=0}^{n-2}, \{y_i(x)\}_{i=0}^{l}, \left\{\frac{q_i(x)}{\delta}\right\}_{i=l+1}^{m}\right]_1$
3. $\boldsymbol{\sigma}_2 \leftarrow \left[\beta, \gamma, \delta, \{x^i\}_{i=0}^{n-1}\right]_2$
4. **return** $(\boldsymbol{\sigma} = \boldsymbol{\sigma}_1 \cup \boldsymbol{\sigma}_2, \mathsf{td})$

$\underline{\mathsf{Prove}(\boldsymbol{\sigma}, \mathsf{x} = \mathsf{x}_1 \ldots \mathsf{x}_l, \mathsf{w} = \mathsf{w}_1 \ldots \mathsf{w}_{m-l})}$:

1. Let $\mathsf{a} := \mathsf{x} \| \mathsf{w}$
2. $r_a, r_b \xleftarrow{\$} \mathbb{Z}_p^*$
3. $[a]_1 \leftarrow \left[\alpha + \sum_{i=0}^{m} \mathsf{a}_i u_i(x) + r_a \delta\right]_1; \ [b]_2 \leftarrow \left[\beta + \sum_{i=0}^{m} \mathsf{a}_i v_i(x) + r_b \delta\right]_2$
4. $[c]_1 \leftarrow \left[\sum_{i=l+1}^{m} \mathsf{a}_i \frac{q_i(x)}{\delta} + \frac{h(x)t(x)}{\delta} + a r_b + b r_a - r_a r_b \delta\right]_1$
5. **return** $([a]_1, [b]_2, [c]_1)$

$\underline{\mathsf{Verify}(\boldsymbol{\sigma}, \mathsf{x} = \mathsf{x}_1 \ldots \mathsf{x}_l, \pi = (a, b, c))}$:

1. **assert** $\hat{e}(a, b) = \hat{e}([\alpha]_1, [\beta]_2) + \hat{e}(\sum_{i=0}^{l} \mathsf{x}_i [y_i(x)]_1, [\gamma]_2) + \hat{e}(c, [\delta]_2)$

$\underline{\mathsf{Sim}(\mathsf{td}, \mathsf{x} = \mathsf{x}_1 \ldots \mathsf{x}_l)}$:

1. $\mu, \nu \xleftarrow{\$} \mathbb{Z}_p^*$; **return** $\left([\mu]_1, [\nu]_2, \left[\frac{\mu\nu - \alpha\beta - \sum_{i=0}^{l} \mathsf{x}_i q_i(x)}{\delta}\right]_1\right)$

$\underline{\mathsf{Rand}(\boldsymbol{\sigma}, \pi = (a, b, c))}$:

1. $r_1, r_2 \xleftarrow{\$} \mathbb{Z}_p^*; \ a \mapsto (1/r_1) a; \ b \mapsto r_1 b + r_1 r_2 [\delta]_2; \ c \mapsto c + r_2 a$
2. **return** (a, b, c)

Figure 2.3: Groth16 zk-SNARK with simulation and randomization procedures.

2.7.1 Privacy in Ledgers

Blockchain protocols started off with none or minimal privacy guarantees, often providing users merely pseudonyms. This pseudonymity is generally illusory and disappears under scrutiny and targeted analysis [Reid and Harrigan, 2011, Ron and Shamir, 2013, Androulaki et al., 2013, Meiklejohn et al., 2016, Béres et al., 2021]. One of the first and, perhaps, most direct approaches to achieve privacy in otherwise public systems is to involve services called *mixers* or *tumblers* [Maxwell, 2013, Ruffing et al., 2014, Bonneau et al., 2014, Heilman et al., 2017, Ruffing and Moreno-Sanchez, 2017, Meiklejohn and Mercer, 2018, Liang et al., 2022], that forward funds pseudorandomly between pseudonymous entities, thus making the transaction graph intangible against correlation analysis. This approach works quite well in other security areas; the onion routing protocol [Dingledine et al., 2004] implemented in the Tor network is one of the prime examples.

A private cryptocurrency is a cryptocurrency designed to takes privacy aspects into consideration on the most foundational layer, instead of trying to graft them on top of existing solution in an ad-hoc manner. Popularised by the Zcash [Miers et al., 2013, Hopwood et al., 2022] and Monero [Noether et al., 2016] protocols, which still exist and are developed to this day as cryptocurrencies, the field has suggested many improvements and follow-ups [Lai et al., 2019, Fauzi et al., 2019, Bünz et al., 2020a, Kerber et al., 2019]. The basic idea is quite commonly similar — instead of having a public ledger with pseudonymous addresses, as in Ethereum or Bitcoin, the individual banknotes or account balances are published in a private way, e.g. using cryptographic commitments or ciphertexts. Then, commonly, some other alternative cryptographic mechanism must be used to prove publicly that a transaction involving blinded values is correct. In the case of Zcash, a NIZK is used; in the case of Monero, a so-called ring signature, which hides the real transaction input among many other fake ones; in other solutions, homomorphic encryption can be used for this task.

Lastly but perhaps most importantly, recent advancements in efficient NIZKs seem to bring us closer to the notion of *private smart contracts*[Kosba et al., 2016, Steffen et al., 2019, Bünz et al., 2020a, Bowe et al., 2020, Kerber et al., 2021b, Steffen et al., 2022]. These systems, albeit complex to analyse and still challenging to implement in a practical manner, allow to create smart contracts where privacy can be programmatically controlled within the smart contract language. This is undoubtedly a great engineering achievement — while private cryptographic protocols usually require a lot of expertise to develop, analyse, and implement, private smart contracts create a convenient abstraction layer that can be approached by software developers with less cryptographic background. This in turn will definitely have a similar ripple effect on the market, bringing more expertise generally into the area.

Zcash. Among these solutions, in this work, however, we will be focusing mostly on the variant of zcash [Hopwood et al., 2022], which allows only basic private transfers. Zcash protocol was originally implementing a variant of the Zerocoin [Miers et al., 2013] paper, but since then the protocol has seen several iterations of improvement. In Chapter 5 we will be mostly working with *Zcash Sapling*, a release which precedes the current most-recent version *Zcash Orchard*. Sapling differs from the Zerocoin paper and earlier Zcash releases in a few important ways, and it is the variant we will describe here (in a simplified manner). We abstract some parts of zcash in Section 5.3 that we need in Chapter 5.

In Zcash Sapling, the main data structure carrying information about a coin is called a note note; it contains the coin value, the public key pk of the owner, randomness, and other values we omit for simplicity. The ledger maintains two main data structures: the set of *nullifiers* Nf and the Merkle tree MT of *note commitments* $C(\mathsf{note})$. A *nullifier* nul is a pseudorandom string. Given a secret key sk, there is only one unique way to derive a pseudorandom nullifier nul from a given note; it is not possible to derive nul without sk corresponding to owner's $\mathsf{pk} \in Note$. Each transaction consumes and creates notes. The nullifier of each consumed note is published into Nf, and commitments to new notes are appended into MT. Since no note can give rise to two distinct nullifiers, by asserting that $\mathsf{nul} \notin \mathsf{Nf}$ when spending notes we guarantee that double-spending does not occur.

A transaction consists of input nullifiers nul_i, output note commitments $C(\mathsf{note})$, and transaction consistency proofs. The latter in turn consists of: (1) an input NIZK for each input nullifier, (2) an output NIZK for each output note commitment, (3) a value commitment com_i for each input and output, (4) a Schnorr-like binding signature. Each input NIZK proves that there exists the note in the current MT, and its nullifier is derived correctly. Each output NIZK shows that the commitment contains a valid note. Each value commitment $\mathsf{com}_i^{\mathsf{in}}$ and $\mathsf{com}_i^{\mathsf{out}}$ is a homomorphic Pedersen commitment to the value inside the (input or output) note. In addition each input and output NIZK shows that the corresponding commitment is well-formed and contains the claimed value. Finally, the binding signature proves that the homomorphic sum $\sum \mathsf{com}_i^{\mathsf{in}} / \sum \mathsf{com}_i^{\mathsf{out}}$ is a commitment to the zero value, which it is only when the transaction is well-balanced (sum of inputs is equal to sum of outputs).

To construct a transaction, one must generate all the $\mathtt{nul}_i$ for coins that one wants to spend; create output notes $\mathtt{note}_i$ to the public keys that will receive a coin, and commit to them; and finally generate proofs and a signature. It is easy to see that given that proofs are ZK, nullifiers are pseudorandom, and commitments are hiding, no information about inputs or outputs is revealed. Showing soundness is harder, but it ultimately relies on NIZKs and the signature guaranteeing the transaction balancing property and validity of inputs/outputs, and impossibility of double-spending due to the nullifier design. To verify a transaction, one must verify all the NIZKs, check $\mathtt{nul}_i \notin \mathsf{Nf}$ for each $\mathtt{nul}_i$, and check the signature.

The homomorphic check and NIZK per input and output is a Sapling feature that was not present in the previous releases, and that we will actively use in Chapter 5.

2.7.2 Asset Exchange in Private Ledgers

Surprisingly little academic work *directly* addresses exchange of multiple assets in a private ledger. Undoubtedly, such a functionality can be achieved through certain generic private smart contract solutions [Kosba et al., 2016, Kerber et al., 2021b, Steffen et al., 2019], but their flexibility comes with a non-negligible performance overhead, since they often require heavy primitives like SNARKs over big contract code-dependant (or even universal) circuits. The performance of universal zero-knowledge based constructions such as ZEXE [Bowe et al., 2020] requires minutes of proving time for ten times the constraints compared to our one second prover runtime. This is why we would like to consider systems with such a functionality embedded directly.

Several solutions take the route of extending the vanilla Zerocash protocol. Ding et al. [Ding et al., 2019] propose a solution supporting multiple assets, but with no exchange mechanism, and with public asset types. Gao et al. [Gao et al., 2019] construct a transaction system specifically for exchanging assets which is based on storing debt in sibling notes which are only spendable if the debt is settled. Its inefficiency results from requiring multiple persisted transactions per swap.

On the other hand, Confidential Assets [Poelstra et al., 2019] use a commitment construction that hashes an asset type descriptor to the bases of an extended Pedersen commitment such that the resulting commitments are additively homomorphic, which facilitates proving the balancing of amounts. We use this sparse commitment scheme as part of our construction. A similar solution by Zheng et al. [Yi et al., 2019] exists for the Mimblewimble private cryptocurrency. Both these works do not provide sender and receiver anonymity.

Regular homomorphic commitments have the drawback that they can only store type-amount pairs. A more flexible approach is to use a hash-based commitment scheme for notes to store a vector of attributes. This is used in Zcash's multi-asset ZIP 220, Shielded Assets[2], or similarly MASP[3]. To achieve balancing, these protocols prove equality between the type-value pair of a note and a sparse homomorphic commitment. In Chapter 5 we follow the same approach. Another system that *does not* rely on homomorphic commitments is Stellar[4], which instead uses shuffle proofs.

Finally, some works emphasize the exchange and offer matching functionalities. Manta [Chu et al., 2020] describes a privacy-preserving decentralized exchange (DEX) based on an automated marked maker (AMM) scheme which works without a second party but does not hide the types, which is an inherent limitation of the AMM approach. Another idea is to privately exchange assets between different systems in cross-chain atomic swaps [Deshpande and Herlihy, 2020].

In contrast to both, in Chapter 5 we propose a more basic mechanism within a single blockchain, and leave it open to implement the concrete offer matching algorithm on top of Zswap. This allows, and will likely enable more powerful DeFi applications, since Zswap provides a more flexible interface to the application layer.

[2] Previously "user-defined assets" (UDA) or UIT, see ZIP 220: `https://github.com/zcash/zcash/issues/830`

[3] `https://github.com/anoma/masp/blob/main/docs/multi-asset-shielded-pool.pdf`

[4] `https://github.com/stellar/slingshot/blob/main/spacesuit/spec.md`

2.8 Privacy for Accountability

Data protection often demands that the actions and personal information of individual users are kept private, but that regulatory organizations can learn about undesirable events that could be triggered by the combined actions of multiple users. In such scenarios requiring both privacy and accountability, cryptographic techniques such as zero knowledge proofs and multiparty computation come to mind as potential solutions. While such techniques may trivially reconcile the paradox of privacy vs. accountability, they often impose performance overheads and trust assumptions that are incompatible with the original scenario. As a motivating example, we consider the problem of implementing *anti-money laundering* (AML) policies in both traditional and decentralized financial systems.

Privacy and accountability in AML. Money laundering is the process of concealing the origins of money procured through illegitimate or illegal sources, by changing its origin to one considered legitimate. This is required as most suppliers of legitimate and expensive goods, such as property or cars, do not accept large amounts of cash, but rather require the money coming from an account in a legitimate financial institution. Money laundering is highly prevalent globally and is estimated to constitute 2-3% of the national GDP in the US alone, excluding tax evasion [Reuter and Truman, 2004, Chap. 2].

The framework of combating money laundering is known as anti-money laundering and currently the main tool used by banks to counter money laundering is a *suspiciousness* score associated with each account [Baum et al., 2023]. The score is computed from a *base* score derived from private meta information about the account and its owner. The score is then updated based on incoming and outgoing transfers, using as reference a *grey* list of *potentially* illegitimate or suspicious customers and accounts. The grey list is secret, but known by all banks (at least within the same jurisdiction). At certain time-intervals the updated suspiciousness scores are checked, and if it is above a certain threshold, the account and its transfers will get manually inspected. Afterwards the suspiciousness score is reset to its base-score (although it can now be adjusted based on the inspection outcome). Unfortunately, this currently has a very high false-positive rate. Furthermore, while banks are legally obliged to perform AML and report accounts they believe are involved in money laundering, they must also be able to *explain* to the authorities why they believe an account is used for illicit activities.

How cryptography can help. Multiparty computation [Goldreich et al., 1987, Chaum et al., 1988] (MPC) is a natural solution to precisely computing suspiciousness scores for transactions among different banks without undermining user privacy. All banks involved in a sequence of transactions could execute an efficient MPC protocol (*e.g.*, [Damgård et al., 2012]) to compute a joint suspiciousness score based on all of their private information on their own clients while only revealing the final score. However, it is unrealistic to require all banks to interact with each other at once, as opposed to only interacting when receiving or sending transactions. Moreover, processing the sheer volume of transactions generated in the traditional financial system via MPC would prove prohibitively expensive.

In the context of decentralized finance (*i.e.* based on cryptocurrencies), systems with no privacy guarantees make it easier to perform AML by analyzing public transaction graphs [Meiklejohn et al., 2016]. On the other hand, privacy preserving cryptocurrencies (*e.g.* [Ben-Sasson et al., 2014a, Fauzi et al., 2019]) completely preclude the use of AML techniques, as they hide all information about transactions. Solutions based on revocable anonymity [Camenisch et al., 1996] have been proposed for both the permissioned setting [Narula et al., 2018, Androulaki et al., 2020, Kiayias et al., 2022], where authorities control the cryptocurrency, and for the permissionless setting [Damgård et al., 2021], where the cryptocurrency is controlled by independent users. However, in order to perform AML checks via these schemes, *an auditor must learn all information about all transactions sent or received by a user*, severely undermining their privacy.

Our “exploding commitment scheme” solution in Chapter 6 provides a building block similar to the recent work introducing a “privacy-preserving blueprint scheme” [Kohlweiss et al., 2023]. The ECS protocol requires quite minimal interaction, compared to general MPC, and it similar to homomorphic commitments in nature — we allow parties to perform a certain homomorphic computation, which can be done sequentially, so that a chosen regulator party can learn only the result of a chosen “explosion” predicate on the value. Compared to the DeFi solutions, ECS does not *require* a ledger, being an independent cryptographic block on its own. But it can be integrated into any bulletin board, a functionality that ledgers implement, thus giving rise to a more complicated protocol allowing more accountability on behalf of the parties, without violating their privacy. We explain this use-case in Section 6.6.

Chapter 3

Extraction and Malleability in Groth16

This chapter is based on the work "Another Look at Extraction and Randomization of Groth's zk-SNARK", published in Financial Cryptography 2021, and co-authored by Karim Baghery, Markulf Kohlweiss, and Janno Siim.

In this chapter we investigate a relaxed weaker notion of simulation-extractability, that *allows proof randomization*, while guaranteeing statement non-malleability, which we argue to be a more natural security property. In Section 2.6.2 we gave a summary of simulation-extractability together with this randomizable counterpart, weak SE, and justified why its study and use is desirable, especially in the simulation-based contexts. Contributions of this section are twofold. First, we show that it is already achievable by Groth16, arguably the most efficient and widely deployed SNARK nowadays. Second, we show that because of this, Groth16 can be *efficiently* transformed into a black-box weakly SE NIZK, which is sufficient for UC protocols.

Surprisingly, the fact that Groth16, as described in the literature and deployed in practical applications, is already white-box weak SE, was not known before. Proof malleability was noted by [Groth and Maller, 2017] as an obstacle for proving the strong SE property for Groth16, which resulted in them constructing a new non-malleable SNARK. Allowing proof randomization in the definition resolves the issue differently by proving a security property for the original system that lies in strength between knowledge soundness and strong SE. Additionally, we show that only a specific type of proof malleability is possible and that rerandomized proofs have the same distribution as fresh proofs of the same statement. We show in the algebraic group model (that we state as an assumption) that the extractor can either obtain the witness or point to the unique simulated proof that was randomized to obtain the proof produced by the adversary. Thus, even if the adversary queries multiple proofs for the same statement, it cannot combine them into a new proof of the same statement, which is the main technical challenge in proving white-box weak SE.

As of our second contribution, we give two optimized constructions for black-box weak SE: Int-Groth16 and Ext-Groth16. Int-Groth16 is based on the (strong) WB-to-BB SE compiler of [Baghery, 2019]. It adds a public key of a cryptosystem to the CRS and a ciphertext of the witness to the proof. It then employs the SNARK to prove an extended statement to ensure that the witness is correctly encrypted. We show that this compiler can be used for *weak* WB-to-BB conversion, and therefore instantiated with Groth16.[1] We optimize the encryption scheme and employ a SNARK-friendly variant of ElGamal with randomness reuse [Kurosawa, 2002]. A noteworthy technical detail is that the witness needs to be mapped to SNARK-friendly elliptic curve points. The downside of this construction is that even state-of-the-art SNARK-friendly public-key operations incur a substantial overhead in the circuit size.

Ext-Groth16 uses a verifiable encryption technique of Lee et al. [Lee et al., 2019] to overcome this limitation. We again encrypt the witness, but with a different encryption scheme in which resulting ciphertexts enter Groth16 verification equation directly and thus have almost no effect on the circuit structure. To show Ext-Groth16 secure, we need to directly prove black-box weak simulation-extractability, which we do by a reduction to white-box weak SE of Groth16. The main technical challenge is, again, to show which types of malleabilities are available to the adversary. Additionally, we prove that the zero-knowledge property of Ext-Groth16 can rely on the standard Decisional Diffie-Hellman assumption rather than the novel assumption stated in [Lee et al., 2019].

To compare the efficiency of these two constructions, we estimate CRS and proof size, prover time, and verifier time as a function of the encrypted witness size. Our results show that both constructions have low overhead compared to the com-

[1] In fact, even weak simulation soundness without extractability is sufficient for the compiler.

monly used generic transformations. In particular, Ext-Groth16 leads to almost no increase in CRS size and prover time, while resulting in slightly bigger proofs and verification time.

Related work on SE. White-box SE SNARKs have been discovered only recently. Groth and Maller [Groth and Maller, 2017] presented the first construction in 2017, targeting the language of Square Arithmetic Programs (SAPs). They also proved a lower bound of three group elements for the proof size and two verification equations for all *non-interactive linear proof* (NILP) based SNARKs, which covers many previously known pairing-based SNARKs. Weak SE allows us to go below this bound with a single verification equation.

Bowe and Gabizon [Bowe and Gabizon, 2018] give a RO-based variant of Groth16 for Quadratic Arithmetic Programs (QAPs) that is simulation-extractable, and has five group elements and two verification equations. Baghery, Pindado, and Rafols [Baghery et al., 2020] improve on this approach by substituting RO with a collision-resistant hash function, while preserving roughly the same efficiency. Lipmaa [Lipmaa, 2019] presents a different technique that allows to construct SE SNARKs for QAP and the three other arithmetization techniques from the QAP family (namely, SAP, SSP, and QSP). Kim, Lee, and Oh [Kim et al., 2019] present a SE SNARK for QAP with three elements but just a single verification equation, avoiding the lower bound of Groth and Maller by using a RO in addition to a knowledge extraction assumptions and a CRS.

As of black-box NIZKs, a generic transformation that makes ordinary NIZKs black-box SE has been known at least since [De Santis et al., 2001]. Along this direction, Kosba et al. [Kosba et al., 2015] extend, analyse, and optimize this transformation technique — they present three transformations; two of which build weak SE NIZKs, while the third builds a strong SE NIZKs. Atapoor and Baghery [Atapoor and Baghery, 2019] adapt Kosba et al.'s work directly to Groth16 and evaluate the efficiency of the resulting strong SE argument. Baghery [Baghery, 2019] analyses a transformation from white-box SE to black-box SE, and instantiates it with the strong SE SNARK by Groth and Maller. We show that this technique also works for lifting white-box *weak* SE to black-box *weak* SE. Other generic transformations take into account CRS subversion and updatability [Abdolmaleki et al., 2020, Baghery and Sedaghat, 2021].

3.1 Approaching SNARK Soundness Algebraically

As a warm-up, in this more technical section we start by presenting a lemma that we will use for proofs in Chapters 3 and 4. Intuitively, it shows that in a typical security proofs (SNARK soundness proofs) against algebraic adversaries, one can view the verification equation as a polynomial equality test where trapdoors (created during CRS generation) are substituted by indeterminates.

We assume that SNARK CRS generation algorithm can be expressed as a two-step sampling procedure $S_\lambda = (\mathcal{D}_\lambda, \mathsf{Setup}_\lambda)$, where an effectively sampleable distribution $\mathcal{D}_\lambda$ defines a set of trapdoors $\mathsf{td} \in (\mathbb{Z}_p^*)^n$, and a polynomial time deterministic procedure $\mathsf{Setup}_\lambda(\mathsf{td})$ generates elements in $\mathbb{G}_1$ and $\mathbb{G}_2$ as polynomials of td. Let $\boldsymbol{T} = T_1, \dots, T_n$ be a set of formal variables corresponding to the trapdoors. In other words, Setup_λ constructs two sets of elements $\boldsymbol{\sigma}_1$ and $\boldsymbol{\sigma}_2$, where every $\boldsymbol{\sigma}_{\iota,i} = P_{\iota,i}(\mathsf{td})$ for some set of polynomials $\{P_{\iota,i}(\boldsymbol{T})\}_{\iota,i}$.

Lemma 3.1.1 (Algebraic Verification Satisfiability). *Let $\boldsymbol{E} = (E_{1,1}, \dots, E_{1,m_1}, E_{2,1}, \dots, E_{2,m_2})$ be a vector of formal variables in $\mathbb{Z}_p$, where $E_{\iota,i}$ represents an exponent value of some $[E_{\iota,i}]_\iota \in \mathbb{G}_\iota$. Let $V(\boldsymbol{E})$ be a pairing equation, expressed in the $\mathbb{G}_T$ exponent*[2]*.*

*Then, assuming (d_1, d_2)**-dlog** holds, for all algebraic PPT $\mathcal{A}$, and all two-step sampling procedures S_λ with trapdoor variables $\boldsymbol{T}$:*

$$\Pr\left[\begin{array}{c} \mathsf{td} \xleftarrow{\$} \mathcal{D}_\lambda; (\boldsymbol{\sigma}_1\ \boldsymbol{\sigma}_2) \leftarrow \mathsf{Setup}_\lambda(\mathsf{td}); \\ [\boldsymbol{e}]_{1,2} \xleftarrow{\$} \mathcal{A}([\boldsymbol{\sigma}_1]_1, [\boldsymbol{\sigma}_2]_2); \\ K \leftarrow \mathsf{Ext}_{\mathcal{A}}^{\mathit{alg}}([\boldsymbol{\sigma}_1]_1, [\boldsymbol{\sigma}_2]_2, \mathsf{view}_{\mathcal{A}}) \end{array} : \begin{array}{c} V(\boldsymbol{e}) = 0\ \wedge \\ V\big(K(\mathsf{Setup}_\lambda(\boldsymbol{T}))\big) \neq 0 \end{array}\right] = \mathsf{negl}(\lambda)$$

where $d_\iota = \max_i(\deg(P_{\iota,i}(\boldsymbol{T})))$ of Setup_λ, and $V\big(K(\mathsf{Setup}_\lambda(\boldsymbol{T}))\big)$ stands for $V(\boldsymbol{e})$ interpreted as a polynomial over $\boldsymbol{T}$. The probability is quantified over $\mathcal{D}_\lambda$ and the private coins of $\mathcal{A}$.

In other words, the lemma says that $\mathcal{A}$ has negligible success in constructing $\boldsymbol{e}$ as linear combination of CRS elements such that $V(\boldsymbol{e})$ evaluates to zero, but $V'(\vec{T}) = V(K \cdot \mathsf{Setup}_\lambda(\boldsymbol{T}))$ is not identically zero as a polynomial in $\boldsymbol{T}$.

Proof of Lemma 3.1.1 (Sketch). The intuition for the lemma is that since CRS trapdoors are chosen uniformly, and are "hidden" in the group exponents (hence the

[2] That is, $V(\boldsymbol{E}) = \sum_i \Gamma_i t_{1,i} t_{2,i}$ for $t_{\iota,i}$ being either some $E_{\iota,i}$ or a constant from $\mathbb{Z}_p^*$, and $\Gamma_i \in \mathbb{Z}_p^*$. This corresponds to the base group elements pairing equation $\prod_i \hat{e}(z_{1,i}, z_{2,i})^{\Gamma_i} = 1$ with $z_{\iota,i}$ being either variable or constant group elements $[t_{\iota,i}]_\iota$.

discrete log assumption), $\mathcal{A}$ combines $\boldsymbol{e}$ as if it has no knowledge of the internal structure of the CRS, and thus this is equivalent to choosing the V', and then evaluating it on random $\boldsymbol{T}$ (reversed order), which is negligible by S-Z. For the detailed proof of a similar statement tailored specifically for Groth16 in AGM, see [Fuchsbauer et al., 2018]. Here we present a sketch of the proof that is slightly more general, and can also be applied to other NILP based SNARKs, e.g. to Groth and Maller SNARK.

The original generic algebraic verification game has the step $[\boldsymbol{e}]_{1,2} \xleftarrow{\$} \mathcal{A}(\boldsymbol{\sigma}); K \leftarrow \mathsf{Ext}^{\mathsf{alg}}_{\mathcal{A}}(\mathsf{view}_{\mathcal{A}})$, where K is a matrix of algebraic coefficients. We modify the game, launching $\mathcal{A}$ also on another independently generated CRS and $\boldsymbol{\xi}$ — we can do that since we know K, essentially "how $\boldsymbol{e}$ was constructed from $\mathtt{td}$", so we just replace the trapdoors and emulate the execution of $\mathcal{A}$. If verification passes on both CRSs, it means that $\mathcal{A}$ constructed its proof $\pi = [\boldsymbol{e}]_{1,2}$ independently of the concrete CRS structure, and otherwise he has used it in proof construction.

We split the game in two scenarios according to the result of this test: either (i) $\mathcal{A}$ does not use the *concrete* CRS and returns coefficients blindly (then we arrive at the main positive lemma statement), or (ii) it uses the CRS, thus we break the (d_1, d_2)**-dlog** assumption.

The first option is that $\mathcal{A}$ succeeded without using the concrete CRS $\boldsymbol{\sigma}$ — meaning that it guessed $\boldsymbol{c}_{\iota,i}$ as if it only knew the structure of the CRS (Setup_λ and all $P_{\iota,i}$, but not the concrete $\boldsymbol{\sigma}_i$ themselves). Then the probability for $\mathcal{A}$ to win is low and bounded by S-Z lemma, since the unknown $\mathtt{td}$ for $\mathcal{A}$ is equivalent to the randomly chosen one — we can generate the concrete CRS after the call to $\mathcal{A}$. By S-Z we know that $\Pr_{\boldsymbol{e} \leftarrow \mathcal{A}(\dots)}[V(\boldsymbol{e}) = 0 \mid V'(\boldsymbol{T}) \neq 0] < \mathsf{negl}(\lambda)$ where $V'(\boldsymbol{T}) = V\big(K(\mathsf{Setup}_\lambda(\boldsymbol{T}))\big)$, and we also assume that $\Pr[V(\boldsymbol{e}) = 0] = p(\lambda)$ is non-negligible, which means that V can be satisfied by a prover. Then:

$$\Pr[V'(\boldsymbol{T}) \neq 0 \mid V(\boldsymbol{e}) = 0] = \frac{\mathsf{negl}(\lambda) \cdot \Pr[V'(\boldsymbol{T}) \neq 0]}{p(\lambda)} = \mathsf{negl}(\lambda)$$

So in the end we arrive at the conclusion that $V'(\boldsymbol{T}) = 0$ in case $V(\boldsymbol{e}) = 0$ with high probability.

The other option is that $\mathcal{A}$ has used the CRS non-trivially, possibly extracting knowledge about the trapdoor, which allowed it to satisfy the verification equation. Formally, $\mathcal{A}$ constructed $\boldsymbol{e}$ such that $V'(\boldsymbol{T}) \neq 0$, but $V'(\mathtt{td}) = V(\boldsymbol{e}) = 0$ for $\mathtt{td}$ being a concrete trapdoor. Then we can embed (d_1, d_2)-**dlog** instance $([z]_\iota, [z^2]_\iota, \ldots, [z^{d_\iota}]_\iota)$ into the CRS before generation (by using the challenge to generate trapdoors) and solve it. We embed by transforming the challenge into CRS trapdoors td $= \{\tau_i\}_{i=1}^n$ in the following way: $[\tau_i]_\iota = [\alpha_i z + \beta_i]_\iota$ for random (α_i, β_i), and then $[\tau_i^j]_\iota = [(\alpha_i z + \beta_i)^j]_\iota$ is a polynomial in z will all known coefficients, so it can be constructed from the q-**dlog** challenge higher powers. Then, after $\mathcal{A}$ returns $\boldsymbol{e}$ that depends on this particular CRS $\boldsymbol{\sigma}$ with z embedded inside, and satisfies $V(\boldsymbol{e}) = 0$, we factor $V'(\boldsymbol{T})$, reconstructed using K, and reinterpreted as a single variable polynomial over z (since in fact it is parameterized only by one unknown z, and we know all of the other coefficient of this equation except for z), and then one of the roots of this $V'(z)$ will be a solution to the discrete log challenge. $\square$

The lemma is defined with respect to positive powers polynomials, while Groth16 CRS is defined for Laurent polynomials. This obstacle is easy to overcome — as shown in [Fuchsbauer et al., 2018], it is enough to modify the group generator by raising it to a certain trapdoor power such that all the negative powers cancel out. This does not change the main statement of Lemma 3.1.1, although it slightly increases the required degree of (d_1, d_2)-**dlog**[3].

It is also not hard to generalize this statement for an adversary $\mathcal{A}$ that also obtains some group elements through queries to oracles, or for multiple equations that $\mathcal{A}$ aims to satisfy.

3.2 White-box Weak SE of Groth16

In this section, we show that Groth16, as defined in Fig. 2.3, is white-box weakly simulation extractable, which to our knowledge is the first SNARK construction that is proved to (only) achieve this notion. Additionally, we provide some facts about randomization of Groth16.

Our proof is in the AGM and relies on the same hardness assumptions ((q_1, q_2)-discrete logarithm) as Groth16 knowledge soundness. Additionally we require a form of linear independence from QAP polynomials — a similar requirement was used for square arithmetic programs in [Groth and Maller, 2017].

[3] In case of Groth16, we multiply by $\gamma\delta$, thus $[x^{n-2}t(x)/\delta]_1$ becomes $[\gamma x^{n-2}t(x)]_1$ of degree $2n-1$, hence $d_1 = 2n-1$.

Theorem 3.2.1. *Assume that* $\{u_i(x)\}_{i=0}^{l}$ *are linearly independent and* $\mathrm{Span}\,\{u_i(x)\}_{i=0}^{l} \cap \mathrm{Span}\,\{u_i(x)\}_{i=l+1}^{m} = \emptyset$. *Then Groth16 achieves weak white-box SE against algebraic adversaries under the* $(2n-1, n-1)$***-dlog*** *assumption.*

The proof of the theorem splits in two branches — we show that either $\mathcal{A}$ uses simulated elements, and in this case it can only use them for a single simulation query k, or it does not use them at all. In particular, this implies that $\mathcal{A}$ cannot combine several elements from different queries algebraically for the π it submits. We then argue that the non-simulation case reduces to knowledge soundness, and in the simulation case we show that $\mathcal{A}$ supplies ϕ that is equal to one of the simulated instances, which proves that $\mathcal{A}$ reuses a simulated proof, potentially randomized. An interesting detail not captured in the weak SE definition is that not only can we decide whether the proof π' provided by algebraic $\mathcal{A}$ is a modification of the simulated proof π queried before in the simulation case, but we can pinpoint which exact simulated proof it was derived from.

Before we start the weak SE proof we present a re-phrased knowledge soundness proof, on top of which we will build the main theorem proof.

Theorem 3.2.2 ([Fuchsbauer et al., 2018])**.** *Groth16 achieves knowledge soundness against algebraic adversaries under the* $(2n-1, n-1)$***-dlog*** *assumption.*

Proof. We start by assuming a certain number of variables to be unknown to $\mathcal{A}$, in this particular case these are just the CRS trapdoors $\mathtt{td} = (\alpha, \beta, \gamma, \delta, x)$. We rely on Lemma 3.1.1. When $\mathcal{A}$ presents the proof $\pi = ([a]_1, [b]_2, [c]_1)$ that satisfies the verification equation, that is $V(\pi) = 0$, we conclude that $\mathcal{A}$ could not come up with π satisfying V unless for $V' = V(K \cdot \mathsf{Setup}_\lambda(\boldsymbol{T}))$ we have $V'(\boldsymbol{T}) = 0$ as a polynomial. Then we, step by step, analyze the coefficients K of the verification equation, by relying on the property that every monomial coefficient of the equation is zero (because the polynomial is constant zero). This is the most technical part of the proof, and we remind the reader that the other part that provides the reduction to $(2n-1, n-1)$**-dlog** is deferred generically to Lemma 3.1.1.

The matrix K contains a representation of A, B, and C as linear combination of public CRS elements (where C follows the same pattern as A):

$$A = A_1\alpha + A_2\beta + A_3\delta + \sum_{i=0}^{n-1} A_{4,i}x^i + \sum_{i=0}^{l} A_{5,i}\frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\gamma} +$$

$$\sum_{i=l+1}^{m} A_{6,i}\frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\delta} + \sum_{i=0}^{n-2} A_{7,i}\frac{x^i t(x)}{\delta}$$

$$B = B_1\beta + B_2\gamma + B_3\delta + \sum_{i=0}^{n-1} B_{4,i}x^i \qquad C = C_1\alpha + \ldots + \sum_{i=0}^{n-2} C_{7,i}\frac{x^i t(x)}{\delta}$$

We let $\boldsymbol{C} = (A_1, \ldots, A_{7,n-2}, \ldots, B_{4,n-1}, \ldots, C_{7,n-2})$ denote this set of variables serving as linear combination coefficients. In the following we will write CRS trapdoors as concrete values $(\alpha, \beta, \ldots, x)$, though they can be equally interpreted as formal variables $(X_\alpha, X_\beta, \ldots, X_x)$; we will avoid these former notation for convenience, since the main variables in scope that the system of equation is over are $\{A_i\}, \{B_i\}, \{C_i\}$, and we use trapdoor variables only to show how to form the system. This is, however, an important distinction: When we write $P(\alpha, x) = 0$, we imply $P(X_\alpha, X_x)$ is constant zero, and not just zero at (α, x).

For a polynomial $P(\boldsymbol{X})$ and a monomial $M = X_1^{b_1}X_2^{b_2}\cdots X_n^{b_n}$, $P_{[M]}$ will denote the coefficient of $P(\boldsymbol{X})$ at M, that is $P(\boldsymbol{X}) = \sum_M P_{[M]}M$. For each monomial M, we write out the corresponding monomial coefficient $V'_{[M]}$ as an equation $V'_{[M]} = 0$, and iteratively simplify the system of equations in $\boldsymbol{C}$. To simplify the proof, the 'monomials' we consider implicitly contain sums of powers of x [4], thus x^i will appear in coefficients. We start with examining the following equations, listed by monomials they are produced by, and by the terms of the verification equation they are extracted from:

$$\alpha\beta \text{ in } AB - \alpha\beta : A_1B_1 = 1 \implies A_1 \neq 0, B_1 \neq 0$$

$$\beta^2 \text{ in } AB : A_2B_1 = 0 \implies A_2 = 0$$

$$\alpha\gamma : A_1B_2 = 0 \implies B_2 = 0$$

$$\beta^2/\delta : \Big(\sum_{i=l+1}^{m} A_{6,i}u_i(x)\Big)B_1 = 0 \implies \sum_{i=l+1}^{m} A_{6,i}u_i(x) = 0$$

$$\beta\alpha/\delta : \Big(\sum_{i=l+1}^{m} A_{6,i}v_i(x)\Big)B_1 = 0 \implies \sum_{i=l+1}^{m} A_{6,i}v_i(x) = 0$$

$$\beta/\delta \text{ in } AB : \Big(\sum_{i=l+1}^{m} A_{6,i}w_i(x) + \sum_{i=0}^{n-2} A_{7,i}x^i t(x)\Big)B_1 +$$

[4] For monomial M instead of analysing $V'_{[M]} = 0$ we set $\tilde{V}'_{[M]} = \sum_i V_{[Mx^i]} = 0$. This is still a valid statement, since $V'(\boldsymbol{T}) = 0$ implies $V'_{[Mx^i]} = 0$ for each i, so each sum over x^i for M not containing any powers of x is also zero. It is always possible to split $\tilde{V}'_{[M]}$ further as $(\tilde{V}'_{[M]})_{[x^i]}$, extracting coefficients of x^i from it. We will do so implicitly in the "different spans of x powers" argument in the proof.

$$
\begin{aligned}
&\Big(\sum_{i=l+1}^{m} A_{6,i}u_i(x)\Big)\Big(\sum_{i=0}^{n-1} B_{4,i}x^i\Big) = 0 \wedge \\
1/\delta :\ &\Big(\sum_{i=l+1}^{m} A_{6,i}w_i(x) + \sum_{i=0}^{n-2} A_{7,i}x^i t(x)\Big)\Big(\sum_{i=0}^{n-1} B_{4,i}x^i\Big) = 0 \\
\implies &\sum_{i=0}^{n-2} A_{7,i}x^i t(x) = 0 \wedge \sum_{i=l+1}^{m} A_{6,i}w_i(x) = 0
\end{aligned}
$$

If $(\sum_{i=0}^{n-1} B_{4,i}x^i) = 0$ then from β/δ we have $\sum_{i=l+1}^{m} A_{6,i}w_i(x) + \sum_{i=0}^{n-2} A_{7,i}x^i t(x) = 0$, and otherwise from $1/\delta$ we have $\sum_{i=l+1}^{m} A_{6,i}w_i(x) + \sum_{i=0}^{n-2} A_{7,i}x^i t(x) = 0$. Now, since the sums have different spans of x^i powers, $\sum_{i=0}^{n-2} A_{7,i}x^i t(x) = 0$ and $\sum_{i=l+1}^{m} A_{6,i}w_i(x) = 0$.

$$
\begin{aligned}
\beta^2/\gamma \text{ in } AB :\ &\Big(\sum_{i=0}^{l} A_{5,i}u_i(x)\Big)B_1 = 0 \implies \sum_{i=0}^{l} A_{5,i}u_i(x) = 0 \\
\beta\alpha/\gamma :\ &\Big(\sum_{i=0}^{l} A_{5,i}v_i(x)\Big)B_1 = 0 \implies \sum_{i=0}^{l} A_{5,i}v_i(x) = 0 \\
\beta/\gamma :\ &\Big(\sum_{i=0}^{l} A_{5,i}w_i(x)\Big)B_1 + \Big(\sum_{i=0}^{l} A_{5,i}u_i(x)\Big)\Big(\sum_{i=0}^{n-1} B_{4,i}x^i\Big) = 0 \wedge \\
1/\gamma :\ &\Big(\sum_{i=0}^{l} A_{5,i}w_i(x)\Big)\Big(\sum_{i=0}^{n-1} B_{4,i}x^i\Big) = 0 \\
\implies &\sum_{i=0}^{l} A_{5,i}w_i(x) = 0 \ \text{ as with } \beta/\delta \wedge 1/\delta
\end{aligned}
$$

We now consider the following three monomials (β, α, and 1 that is only x powers) that we will call critical (and, respectively, the related equations too). Critical equations contain parts of the QAP, and we will eventually extract the witness from them. The underlined coefficients are already known to be zero, and thus the related sums are immediately cancelled:

$$
\begin{aligned}
&\beta \text{ in } AB - \varphi(\vec{\mathsf{x}})\gamma - C\delta : \\
&\Big(\sum_{i=0}^{n-1} A_{4,i}x^i\Big)B_1 + \Big(\sum_{i=0}^{n-1} B_{4,i}x^i\Big)\underline{A_2} + \underline{\Big(\sum_{i=0}^{l} A_{5,i}u_i(x)\Big)\underline{B_2}} + \underline{\Big(\sum_{i=l+1}^{m} A_{6,i}u_i(x)\Big)B_3} \\
&\qquad\qquad\qquad\qquad\qquad\qquad \sum_{i=0}^{l} a_i u_i(x) + \sum_{i=l+1}^{m} C_{6,i}u_i(x)
\end{aligned}
$$

α in $AB - \varphi(\vec{\mathsf{x}})\gamma - C\delta$:

$$\Big(\sum_{i=0}^{n-1} B_{4,i}x^i\Big)A_1 + \underline{\Big(\sum_{i=0}^{l} A_{5,i}v_i(x)\Big)\underline{B_2}} + \underline{\Big(\sum_{i=l+1}^{m} A_{6,i}v_i(x)\Big)}B_3 = \sum_{i=0}^{l} a_i v_i(x) + \sum_{i=l+1}^{m} C_{6,i}v_i(x)$$

1 (only x) in $AB - \varphi(\vec{\mathsf{x}})\gamma - C\delta$:

$$\Big(\sum_{i=0}^{n-1} A_{4,i}x^i\Big)\Big(\sum_{i=0}^{n-1} B_{4,i}x^i\Big) + \underline{\Big(\sum_{i=0}^{l} A_{5,i}w_i(x)\Big)\underline{B_2}} + \Big(\underline{\sum_{i=l+1}^{m} A_{6,i}w_i(x)} + \underline{\sum_{i=0}^{n-2} A_{7,i}x^i t(x)}\Big)B_3 = \sum_{i=0}^{l} \mathsf{a}_i w_i(x) + \sum_{i=l+1}^{m} C_{6,i}w_i(x) + \sum_{i=0}^{n-2} C_{7,i}x^i t(x)$$

Substituting the first two equations into the left hand side of the third one, given that $A_1B_1 = 1$:

$$\Big(\sum_{i=0}^{l} \mathsf{a}_i u_i(x) + \sum_{i=l+1}^{m} C_{6,i}u_i(x)\Big)\Big(\sum_{i=0}^{l} \mathsf{a}_i v_i(x) + \sum_{i=l+1}^{m} C_{6,i}v_i(x)\Big) = \sum_{i=0}^{l} \mathsf{a}_i w_i(x) + \sum_{i=l+1}^{m} C_{6,i}w_i(x) + \sum_{i=0}^{n-2} C_{7,i}x^i t(x)$$

Because a_0 is always 1 and A_1 and B_1 are nonzero, what we obtain is exactly a QAP statement with $h(x) = \sum_{i=0}^{n-2} C_{7,i}x^i$, hence $\{C_{6,i}\}_{i=l+1}^{m}$ is the assignment of the witness wires. The extractor can thus simply return these values. □

Finally, we give a proof for Theorem 3.2.1 which shows that Groth16 has white-box weak SE.

Proof of Theorem 3.2.1, Weak SE of Groth16. Let q denote the number of simulation queries of $\mathcal{A}$, and $\{\mathsf{a}_{i,j}\}_{j=0}^{l}$ denote the instance for the ith query. We now add the three proof elements $[\tilde{a}_i]_1, [\tilde{b}_i]_2, [\tilde{c}_i]_1$ revealed in each simulation to the list of elements that $\mathcal{A}$ can use as an algebraic extraction basis: $\tilde{a}_i = \mu_i, \tilde{b}_i = \nu_i$, and $\tilde{c}_i = (\mu_i\nu_i - \alpha\beta - \sum_{j=0}^{l} \mathsf{a}_{i,j}(\beta u_j(x) + \alpha v_j(x) + w_j(x)))/\delta$. We write out the representation of A and B (C follows the same pattern as A) from the verification equation as the linear combination of the public CRS and new simulated proof elements:

$$A = \cdots + \sum_{i=1}^{q} A_{8,i}\mu_i + \sum_{i=1}^{q} A_{9,i}\frac{\mu_i\nu_i - \alpha\beta - \sum_{j=0}^{l} \mathsf{a}_{i,j}(\beta u_j(x) + \alpha v_j(x) + w_j(x))}{\delta}$$

$$B = \cdots + \sum_{i=1}^{q} B_{5,i}\nu_i$$

Our goal is to reduce the theorem to the knowledge-soundness case by restricting the coefficients related to the new simulated proofs variables, namely $A_{8,i}$, $A_{9,i}$, $B_{5,i}$, $C_{8,i}$, $C_{9,i}$. We will show that a successful $\mathcal{A}$ must either reuse one of the simulated proofs (potentially randomizing it), or it must not have used any simulation-related variables, thus allowing for the reuse of the extraction argument from knowledge soundness. We start by inspecting coefficients of the following monomials (affected by simulated proofs):

$$\alpha\beta \text{ in } AB - C\delta : A_1B_1 - \sum_{i=1}^{q} A_{9,i}B_3 + \sum_{i=1}^{q} C_{9,i} = 1 \qquad \mu_i\beta \text{ in } AB : A_{8,i}B_1 = 0$$

$$\mu_i\nu_j (i \neq j) \text{ in } AB : A_{8,i}B_{5,j} = 0 \qquad \mu_i\gamma \text{ in } AB : A_{8,i}B_2 = 0$$

$$\mu_i\nu_i \text{ in } AB - C\delta : A_{9,i}B_3 + A_{8,i}B_{5,i} - C_{9,i} = 0 \qquad \mu_i\delta \text{ in } AB - C\delta : A_{8,i}B_3 - C_{8,i} = 0$$

$$\mu_i\nu_i\nu_j/\delta \text{ in } AB : A_{9,i}B_{5,j} = 0 \qquad \nu_i\alpha \text{ in } AB : B_{5,i}A_1 = 0$$

$$\mu_i\nu_i\beta/\delta \text{ in } AB : A_{9,i}B_1 = 0 \qquad \nu_i\beta \text{ in } AB : B_{5,i}A_2 = 0$$

$$\nu_i\delta \text{ in } AB : B_{5,i}A_3 = 0$$

First, we show that all $A_{9,i} = 0$. Assume the contrary: $A_{9,k} \neq 0$ for some k. Then from Equation ($\mu_k\nu_k\nu_j/\delta$) for all j: $B_{5,j} = 0$. From Equation ($\mu_i\nu_i$) for all i we have that $C_{9,i} = A_{9,i}B_3$, which, substituted into Equation ($\alpha\beta$) give us $A_1B_1 = 1$. Hence $B_1 \neq 0$, but from Equation ($\mu_k\nu_k\beta/\delta$) we see that $A_{9,k}B_1 = 0$, but neither $A_{9,k}$ nor B_1 is zero, a contradiction. Thus, all $A_{9,i} = 0$, and furthermore Equation ($\alpha\beta$) simplifies to $A_1B_1 + \sum_{i=1}^{q} C_{9,i} = 1$ and Equation ($\mu_i\nu_i$) simplifies to $A_{8,i}B_{5,i} = C_{9,i}$.

We now show, that if at least one $A_{8,k} \neq 0$, then $\mathcal{A}$ reuses the kth simulated proof, and otherwise if all $A_{8,i} = 0$ it does not use any simulation-related elements.

- Assume, first, that all $A_{8,i} = 0$: From Equation ($\mu_i\nu_i$) all $C_{9,i} = 0$. Then, $A_1B_1 = 1$ by Equation ($\alpha\beta$), so from Equation ($\nu_i\alpha$) all $B_{5,i} = 0$ (since $A_1 \neq 0$), and from Equation ($\mu_i\delta$) all $C_{8,i} = 0$ because all $A_{8,i} = 0$. We now have cancelled all the simulation-related variables, and thus $\mathcal{A}$ does not use simulation queries when constructing its proof, and we can reduce the proof to the knowledge soundness case.
- Assume, otherwise, that at least one $A_{8,k} \neq 0$: Then $B_1 = B_2 = 0$ from Equation ($\mu_k\beta$) and Equation ($\mu_k\gamma$). For all $j \neq k$ from Equation ($\mu_k\nu_j$) we have $B_{5,j} = 0$, and since $C_{9,j} = B_{5,j}A_{8,j}$, all $C_{9,j} = 0$ for $j \neq k$ too. From Equation ($\alpha\beta$) we obtain $C_{9,k} = 1$, thus $B_{5,k} = 1/A_{8,k}$ by Equation ($\mu_i\nu_i$). Since now $B_{5,k} \neq 0$, from the Equations ($\nu_k\alpha$), ($\nu_k\beta$), ($\nu_k\delta$) we

have $A_1 = A_2 = A_3 = 0$. Thus, we are only left with exactly one nonzero triple $(A_{8,k}, B_{5,k}, C_{9,k})$, which means $\mathcal{A}$ has used at most one simulated proof number k, not being able to combine several simulated proofs into one.

We next look at additional coefficients related to monomials that include ν_k and μ_k. From Equations ($\nu_i\beta/\delta$), ($\nu_i\alpha/\delta$), (ν_i/δ) we have $\sum_{i=l+1}^{m} A_{6,i}(\beta u_i(x) + \alpha v_i(x) + w_i(x))/\delta + \sum_{i=0}^{n-2} A_{7,i}x^i t(x)/\delta = 0$ (related terms of A are the only terms matching this ν_i in B):

$$
\begin{aligned}
\nu_k\beta/\delta \text{ in } AB : &\Big(\sum_{j=l+1}^{m} A_{6,j}u_j(x) - \sum_{i=1}^{q} \underline{A_{9,i}} \sum_{j=0}^{l} u_j(x)\Big)B_{5,k} = 0 \\
\implies &\sum_{j=l+1}^{m} A_{6,j}u_j(x) = 0 \\
\nu_k\alpha/\delta \text{ in } AB : &\Big(\sum_{j=l+1}^{m} A_{6,j}v_j(x) - \sum_{i=1}^{q} \underline{A_{9,i}} \sum_{j=0}^{l} v_j(x)\Big)B_{5,k} = 0 \\
\implies &\sum_{j=l+1}^{m} A_{6,j}v_j(x) = 0 \\
\nu_k/\delta \text{ in } AB : &\Big(\sum_{j=l+1}^{m} A_{6,j}w_j(x) + \sum_{i=0}^{n-2} A_{7,i}x^i t(x) - \sum_{i=1}^{q} \underline{A_{9,i}} \sum_{j=0}^{l} w_j(x)\Big)B_{5,k} = 0 \\
\implies &\sum_{j=l+1}^{m} A_{6,j}w_j(x) = 0 \wedge \sum_{i=0}^{n-2} A_{7,i}x^i t(x) = 0 \text{ (different powers of } x)
\end{aligned}
$$

Similarly, from Equations ($\nu_i\beta/\gamma$), ($\nu_i\alpha/\gamma$), (ν_i/γ) we have $\sum_{i=0}^{l} A_{5,i}(\beta u_i(x)/\gamma) = \sum_{i=0}^{l} A_{5,i}(\alpha v_i(x)/\gamma) = \sum_{i=0}^{l} A_{5,i}(w_i(x)/\gamma) = 0$ (the coefficients are also extracted from AB).

$$
\begin{aligned}
\nu_k\beta/\gamma \text{ in } AB : &\Big(\sum_{j=0}^{l} A_{5,j}u_j(x)\Big)B_{5,k} = 0 \implies \sum_{j=0}^{l} A_{5,j}u_j(x) = 0 \\
\nu_k\alpha/\gamma \text{ in } AB : &\Big(\sum_{j=0}^{l} A_{5,j}v_j(x)\Big)B_{5,k} = 0 \implies \sum_{j=0}^{l} A_{5,j}v_j(x) = 0 \\
\nu_k/\gamma \text{ in } AB : &\Big(\sum_{j=0}^{m} A_{5,j}w_j(x)\Big)B_{5,k} = 0 \implies \sum_{j=l+1}^{m} A_{5,j}w_j(x) = 0
\end{aligned}
$$

Because of Equation (ν_k) and Equation (μ_k) we have $\sum_{i=0}^{n-1} A_{4,i}x^i = 0$ and $\sum_{i=0}^{n-1} B_{4,i}x^i = 0$ related terms cancelled as well:

$$
\nu_k \text{ in } AB : \Big(\sum_{i=0}^{n-1} A_{4,i}x^i\Big)B_{5,k} = 0 \implies \sum_{i=0}^{n-1} A_{4,i}x^i = 0
$$

$$\mu_k \text{ in } AB : \Big(\sum_{i=0}^{n-1} B_{4,i}x^i\Big)A_{8,k} = 0 \implies \sum_{i=0}^{n-1} B_{4,i}x^i = 0$$

Which also implies $A_{4,i} = B_{4,i} = 0$ for all i. We now focus on the first critical equation, β, which has the same elements as in the KS case, except for the additional $C_{9,i}$. Its left side vanishes completely, and on the right we have exactly one additional simulated instance wires set corresponding to $C_{9,k} = 1$:

$$0 = \sum_{i=0}^{l} \mathsf{a}_i u_i(x) + \sum_{i=l+1}^{m} C_{6,i}u_i(x) - \sum_{i=0}^{l} \mathsf{a}_{k,i}u_i(x)$$

Because of disjointness[5] between $u_i(x)$ for witness and instance sets of indices we have both $\sum_{i=0}^{l}(\mathsf{a}_i - \mathsf{a}_{k,i})u_i(x) = 0$ and $\sum_{i=l+1}^{m} C_{6,i}u_i(x) = 0$, thus also $\mathsf{a}_i = \mathsf{a}_{k,i}$ because of the linear independence of the first set. Then $\mathcal{A}$ has reused the simulated instance $\mathbf{x}_k = \{\mathsf{a}_{k,i}\}_{i=0}^{l}$, which concludes the proof.

□

3.3 Malleability of Groth16

Practically, it is known that Groth16 has malleable proofs. It is not hard to extend this statement to show that Groth16 is rerandomizable, that is its output of Rand is indistinguishable from honest proofs, even if Rand is applied to maliciously generated (but verifiable) proofs.

Theorem 3.3.1. *Groth16 zk-SNARK is rerandomizable[6] with respect to the randomization transformation* Rand *presented in Fig. 2.3.*

Proof. Deferred to the end of this section. In a nutshell, the proof elements a and b output by Rand are random and independent of each other; and the verification equation fixes a unique c based on $a, b, \boldsymbol{\sigma}, \mathbf{x}$. □

Together with white-box weak SE forbidding instance malleability, and perfect ZK, Theorem 3.3.1 implies that randomization is equivalent to any other way to transform the honest (or simulated) proofs. But this does not give an explicit algebraic characterization of the transformation — we do not know if there is any other way to

[5] This technique was applied in a similar manner for strong SE in [Groth and Maller, 2017]

[6] This property has been observed before, for example in [Lee et al., 2019] in a similar context.

Observation 1.

$$a = A_1\alpha + A_3\delta + A_1\sum_{i=0}^{m} a_i u_i(x) \qquad b = \frac{1}{A_1}\beta + B_3\delta + \frac{1}{A_1}\sum_{i=0}^{m} \mathsf{a}_i v_i(x)$$

$$c = B_3 A + A_3 B - A_3 B_3 \delta + \sum_{i=l+1}^{m} \mathsf{a}_i \frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\delta} + \sum_{i=0}^{n-2} h_i \frac{x^i t(x)}{\delta}$$

Figure 3.1: The kernel of Groth16 verification equation (a subspace of $Z_p^{9+5n+2m}$) structured as a proof generation routine (the most general one). Note the additional random value A_1, that is set to 1 in Prove of Fig. 2.3, but is affected by randomization.

create an honest proof, or any other way to rerandomize it (that would produce the same distribution). One of the interesting properties of the proof of Theorem 3.2.1 is that it can be extended to show that Rand is the only algebraic transformation possible, which we present as an independent result. We also show that the most-general algebraic form of the honest generation procedure has at most three random “axes”, any two of which are required for perfect zero-knowledge; Rand, parametrised by just two random values, changes all three of them. Details are provided in Appendix 3.3.

Let $V(\boldsymbol{C}) = 0$ with $\boldsymbol{C} = (A_1, \ldots, A_{7,n-2}, B_1, \ldots, B_{4,n-1}, C_1, \ldots, C_{7,n-2})$ be the verification equation of Groth16 expressed in terms of exponent of $\mathbb{G}_T$ with the $9 + 5n + 2m$ variables serving as linear coefficients that construct the proof from CRS elements, then the kernel[7] of $V(\boldsymbol{C})$ is as presented in Fig. 3.1.

Observation 2. *The only form of algebraic transformation on Groth16 proofs that is possible without violating its verification equation is the randomization procedure* Rand$(\boldsymbol{\sigma}, \pi = (a, b, c); r_1, r_2)$*, where r_1, r_2 are chosen by the adversary.*

Proof of Theorem 3.3.1. In order to prove the statement, we need to show that the distribution of honestly generated proofs $\{\pi\}_\lambda = \{(a, b, c)\}_\lambda$ is the same as the distribution of re-randomized proofs $\{\mathsf{Rand}(\pi)\}_\lambda = \{(a', b', c')\}_\lambda$, where π is perhaps not honestly generated, but necessarily verifies. In honestly generated proofs, first two values a, b are independently uniform, and the third element of the tuple is defined from them.

[7]That is, $X \subset \mathbb{Z}_p^{|\boldsymbol{C}|}$ such that $\forall \boldsymbol{c} \in X. V(\boldsymbol{c}) = 0$

By examining Rand, we immediately see that r_1 makes $a' = r_1 a$ uniform, and that $b' = r_1 b + r_1 r_2 [\delta]_2$ is also independent since $r_1 r_2 \delta$ is uniform because of r_2. Thus we obtain two uniform distributions, and this is true irrespectively of the original distribution of π. Since Rand is correct, the modified proof also verifies. Hence in both distributions the first two tuple elements are uniform, and the third depends on them in the same way, defined by Groth16 verification equation. □

Proof of Observation 1. We start by taking the KS version of the proof elements parametrisation (A, B, C expressed as a linear combination of CRS elements with coefficients containing A_i, B_i and C_i), and applying the constraints we obtained in the KS proof. The malleability constraints we will show are the same for both simulated and real proofs because of indistinguishability of simulated proofs. We apply the reductions from the KS proof, and immediately cancel A_2, B_2, $A_{6,i}$ and $A_{5,i}$ related sums, and the sum with $A_{7,i}$. We also substitute a_i instead of $C_{6,i}$ and $h(x)$ instead of $C_{7,i}$. Since $A_1 B_1 = 1$, we set $B_1 = 1/A_1$.

$$A = A_1\alpha + A_3\delta + \sum_{i=0}^{n-1} A_{4,i}x^i \qquad B = \frac{1}{A_1}\beta + B_3\delta + \sum_{i=0}^{n-1} B_{4,i}x^i$$

$$C = C_1\alpha + C_2\beta + C_3\delta + \sum_{i=0}^{n-1} C_{4,i}x^i + \sum_{i=0}^{l} C_{5,i}\frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\gamma} +$$
$$+ \sum_{i=l+1}^{m} \mathsf{a}_i \frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\delta} + \sum_{i=0}^{n-2} h_i \frac{x^i t(x)}{\delta}$$

In order to restrain $C_{5,i}$ we need to investigate another set of coefficients:

$$\beta\delta/\gamma : \Big(\sum_{i=0}^{l} A_{5,i}u_i(x)\Big)B_3 + \sum_{i=0}^{l} C_{5,i}u_i(x) = 0$$
$$\alpha\delta/\gamma : \Big(\sum_{i=0}^{l} A_{5,i}v_i(x)\Big)B_3 + \sum_{i=0}^{l} C_{5,i}v_i(x) = 0$$
$$\delta/\gamma : \Big(\sum_{i=0}^{l} A_{5,i}w_i(x)\Big)B_3 + \sum_{i=0}^{l} C_{5,i}w_i(x) = 0$$

And as sums with $A_{5,i}$ are zero, we conclude that the relevant sums with $C_{5,i}$ are also zero, so we can exclude them from C. We once again investigate critical equations for α and β:

$$\beta : \Big(\sum_{i=0}^{n-1} A_{4,i}x^i\Big) = A_1\Big(\sum_{i=0}^{l} \mathsf{a}_i u_i(x) + \sum_{i=l+1}^{m} C_{6,i}u_i(x)\Big)$$
$$\alpha : \Big(\sum_{i=0}^{n-1} B_{4,i}x^i\Big) = \frac{1}{A_1}(\sum_{i=0}^{l} \mathsf{a}_i v_i(x) + \sum_{i=l+1}^{m} C_{6,i}v_i(x)\Big)$$

We substitute $A_{4,i}$ and $B_{4,i}$ sums into the general form of an honest proof, given that $C_{6,i} = \mathsf{a}_i$. What we get is:

$$A = A_1\alpha + A_3\delta + A_1\sum_{i=0}^{m} \mathsf{a}_i u_i(x) \qquad B = \frac{1}{A_1}\beta + B_3\delta + \frac{1}{A_1}\sum_{i=0}^{m} \mathsf{a}_i v_i(x)$$
$$C = C_1\alpha + C_2\beta + C_3\delta + \sum_{i=0}^{n-1} C_{4,i}x^i + \sum_{i=l+1}^{m} \mathsf{a}_i \frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\delta} + \sum_{i=0}^{n-2} h_i \frac{x^i t(x)}{\delta}$$

We now restrain A_3, B_3 ($A_2 = 0$):

$$\delta^2 : A_3B_3 = C_3$$
$$\beta\delta : A_3B_1 + \underline{A_2}B_3 = C_2$$
$$\alpha\delta : A_1B_3 = C_1$$

And express $C_{4,i}$ related sum using $A_{4,i}$ and $B_{4,i}$:

$$\delta : \Big(\sum_{i=0}^{n-1} B_{4,i}x^i\Big)A_3 + \Big(\sum_{i=0}^{n-1} A_{4,i}x^i\Big)B_3 = \sum_{i=0}^{n-1} C_{4,i}x^i$$

The fully reduced system that we obtain now has three free variables (A_1, A_3, B_3), and has the following form:

$$A = A_1\alpha + A_3\delta + A_1\sum_{i=0}^{m} \mathsf{a}_i u_i(x) \qquad B = \frac{1}{A_1}\beta + B_3\delta + \frac{1}{A_1}\sum_{i=0}^{m} \mathsf{a}_i v_i(x)$$
$$C = A_1B_3\alpha + \frac{A_3}{A_1}\beta + A_3B_3\delta + B_3A_1\sum_{i=0}^{m} \mathsf{a}_i u_i(x) + \frac{A_3}{A_1}\sum_{i=0}^{m} \mathsf{a}_i v_i(x) + \sum_{i=l+1}^{m} \mathsf{a}_i \frac{q_i(x)}{\delta} +$$
$$\sum_{i=0}^{n-2} h_i \frac{x^i t(x)}{\delta} = B_3A + A_3B - A_3B_3\delta + \sum_{i=l+1}^{m} \mathsf{a}_i \frac{q_i(x)}{\delta} + \sum_{i=0}^{n-2} h_i \frac{x^i t(x)}{\delta}$$

Since this general form of proof generation satisfies the verification equation (this is easy to verify), no further reductions are possible. Indeed, two out of three free variables are used in the honest generation procedure, and the third one is modified in the randomization transformation. □

Proof of Observation 2. Now, in order to obtain the explicit form randomization transformation, we would need to trasform each proof element so that they still fit the bounds we have just presented. Although, this is easier to show if we repeat the process over again, but with the weak SE proof, now assuming that $\mathcal{A}$ uses one simulated query (weak SE has shown that no combination of two proofs can be a valid proof). This makes things simpler, because simulated variables μ_i and ν_i stand exactly for already-composed proof elements a and b.

Assume that $A_{8,k} \neq 0$. In the SE proof we already show almost all the coefficient reductions (all A_i except for $A_{8,k}$, all B_i except for B_3 and $B_{5,k}$, $C_{7,i}$, $C_{9,i}$ for $i \neq k$, and $C_{9,k} = 1$). This gives us the following set of equations:

$$\begin{aligned} A &= A_{8,k}\mu_k \qquad\qquad B = \ B_3\delta + B_{5,k}\nu_k \\ C &= C_1\alpha + C_2\beta + C_3\delta + \sum_{i=0}^{n-1} C_{4,i}x^i + \sum_{i=0}^{l} C_{5,i}y_i(x) \\ &\quad + \sum_{i=l+1}^{m} C_{6,i}\frac{q_i(x)}{\delta} + \sum_{i=1}^{q} C_{8,i}\mu_i + \frac{\mu_k\nu_k - \alpha\beta - \sum_{j=0}^{l} \mathsf{a}_{k,j}q_j(x)}{\delta} \end{aligned}$$

Further reductions are also easy to discover. From Equation ($\mu_i\delta$), $B_3 = C_{8,k}/A_{8,k}$, and all other $C_{8,i} = 0$. From Equation (δ^2), $C_3 = A_3B_3 = 0$. From Equation ($\alpha\delta$), $C_1 = A_1B_3 = 0$. From Equation ($\beta\delta$), $C_2 = A_3B_1 + A_2B_3 = 0$. We also substitute already obtained $B_{5,k} = 1/A_{8,k}$ from the SE proof:

$$\begin{aligned} A &= A_{8,k}\mu_k \qquad B = \frac{1}{A_{8,k}}\nu_k + \frac{C_{8,k}}{A_{8,k}}\delta \\ C &= \sum_{i=0}^{n-1} C_{4,i}x^i + \sum_{i=0}^{l} C_{5,i}y_i(x) + \sum_{i=l+1}^{m} C_{6,i}\frac{q_i(x)}{\delta} + C_{8,k}\mu_k + \frac{\mu_k\nu_k - \alpha\beta - \sum_{j=0}^{l} \mathsf{a}_{k,j}q_j(x)}{\delta} \end{aligned}$$

We now need to remove the $C_{4,i}, C_{5,i}, C_{6,i}$ related sums. Nothing can compensate $\sum_{i=0}^{n-1} C_{4,i}x^i$ if we take a look at δ, so it cancels out. Same for $C_{5,i}$ related sum, and monomials $\beta\delta/\gamma$, $\alpha\beta/\gamma$, δ/γ. $C_{6,i}$ also can not be compensated, because of span disjointness of $u_i(X)$ for instance and witness wires, and since the verification equation only includes the instance-related sum (formally, we view the monomial β equation; the end of Theorem 3.2.1 proof explains the technique). What we left with is precisely the well-known randomization Rand, where $r_1 = 1/A_{8,k}$, and

$r_2 = C_{8,k}$:

$$A = A_{8,k}\mu_k \qquad B = \frac{1}{A_{8,k}}\nu_k + \frac{C_{8,k}}{A_{8,k}}\delta$$

$$C = C_{8,k}\mu_k + \frac{\mu_k\nu_k - \alpha\beta - \sum_{j=0}^{l} \mathsf{a}_{k,j}(\beta u_j(x) + \alpha v_j(x) + w_j(x))}{\delta}$$

$\square$

3.4 Black-box Weak SE of Groth16

We study two approaches to achieve black-box weak SE by encrypting the witness. The first construction Int-Groth16 integrates ciphertexts directly to the relation, and the second construction Ext-Groth16 proves the correctness of ciphertexts with external techniques.

3.4.1 BB Weak SE with Internal Encryption

First, we describe a generic transformation for achieving black-box weak SE. We let the prover encrypt the witness w with a IND-CPA secure PKE and then use a weak simulation sound NIZK (e.g., Groth16) to prove the relation

$$\mathcal{R}' := \{((\mathsf{x}, \mathsf{pk}, \boldsymbol{c}), (\mathsf{w}, r)) : (\mathsf{x}, \mathsf{w}) \in \mathcal{R} \wedge \boldsymbol{c} = \mathsf{Enc}(\mathsf{pk}, \mathsf{w}; r)\},$$

where x is the statement the prover wants to prove and $\mathcal{R}$ is the corresponding relation. Since we make the public key pk a part of the reference string, it will be possible to black-box extract the witness from the ciphertext. Full details of the construction can be seen in Fig. 3.2.

This transformation was first analyzed in [Baghery, 2019], where it was shown to lift a white-box *strong* SE NIZK to a black-box *strong* SE. Below we sketch a proof that it also lifts a weak simulation sound NIZK to a black-box weak SE NIZK.

Theorem 3.4.1. *Let* $\mathsf{NIZK}' = (\mathsf{Setup}', \mathsf{Prove}', \mathsf{Verify}', \mathsf{Sim}')$ *be a complete, weak simulation sound, and computational zero-knowledge non-interactive proof system and* $\mathsf{PKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *an IND-CPA secure encryption scheme. Then the* NIZK *construction in Fig. 3.2 is complete, black-box weak SE, and computational zero-knowledge.*

$\underline{\mathsf{Setup}(\mathcal{R}_\lambda):}$

$(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$

$(\boldsymbol{\sigma}', \mathsf{td}') \xleftarrow{\$} \mathsf{Setup}'(\mathcal{R}')$

$\textbf{return } (\boldsymbol{\sigma} := \boldsymbol{\sigma}' \cup \mathsf{pk},$

$\qquad \mathsf{td} := \mathsf{td}', \mathsf{td}_{\mathsf{ext}} := \mathsf{sk})$

$\underline{\mathsf{Prove}(\boldsymbol{\sigma} = \boldsymbol{\sigma}' \cup \mathsf{pk}, \mathsf{x}, \mathsf{w}):}$

$r \xleftarrow{\$} \mathbb{Z}_p^*, \boldsymbol{c} \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathsf{w}; r)$

$\pi' \leftarrow \mathsf{Prove}'(\boldsymbol{\sigma}, (\mathsf{x}, \mathsf{pk}, \boldsymbol{c}), (\mathsf{w}, r))$

$\textbf{return } (\boldsymbol{c}, \pi')$

$\underline{\mathsf{Verify}(\boldsymbol{\sigma} = \boldsymbol{\sigma}' \cup \mathsf{pk}, \mathsf{x}, \pi = (\boldsymbol{c}, \pi')):}$

$\textbf{assert } \mathsf{Verify}'(\boldsymbol{\sigma}, (\mathsf{x}, \mathsf{pk}, \boldsymbol{c}), \pi')$

$\underline{\mathsf{Sim}(\boldsymbol{\sigma} = \boldsymbol{\sigma}' \cup \mathsf{pk}, \mathsf{td}, \mathsf{x}):}$

$\boldsymbol{c} \leftarrow \mathsf{Enc}(\mathsf{pk}, 0; r) \text{ for } r \xleftarrow{\$} \mathbb{Z}_p^*$

$\pi' \xleftarrow{\$} \mathsf{Sim}'(\boldsymbol{\sigma}', \mathsf{td}, (\mathsf{x}, \mathsf{pk}, \boldsymbol{c}))$

$\textbf{return } (\boldsymbol{c}, \mathsf{x});$

$\underline{\mathsf{Ext}(\boldsymbol{\sigma}, \mathsf{td}_{\mathsf{ext}}, \mathsf{x}, \pi = (\boldsymbol{c}, \pi')):}$

$\textbf{return } \mathsf{Dec}(\mathsf{td}_{\mathsf{ext}}, \boldsymbol{c})$

Figure 3.2: The construction for black-box weak SE NIZK where $\mathsf{NIZK}' = (\mathsf{Setup}', \mathsf{Prove}', \mathsf{Verify}', \mathsf{Sim}')$ is weakly simulation sound, and $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ is an IND-CPA secure PKE.

Proof (sketch). Completeness of NIZK follows from the completeness of NIZK$'$ and correctness of the cryptosystem. Computational zero-knowledge holds since $\mathsf{Enc}(\mathsf{pk}, 0)$ is computationally indistinguishable from $\mathsf{Enc}(\mathsf{pk}, \mathsf{w})$ and since NIZK$'$ already has computational zero-knowledge. Finally, suppose that there exists a PPT adversary $\mathcal{A}$ that can break black-box weak SE of NIZK. We can easily construct a PPT adversary $\mathcal{B}$ that can break weak simulation soundness of NIZK$'$. $\mathcal{B}$ gets $\boldsymbol{\sigma}'$ as an input and generates pk itself. Now $\mathcal{B}$ can run $\mathcal{A}(\boldsymbol{\sigma}' \cup \mathsf{pk})$ internally and whenever $\mathcal{A}$ makes a simulation query x, $\mathcal{B}$ makes a simulation query $(\mathsf{x}, \mathsf{pk}, \boldsymbol{c} = \mathsf{Enc}(\mathsf{pk}, 0))$ and gets back a proof π' which allows him to send $(\boldsymbol{c}, \pi')$ to $\mathcal{A}$. Finally, $\mathcal{A}$ outputs $(\mathsf{x}^*, (\boldsymbol{c}^*, \pi^*))$ such that x^* has not been queried and either x^* is an invalid statement or $\boldsymbol{c}$ does not encrypt the correct witness. Now $\mathcal{B}$ can output $((\mathsf{x}^*, \boldsymbol{c}^*), \pi^*)$ which will break weak simulation soundness. □

We can obtain good efficiency if we instantiate the above construction by taking Groth16 as NIZK$'$ and by using vector ElGamal (see Section 2.4 for details) as a cryptosystem. We call this instantiation Int-Groth16. In Section 3.5 we discuss further optimization of this construction.

Corollary 3.4.1.1. *Int-Groth16 is a complete, black-box weak SE, and computational zero-knowledge NIZK argument.*

3.4.2 BB Weak SE with External Encryption

The disadvantage of the previous construction is that one needs to encode the extended relation as an arithmetic circuit, that is shown, e.g. in Hawk, to result in a considerably larger public parameters and a slower prover. Thus, we propose a second construction Ext-Groth16 which is closely based on the SAVER cryptosystem [Lee et al., 2019] which in a sense gives ciphertexts as a public input to Groth16. Having the encryption outside of the circuit allows us to have smaller circuit overhead which results in smaller CRS size and higher prover efficiency. As before, proof size is linear, and is dominated by the size of the encrypted witness (this is inevitable for black-box constructions, as discussed before [Gentry and Wichs, 2011]). The formal description is presented on Fig. 3.3. Roughly speaking, we reinstantiate SAVER, but also prove that the construction is black-box weak simulation extractable. Additionally we re-prove computational zero-knowledge under the weaker and more standard DDH assumption.

3.4.2.1 Technical Details

As Ext-Groth16 is based on SAVER, we point out the important ways it is different from Groth16. First, we extend the CRS with the pk elements, similarly to how it is done in Int-Groth16 (since pk uses Groth16 trapdoors, it changes the security proof). Second, Groth16 itself is modified: while constructing the proof, element c has an additional coefficient, that is needed to balance out ciphertext randomness.

Crucially, Ext-Groth16 cannot achieve black-box strong SE, because it is proof malleable (and rerandomizable). First, the rerandomization of embedded Groth16 still works, because it does not interfere with the "ciphertext randomness cancelling term" of c. Second, ciphertexts are also rerandomizable: we can replace r with $r + r'$ additively in all c_i, in ψ and c (as shown in Fig. 3.3).

Another important distinction is that in order for the decryption to work efficiently (since it relies on solving discrete logarithm), plaintexts should be small enough. This is critical to guarantee the extraction — to prevent $\mathcal{A}$ from creating un-extractable proofs, we require the circuit itself to make range-checks on plaintext values. We account for the circuit growth in our efficiency evaluation, but in this section we assume the circuit transformation to be an implicit part of the construction, since this suffices for our security analysis.

Setup($\mathcal{R}$):

1. $\mathsf{td}_1 = x, \alpha, \beta, \gamma, \delta \xleftarrow{\$} \mathbb{Z}_p^*$; $\mathsf{td}_2 = \{s_i\}_{i=1}^{l_\mathsf{w}}, \{t_i\}_{i=0}^{l_\mathsf{w}} \xleftarrow{\$} \mathbb{Z}_p^*$
2. $\boldsymbol{\sigma}_1 \leftarrow \left[\alpha, \beta, \delta, \{x^i\}_{i=0}^{n-1}, \left\{\frac{x^i t(x)}{\delta}\right\}_{i=0}^{n-2}, \{y_i(x)\}_{i=0}^{l+l_\mathsf{w}}, \left\{\frac{q_i(x)}{\delta}\right\}_{i=l+l_\mathsf{w}+1}^{m}\right]_1$
3. $\boldsymbol{\sigma}_2 \leftarrow \left[\beta, \gamma, \delta, \{x^i\}_{i=0}^{n-1}\right]_2$
4. $\mathsf{pk}_1 \leftarrow \left[\{\delta s_i\}_{i=1}^{l_\mathsf{w}}, \{y_{l+i}(x) t_i\}_{i=1}^{l_\mathsf{w}}, \delta(t_0 + \sum_{i=1}^{l_\mathsf{w}} t_i s_i), \gamma(1 + \sum_{i=1}^{l_\mathsf{w}} s_i)\right]_1$
5. $\mathsf{pk}_2 \leftarrow \left[\{t_i\}_{i=0}^{l_\mathsf{w}}\right]_2$
6. **return** $(\boldsymbol{\sigma} = \boldsymbol{\sigma}_1 \cup \boldsymbol{\sigma}_2 \cup \mathsf{pk}_1 \cup \mathsf{pk}_2, \mathsf{td} = \mathsf{td}_1 \cup \mathsf{td}_2, \mathsf{td}_\mathsf{ext} = \{s_i\}_{i=1}^{l_\mathsf{w}})$

Prove($\boldsymbol{\sigma}, \mathbf{x} = \mathsf{x}_1 \dots \mathsf{x}_l, \mathbf{w} = \mathsf{w}_1 \dots \mathsf{w}_{l_\mathsf{w}} \dots \mathsf{w}_{m-l}$):

1. $r, r_a, r_b \xleftarrow{\$} \mathbb{Z}_p^*$
2. $c_0 \leftarrow r[\delta]_1;\ c_i \leftarrow r[\delta s_i]_1 + \mathsf{w}_i[y_{l+i}(x)]_1$ for $i \in [1 \dots l_\mathsf{w}]$
3. $\psi \leftarrow r\left[\delta(t_0 + \sum_{j=1}^{l_\mathsf{w}} t_j s_j)\right]_1 + \sum_{i=1}^{l_\mathsf{w}} \mathsf{w}_i [y_{l+i}(x) t_i]_1$
4. $[a]_1 \leftarrow \left[\alpha + \sum_{i=0}^{m} a_i u_i(x) + r_a \delta\right]_1$
5. $[b]_2 \leftarrow \left[\beta + \sum_{i=0}^{m} a_i v_i(x) + r_b \delta\right]_2$
6. $[c]_1 \leftarrow \left[\sum_{i=l+l_\mathsf{w}+1}^{m} \mathsf{w}_{i-l} \frac{q_i(x)}{\delta} + \frac{h(x)t(x)}{\delta} + a r_b + b r_a - r_a r_b \delta\right]_1 - r\left[\gamma\left(1 + \sum_{i=1}^{l_\mathsf{w}} s_i\right)\right]_1$
7. **return** $(([a]_1, [b]_2, [c]_1), \mathcal{CT} = (c_0, \dots, c_{l_\mathsf{w}}, \psi))$

Verify($\boldsymbol{\sigma}, \mathbf{x} = \mathsf{x}_1 \dots \mathsf{x}_l, \pi = ((a, b, c), (c_0, \dots, c_{l_\mathsf{w}}, \psi))$):

1. **assert** $\sum_{i=0}^{l_\mathsf{w}} \hat{e}(c_i, [t_i]_2) = \hat{e}(\psi, H)$
2. **assert** $\hat{e}(a, b) = \hat{e}([\alpha]_1, [\beta]_2) + \hat{e}(\sum_{i=0}^{l} \mathsf{x}_i [y_i(x)]_1 + \sum_{i=0}^{l_\mathsf{w}} c_i, [\gamma]_2) + \hat{e}(c, [\delta]_2)$

Rand($\boldsymbol{\sigma}, \pi = ((a, b, c), (c_0, c_1, \dots, c_{l_\mathsf{w}}, \mathsf{x}))$):

1. $r_1, r_2, r' \xleftarrow{\$} \mathbb{Z}_p^*$
2. $c_0 \mapsto c_0 + r'[\delta]_1;\ c_i \mapsto c_i + r'[\delta s_i];\ \psi \mapsto \psi + r'[\delta t_0 + \sum_{j=1}^{l_\mathsf{w}} \delta t_j s_j]_1$
3. $a \mapsto (1/r_1) a;\ b \mapsto r_1 b + r_1 r_2 [\delta]_2;\ c \mapsto c + r_2 a - r'[\gamma\left(1 + \sum_{i=1}^{l_\mathsf{w}} s_i\right)]_1$
4. **return** $((a, b, c), (c_0, c_1, \dots, c_{l_\mathsf{w}}, \mathsf{x}))$

Figure 3.3: Ext-Groth16: the black-box-extractable SAVER-inspired variant of Groth16. The relation $\mathcal{R}$ must assert that inputs on witness input wires $l \dots l + l_\mathsf{w}$ are small enough to be efficiently decryptable. $q_i(x)$ and $y_i(x)$ are as for Groth16, e.g. in Fig. 2.3.

$\underline{\mathsf{Sim}(\mathsf{td}, \mathbf{x} = \mathsf{x}_1 \dots \mathsf{x}_l):}$

$\mu, \nu, c_0, \dots, c_{l_\mathsf{w}} \xleftarrow{\$} \mathbb{Z}_p^*$

$(a, b, c) \leftarrow \left([\mu]_1, [\nu]_2, \left[\frac{\mu\nu - \alpha\beta - \gamma(\sum_{i=0}^{l} \mathsf{x}_i y_i(x) + \sum_{i=1}^{l_\mathsf{w}} c_i)}{\delta} \right]_1 \right)$

$\psi \leftarrow \left[\sum_{i=0}^{l_\mathsf{w}} t_i c_i \right]_1$

return $((a, b, c), \mathcal{CT} = ([c_0]_1, \dots, [c_{l_\mathsf{w}}]_1, \psi))$

$\underline{\mathsf{Ext}(\boldsymbol{\sigma}, \mathsf{td}_\mathsf{ext} = \{s_i\}_{i=1}^{l_\mathsf{w}}, \mathsf{x}, \pi = (\cdot, (c_0, c_1, \dots, c_{l_\mathsf{w}}, \cdot)):}$

for $i \in [1 \dots l_\mathsf{w}]$ **do**

  $[y_i(x)\mathsf{w}_i]_1 \leftarrow c_i - s_i c_0$

  $\mathsf{w}_i \leftarrow \mathsf{dlog}_{[y_i(x)]_1}([y_i(x)\mathsf{w}_i]_1)$

return $\mathsf{w}_1, \dots, \mathsf{w}_{l_\mathsf{w}}$

Figure 3.3: (cont.) Ext-Groth16: the black-box-extractable SAVER-inspired variant of Groth16.

Finally, we estimate the resulting performance parameters of Ext-Groth16. Construction CRS size (omitting constants) is $(m + 2n + 2l_\mathsf{w})\ \mathbb{G}_1$, and $(n + l_\mathsf{w})\ \mathbb{G}_2$. Proof size is $(l_\mathsf{w} + 4)\ \mathbb{G}_1$ and $1\ \mathbb{G}_2$, so $l_\mathsf{w} + 2$ times more $\mathbb{G}_1$ than in Groth16. Prover time is (omitting constants) $(m + 3n - l + 2l_\mathsf{w})\ E_1$ and $n\ E_2$. Verifier time is $l\ E_1$ and $(l_\mathsf{w} + 5)\ P$, so $l_\mathsf{w} + 2$ pairings more than in Groth16.

3.4.2.2 Security

We give a direct proof for the security of Ext-Groth16, as opposed to relying on the security of a transformation as for Int-Groth16. We prove computational zero-knowledge under the standard DDH assumption, as compared to a decisional polynomial assumption introduced and used in SAVER. The weak SE proof is structurally similar to the proof of Theorem 3.2.1: that is, we show that either $\mathcal{A}$ reuses a simulated proof (potentially randomizing it), or it does not use simulated data at all, and in that case we can extract the witness. The crucial difference now is that extractor Ext is black-box and operates by decrypting the ciphertext.

Theorem 3.4.2. *The Ext-Groth16 NIZK argument in Fig. 3.3 achieves perfect completeness; computational zero-knowledge under the DDH assumption; and black-box weak SE against algebraic adversaries under linear independence of* $U = \{u_i(X)\}_{i=0}^{l+l_\mathsf{w}}$*, and span independence between* U *and rest of* $u_i(X)$*.*

$\underline{\mathcal{S}'_{0,\boldsymbol{\sigma},\mathsf{td}}(\mathsf{x} = \mathsf{x}_1 \dots \mathsf{x}_l, \mathsf{w} = \mathsf{w}_1 \dots \mathsf{w}_{m-l})}$:

1. **if** $(\mathsf{x}, \mathsf{w}) \notin \mathcal{R}$ **then return** $\perp$
2. $\mu, \nu, r \xleftarrow{\$} \mathbb{Z}_p^*$; $c_0 \leftarrow r[\delta]_1$
3. $c_i \leftarrow r[\delta s_i]_1 + \mathsf{w}_i[y_{l+i}(x)]_1$ for $i \in [1 \dots l_{\mathsf{w}}]$
4. $(a, b, c) \leftarrow \left([\mu]_1, [\nu]_2, \left[\frac{\mu\nu - \alpha\beta - \gamma \sum_{i=0}^{l} \mathsf{x}_i y_i(x)}{\delta}\right]_1 + \sum_{i=1}^{l_{\mathsf{w}}} (\frac{c_i}{\delta})\right)$ *
5. $\psi \leftarrow \sum_{i=0}^{l_{\mathsf{w}}} t_i c_i$ *
6. **return** $((a, b, c), \mathcal{CT} \leftarrow (c_0, \dots, c_{l_{\mathsf{w}}}, \psi))$

Figure 3.4: Simulation oracle $\mathcal{S}'_{0,\boldsymbol{\sigma},\mathsf{td}}$ in $\mathcal{G}_1$

Theorem 3.4.2, Perfect Completeness. The validity of the statement is ensured by straightforward verification of the simulated ciphertext satisfying both verification equations. □

Theorem 3.4.2, Computational Zero-Knowledge. Let $\mathcal{A}$ be an arbitrary PPT adversary that makes q queries to the simulation oracle in the ZK game. We consider a sequence of games and prove that in each game adversary's advantage changes at most by a negligible amount. Let us denote the probability that $\mathcal{A}$ outputs 1 in $\mathcal{G}_x$ by ε_x and let ε_{DDH} denote the maximum distinguishing advantage in the DDH game.

$\underline{\mathcal{G}_0}$: This is the original game in Definition 2.6.5 when $b = 0$. That is, $\mathcal{A}$ can query the oracle $\mathcal{S}_{0,\boldsymbol{\sigma},\mathsf{td}}$ with inputs $(\mathsf{x}, \mathsf{w}) \in R$ and gets back proofs $\pi = \mathsf{Prove}(\boldsymbol{\sigma}, \mathsf{x}, \mathsf{w})$.

$\underline{\mathcal{G}_1}$: In this game, we change the oracle $\mathcal{S}_{0,\boldsymbol{\sigma},\mathsf{td}}$ to $\mathcal{S}'_{0,\boldsymbol{\sigma},\mathsf{td}}$ in Fig. 3.4 that uses the trapdoor td to simulate a, b, c, and ψ. Highlighted and marked with a "$*$" elements in the figure are changed compared to $\mathcal{S}_{0,\boldsymbol{\sigma},\mathsf{td}}$.

Let us argue that the probability that $\mathcal{A}$ outputs 1 in the $\mathcal{G}_0$ is the same as in the $\mathcal{G}_1$, i.e., $\varepsilon_0 = \varepsilon_1$. Firstly, $c_0, \dots, c_{l_{\mathsf{w}}}$ are computed the same way in both games. Since $\psi = \sum_{i=0}^{l_{\mathsf{w}}} t_i c_i = t_0[r\delta]_1 + \sum_{i=1}^{l_{\mathsf{w}}} t_i([r\delta s_i + \mathsf{w}_i y_{l+i}(x)]_1) = r[\delta(t_0 + \sum_{i=1}^{l_{\mathsf{w}}} t_i s_i)]_1 + \sum_{i=1}^{l_{\mathsf{w}}} \mathsf{w}_i[y_{l+i} t_i]_1$, then in both games ψ is uniqely determined by $c_0, \dots, c_{l_{\mathsf{w}}}$. Finally, a, b are uniformly random in both games and c is the unique value determined by $c_0, \dots, c_{l_{\mathsf{w}}}, a$ and b. It is easy to verify that c in Fig. 3.4 does indeed satisfy the verification equation. Hence, output of $\mathcal{S}'_{0,\boldsymbol{\sigma},\mathsf{td}}$ has the same distribution as the output of $\mathcal{S}_{0,\boldsymbol{\sigma},\mathsf{td}}$. From this it follows that $\varepsilon_0 = \varepsilon_1$.

$\underline{\mathcal{G}_{2:(1,1)}}$: For convenience, let us denote the ciphertext elements of different queries by $c_{i,j}$ where $i \in [0, l_\mathsf{w}]$ and $j \in [1, q]$. We change the oracle in the previous game such that $c_{1,1}$ is sampled randomly.

We show that $|\varepsilon_1 - \varepsilon_2|$ is bounded by the probability of breaking the DDH assumption. Let us construct an adversary $\mathcal{B}$ that uses $\mathcal{A}$ to distinguish DDH tuples. The adversary $\mathcal{B}$ gets as an input $[z_x, z_y, z]_1$ where $z_x, z_y \xleftarrow{\$} \mathbb{Z}_p$ and either $z = z_x z_y$ or $z \xleftarrow{\$} \mathbb{Z}_p$. Next, $\mathcal{B}$ samples $\boldsymbol{\sigma}$ and $\mathtt{td}$ except that $[s_1]_1 = [z_x]_1$ and thus that element of $\mathtt{td}$ is unknown. The adversary $\mathcal{B}$ continues by running $\mathcal{A}$ on the input $\boldsymbol{\sigma}$ while simulating the query oracle. The query oracle behavies like $\mathcal{S}'_{0,\boldsymbol{\sigma},\mathsf{td}}$ except on the first query it outputs $c_{0,1} = \delta[z_x]_1$, $c_{1,1} = \delta[z]_1 + \mathsf{w}_1[y_{l+1}(x)]_1$, $c_{1,2} = (\delta s_2)[z_x]_1 + \mathsf{w}_2[y_{l+2}(x)]_1, \ldots, c_{1,l_\mathsf{w}} = (\delta s_{l_\mathsf{w}})[z_x]_1 + \mathsf{w}_{l_\mathsf{w}}[y_{l+l_\mathsf{w}}(x)]_1$. Note that if z is uniformly random then $c_{1,1}$ is uniformly random as in $\mathcal{G}_{2:(1,1)}$, but when $z = z_x z_y$, then $c_{1,1}$ is a valid ciphetext element as in $\mathcal{G}_1$, where z_x takes the role of r, and z_y of s_1. Therefore, $|\varepsilon_1 - \varepsilon_{2:(1,1)}| \leq \varepsilon_{DDH}$.

<u>$\mathcal{G}_{2:(i,j)}$ for $i \in [2 \ldots l_\mathsf{w}], j \in [1 \ldots q]$</u>: We continue with a similar strategy as in $\mathcal{G}_{2:(1,1)}$. Namely, we change the oracle of the previous game by sampling $c_{i,j}$ uniformly randomly. We use the same reduction idea as in $\mathcal{G}_{2:(1,1)}$ and show that $|\varepsilon_{2:(i-1,j)} - \varepsilon_{2:(i,j)}| \leq \varepsilon_{DDH}$ (or $|\varepsilon_{2:(l_\mathsf{w},j-1)} - \varepsilon_{2:(1,j)}| \leq \varepsilon_{DDH}$).

Finally, $\mathcal{G}_{2:(l_\mathsf{w},q)}$ is the original ZK game where $\mathcal{A}$ has an oracle access to the simulator presented in Fig. 3.3, which produces all $c_{i,j}$ uniformly random; this is equivalent to honest encryption of a random message. It follows that the advantage that $\mathcal{A}$ breaks ZK is bounded by $l_\mathsf{w} \cdot q \cdot \varepsilon_{DDH}$. □

Theorem 3.4.2, Weak BB SE. The knowledge soundness (KS) theorem of [Lee et al., 2019] shows how to reduce KS of the SAVER scheme (with two verification equations) to the KS of Groth16. This theorem is also structured as a reduction, but to a weak white-box SE of Groth16 that we have proved in Section 3.2.

Additionally to the set of elements $\mathcal{A}$ sees in the proof of Theorem 3.2.1, we have two more: (1) the CRS is extended with one embedded public key, hence we have elements that depend on the t_i and s_i trapdoors; (2) simulation queries now also produce random ciphertexts $c_{i,j}$ (also, simulated C depends on these ciphertexts, which changes $C_{9,i}$ element).

We write out the representation of A and B (C, Ψ, C_i follow the same pattern as A) from the verification equation as the linear combination of the public CRS and new simulated proof elements:

$$
\begin{aligned}
A = {} & A_1\alpha + A_2\beta + A_3\delta \\
& + \sum_{i=0}^{n-1} A_{4,i}x^i + \sum_{i=0}^{l+l_{\mathsf{w}}} A_{5,i}y_i(x) + \sum_{l+l_{\mathsf{w}}+1}^{m} A_{6,i}\frac{q_i(x)}{\delta} + \sum_{i=0}^{n-2} A_{7,i}\frac{x^i t(x)}{\delta} + \\
& + \sum_{i=1}^{q}\Big(A_{8,i}\mu_i + A_{9,i}\Big(\frac{\mu_i\nu_i - \alpha\beta - \gamma(\sum_{j=0}^{l}\mathsf{x}_{i,j}y_j(x) + \sum_{j=r}^{l_{\mathsf{w}}} c_{i,j})}{\delta}\Big)\Big) \\
& + \sum_{i=1}^{l_{\mathsf{w}}} A_{10,i}\delta s_i + \sum_{i=1}^{l_{\mathsf{w}}} A_{11,i}t_i y_{l+i}(x) + A_{12}\delta(t_0 + \sum_{i=1}^{l_{\mathsf{w}}} t_i s_i) + A_{13}\gamma(1 + \sum_{i=1}^{l_{\mathsf{w}}} s_i) + \\
& + \sum_{i=1}^{q}\Big(\sum_{j=0}^{l_{\mathsf{w}}} A_{14,i,j}c_{i,j} + A_{15,i}\sum_{j=0}^{l_{\mathsf{w}}} t_j c_{i,j}\Big)\Big) \\
B = {} & B_1\beta + B_2\gamma + B_3\delta + \sum_{i=0}^{n-1} B_{4,i}x^i + \sum_{i=1}^{q} B_{5,i}\nu_i + \sum_{i=0}^{l_{\mathsf{w}}} B_{6,i}t_i
\end{aligned}
$$

We will refer to the terms $A_1 \dots A_{9,i}$ and $B_1 \dots B_{5,i}$ as “first category” (since they are used in the SE proof), and the other terms are, correspondingly, “second category”. We use the same indexing for the first category coefficients as in the SE proof for compatibility; the only difference is that there are fewer $A_{6,i}$ coefficients, and $A_{5,i}$ ranges to $l + l_{\mathsf{w}}$ and not l. Technically, this change is merely syntactical: we could assume secret inputs are part of the (hidden) instance, which would leave the coefficients as they were before (by setting $l = l + l_{\mathsf{w}}$).

We will show that it is possible to extract the witness from the coefficients an algebraic $\mathcal{A}$ returns for the ciphertexts. At the same time, Ext in Fig. 3.3 is black-box. The security proof will use the white-box extracted coefficients, but they are equal to those returned by Ext because of correctness of the encryption scheme.

We now analyse the first verification equation of Ext-Groth16:

$$\prod_{i=0}^{l_{\mathsf{w}}}[C_i]_1[t_i]_2 = [\Psi]_1[1]_2 \quad \text{or, in exponent form:} \quad C_0t_0 + \ldots + C_{l_{\mathsf{w}}}t_{l_{\mathsf{w}}} = \Psi$$

It is immediately clear that Ψ can only be composed of elements that contain t_i, since they are in the immutable part of the left hand side:

$$\prod_{i=0}^{l_{\mathsf{w}}} C_i t_i = \sum_{i=1}^{l_{\mathsf{w}}} \Psi_{11,i}t_i y_{l+i}(x) + \Psi_{12}\delta(t_0 + \sum_{i=1}^{l_{\mathsf{w}}} t_i s_i) + \sum_{i=1}^{q} \Psi_{15,i}(\sum_{j=0}^{l_{\mathsf{w}}} t_j c_{i,j})$$

Now, we derive the restrictions on the ciphertext coefficients C_i. Going through the other elements of the equation, to balance properly, each C_i must consist of either: (1) some strictly other t_j (clearly we cannot have t_i^2 in the equation; in particular, $C_{i,12} = C_{i,15,j} = 0$ for all $j \in [1 \dots l]$ because of that); or (2) $y_{l+i}(x)$ for $i > 0$; or (3) δ for $i = 0$ and $s_i\delta$ for $i > 0$; or (4) simulated ciphertexts $\{c_{j,i}\}_{j=1}^{q}$. Substituting it into the equation:

$$\begin{aligned}
&\Big(C_{0,3}\delta + \sum_{i=1}^{l_{\mathsf{w}}} C_{0,11,i} t_i y_{l+i}(x) + \sum_{i=1}^{q}\sum_{j=0}^{l_{\mathsf{w}}} C_{0,14,i,j} c_{i,j}\Big) t_0 \\
&+ \sum_{j=1}^{l_{\mathsf{w}}} \Big(C_{j,10,j} s_j \delta + C_{j,5,j} y_{l+j}(x) + \sum_{i=1, i\neq j}^{l_{\mathsf{w}}} C_{j,11,i} t_i y_{l+i}(x) + \sum_{i=1}^{q}\sum_{k=0}^{l_{\mathsf{w}}} C_{j,14,i,k} c_{i,k}\Big) t_j \\
&\qquad = \sum_{i=1}^{l_{\mathsf{w}}} \Psi_{11,i} t_i y_{l+i}(x) + \Psi_{12}\delta(t_0 + \sum_{i=1}^{l_{\mathsf{w}}} t_i s_i) + \sum_{i=1}^{q} \Psi_{15,i}(\sum_{j=0}^{l_{\mathsf{w}}} t_j c_{i,j})
\end{aligned}$$

From δt_0 we obtain $C_{0,3} = \Psi_{12}$, and for $i > 0$ from $\delta t_j s_j$ we get $\Psi_{12} = C_{j,10,j}$. Now, looking at $t_j y_{l+j}(x)$ (in fact, on $t_j x/\gamma, \alpha t_j x/\gamma, \beta t_j x/\gamma$), we also get $C_{j,5,j} = \Psi_{11,j}$. From each $t_0 c_{i,j}$ we derive that $C_{0,14,i,0} = \Psi_{15,i}$, and all other $C_{0,14,i,j} = 0$; similarly only $C_{j,14,i,j} = \Psi_{15,i}$. Finally, notice that $t_j t_0$ for $j > 0$ cannot be balanced by anything from C_j ($C_{j,11,i}$ start from $i = 1$) or Ψ, so all $C_{0,11,i} = 0$. Applying these changes, we derive:

$$\begin{aligned}
&\Big(\Psi_{12}\delta + \sum_{i=1}^{q} \Psi_{15,i} c_{i,0}\Big) t_0 + \sum_{j=1}^{l_{\mathsf{w}}} \Big(\Psi_{12} s_j \delta + \Psi_{11,j} y_{l+j}(x) + \sum_{i=1, i\neq j}^{l_{\mathsf{w}}} C_{j,11,i} t_i y_{l+i}(x) + \sum_{i=1}^{q} \Psi_{15,i} c_{i,j}\Big) t_j \\
&\qquad = \sum_{i=1}^{l_{\mathsf{w}}} \Psi_{11,i} t_i y_{l+i}(x) + \Psi_{12}\delta(t_0 + \sum_{i=1}^{l_{\mathsf{w}}} t_i s_i) + \sum_{i=1}^{q} \Psi_{15,i}(\sum_{j=0}^{l_{\mathsf{w}}} t_j c_{i,j})
\end{aligned}$$

The coefficients of the resulting equation represent the following logic: (1) original honest ciphertexts ($\Psi_{11,j}$ are encrypted witness wires, and Ψ_{12} is randomness), (2) homomorphically added simulation ciphertexts ($\Psi_{15,i}$ for $i = 1 \dots q$), and (3) linear combination on the left hand side (nonzero $C_{j,11,i}$).

We now argue that all $C_{j,11,i} = 0$, and thus no nontrivial linear combination is possible. Assuming the contrary, and analysing monomial $t_{k_1} t_{k_2}$ for some pair of positive indices $k_1 \neq k_2$ (both $\in [1 \dots l_{\mathsf{w}}]$), we have $C_{k_1,11,k_2} y_{l+k_2}(x) + C_{k_2,11,k_1} y_{l+k_1}(x) = 0$. This, in turn, implies, simultaneously, $C_{k_1,11,k_2} f_{l+k_2}(x) + C_{k_2,11,k_1} f_{l+k_1}(x) = 0$, for $f_i(X) = v_i(X), u_i(X), w_i(X)$ (viewing $\alpha x, \beta x, x$). But we assumed $\{u_i(X)\}_{i=l+1}^{l_{\mathsf{w}}}$ to be linearly independent, and therefore all $C_{j,11,i} = 0$.

The resulting view on the equation is now:

$$\Big(\Psi_{12}\delta + \sum_{i=1}^{q}\Psi_{15,i}c_{i,0}\Big)t_0 + \sum_{j=1}^{l_{\mathsf{w}}}\Big(\Psi_{12}s_j\delta + \Psi_{11,j}y_{l+j}(x) + \sum_{i=1}^{q}\Psi_{15,i}c_{i,j}\Big)t_j$$
$$= \sum_{i=1}^{l_{\mathsf{w}}}\Psi_{11,i}t_iy_{l+i}(x) + \Psi_{12}\delta(t_0 + \sum_{i=1}^{l_{\mathsf{w}}}t_is_i) + \sum_{i=1}^{q}\Psi_{15,i}(\sum_{j=0}^{l_{\mathsf{w}}}t_jc_{i,j})$$

As we can see, it deviates from the proof of SAVER KS in exactly one aspect: $\mathcal{A}$ can combine its message encryption with the simulated zero-ciphertexts homomorphically. This does not give $\mathcal{A}$ any real power: we will show that this combination cannot satisfy the *second* verification equation, because $\mathcal{A}$ cannot produce the "ciphertext randomness cancelling value" for C.

After showing how exactly C_i are restricted, our next step is substituting their general form into the second equation which corresponds to the verification equation of Groth16:

$$AB - \alpha\beta - \Bigg(\sum_{i=0}^{l}\mathsf{x}_iy_i(x) + \Big(\Psi_{12}\delta + \sum_{i=1}^{q}\Psi_{15,i}c_{i,0}\Big)$$
$$+ \sum_{j=1}^{l_{\mathsf{w}}}\Big(\Psi_{12}s_j\delta + \Psi_{11,j}y_{l+j}(x) + \sum_{i=1}^{q}\Psi_{15,i}c_{i,j}\Big)\Bigg)\gamma - C\delta = 0$$

We now follow the SE proof reduction: the block of 11 equations from which it starts remains the same. The equation extracted from $\alpha\beta$ is not affected by the change of terms since no additional $\alpha\beta$ terms are created either by second category coefficients, or by the ciphertext terms. The other 10 equations depend either on μ_i or ν_i ($\mu_i\nu_j, \mu_i, \nu_i, \ldots, \nu_i\delta$), and they are exactly the same in our case too, since (1) they do not contain $A_{5,i}$ and $A_{6,i}$, (2) the second category monomials do not contain μ_i or ν_i, and (3) the ciphertext coefficients do not either. Hence, in both cases all $A_{9,i} = 0$, and we follow the branching of the SE proof:

- *Non-simulation case.* All the simulation elements are zero: $A_{9,i} = A_{8,i} = B_{5,i} = C_{8,i} = C_{9,i} = 0$, and thus we have as before $A_1B_1 = 1$. From β^2 and $\alpha\delta$, we get $A_2 = B_2 = 0$.

  From Equation ($c_{i,0}\gamma$): $A_{9,i}B_{4,0} + A_{14,i,0}B_2 - \Psi_{15,i} - C_{9,i} = 0$, so $\Psi_{15,i} = 0$ — this means $\mathcal{A}$ cannot add simulated ciphertexts into a non-simulated one. Indeed, to balance out simulated $c_{i,j}$ $\mathcal{A}$ would need to add the cancelling coefficient to C, but it cannot do that since it is part of $C_{9,i}$ which is zero.

We easily cancel all the second category terms from A, B. From Equation ($\beta\delta s_i$) we have $A_{10,i} = 0$, from monomials $\beta t_i x^i$ we get $\sum_{i=1}^{l_\mathsf{w}} A_{11,i} t_i y_{l+i}(x) = 0$ (coefficients may be nonzero since forming a linear combination may be possible). From Equation ($\delta\beta t_0$), $A_{12}B_1 = 0$, so $A_{12} = 0$. From Equation ($\gamma\beta$): $A_2B_2 + A_{13}B_1 = 0$, so $A_{13} = 0$. At the same time, from Equation ($\beta c_{i,j}$): $A_{14,i,j}B_1 = 0$, and from Equation ($\beta c_{i,0}t_0$): $A_{15,i}B_1 = 0$, so all $A_{14,i,j} = A_{15,i} = 0$. Considering Equation ($\alpha t_i$) we get $A_1 B_{6,i} = 0$, so $B_{6,i} = 0$.

To characterise Ψ_{12} we must look at Equation ($\delta\gamma$): $A_3B_2 + A_{13}B_3 - \Psi_{12} - C_{13} = 0$. Since $B_2 = A_{13} = 0$, we derive $C_{13} = -\Psi_{12}$ — that is, the cancelling coefficient in C must be balanced out by the ciphertext randomness, which is in line with honest proof generation logic.

Other second category terms in C also cancel: $C_{10,i} = 0$ because of $\delta^2 s_i$; $C_{11,i} = 0$ from $\delta t_i x^i$; $C_{12,i} = 0$ because of $\delta^2 t_0$; and $C_{14,ij}$ with $C_{15,i}$ are zero from $\delta c_{i,j}$ and $\delta c_{i,j}t_j$ correspondingly.

Now the verification equation looks like as if $\mathcal{A}$ uses a single honestly constructed ciphertext:

$$AB - \alpha\beta - \left(\sum_{i=0}^{l} \mathsf{x}_i y_i(x) + \Psi_{12}\delta + \sum_{j=1}^{l_\mathsf{w}} \Big(\Psi_{12} s_j \delta + \Psi_{11,j} y_{l+j}(x) \Big) \right) \gamma + C\delta = 0$$

Moreover, as we showed, Ψ_{12} (ciphertext cancelling term coefficient) is cancelled out by C_{13} — the only nonzero from the second category coefficients. So from here we can reduce to the basic Groth16, with the only difference that now $A_{5,i}$ has more wires and $A_{6,i}$ has less. We show that this minor change does not significantly affect the proof of Groth16 KS.

For all the equations in the KS proof until we get to critical equations (that is, $(\beta^2/\delta, \beta\alpha/\delta, \beta/\delta, 1/\delta, \beta^2/\gamma, \beta\alpha/\gamma, \beta/\gamma, 1/\gamma)$) the only change that happens is that whenever a sum with $A_{5,i}$ appears it now spans to $l + l_\mathsf{w}$ not to l, and whenever a sum with $A_{6,i}$ appears, it goes from $l + l_\mathsf{w} + 1$ to m, not from $l + 1$. This is easy to verify, since the only new monomials that we have are $\delta\gamma$ (with Ψ_{12} and with C_{13}; is not part of the monomials listed), $\delta\gamma s_i$ (similarly), and $\Psi_{11,j} y_{l+j}(x)\gamma$ only affect critical equations. And other second category elements are zero.

Looking at the third critical coefficient (corresponding to powers of x only) we see almost the same equation as in the KS proof, except now $\Psi_{11,i}$ are instead of first l_{w} wires of $C_{6,i}$. No other new terms are added, and coefficients with B_2, $A_{6,i}$ and $A_{7,i}$ are cancelled as before, which gives:

$$\Big(\sum_{i=0}^{n-1} A_{4,i}x^i\Big)\Big(\sum_{i=0}^{n-1} B_{4,i}x^i\Big) = \sum_{i=0}^{l} \mathsf{x}_i w_i(x) \\ + \sum_{i=l+1}^{l+l_{\mathsf{w}}} \Psi_{11,i-l} w_i(x) + \sum_{i=l+l_{\mathsf{w}}+1}^{m} C_{6,i} w_i(x) + \sum_{i=0}^{n-2} C_{7,i} x^i t(x)$$

To argue that $A_{4,i}$ and $B_{4,i}$ form $u_i(x)$ and $v_i(x)$ sets, we, as in the KS proof, look at α and β:

$$\beta: \quad \Big(\sum_{i=0}^{n-1} A_{4,i}x^i\Big)B_1 = \sum_{i=0}^{l} \mathsf{x}_i u_i(x) + \sum_{i=l+1}^{l+l_{\mathsf{w}}} \Psi_{11,i-l} u_i(x) + \sum_{i=l+l_{\mathsf{w}}}^{m} C_{6,i} u_i(x)$$

$$\alpha: \quad \Big(\sum_{i=0}^{n-1} B_{4,i}x^i\Big)A_1 = \sum_{i=0}^{l} \mathsf{x}_i v_i(x) + \sum_{i=l+1}^{l+l_{\mathsf{w}}} \Psi_{11,i-l} v_i(x) + \sum_{i=l+l_{\mathsf{w}}}^{m} C_{6,i} v_i(x)$$

Therefore we trivially conclude, substituting the last two equations into the previous one, and extracting from $\Psi_{11,j}$, as we extracted from $C_{6,j}$ instead in the KS case.

- *Simulation Case*. This branch is characterised by $\mathcal{A}$ reusing the simulated proof number k with $C_{9,k} = 1$; as we showed in Groth16 weak SE proof, all other $A_{8,i}, B_{5,i}$ and $C_{9,i}$ are zero. From the very same block of 11 equations we derive, as before: $A_1 = A_2 = A_3 = B_1 = B_2 = 0$.

  Since simulation "ciphertext cancelling" terms are embedded with simulated C, and only $C_{9,k}$ is nonzero, $\mathcal{A}$ cannot use any other set of ciphertexts than $\{c_{k,i}\}_i$. Formally, we show it by looking at Equation ($\gamma c_{i,0}$): $A_{9,i}B_{4,0} + A_{14,i,0}B_2 - \Psi_{15,i} + C_{9,i} = 0$, so we conclude that $\Psi_{16,i} = 0$ for $i \neq k$, and for $i = k$ since all $A_{9,i} = 0$ we get $\Psi_{15,k} = A_{9,k}B_{4,0} + C_{9,k} = 0 + 1 = 1$. That is, $\mathcal{A}$ uses exactly one simulated ciphertext vector, unmodified.

  We now cancel all the second category terms for A and B, looking at combinations of coefficients in A with nonzero $B_{5,k}(\nu_k)$ and the coefficients of B with $A_{8,k}(\mu_k)$; all they are extracted from AB only. From Equation ($\delta s_i \nu_k$): $A_{10,i}B_{5,k} = 0$, so $A_{10,i} = 0$. Looking at monomials $\nu_k t_i \alpha x, \nu_k t_i \beta x, \nu_k t_i x$ simultaneously: $A_{11,i} t_i y_{l+i}(x) B_{5,k} \nu_k = 0$, hence $A_{11,i} = 0$. From Equation ($\delta t_0 \nu_k$),

$A_{12}B_{5,k} = 0$, so $A_{12} = 0$. From Equation ($\gamma\nu_k$): $A_{13}B_{5,k} = 0$, so $A_{13} = 0$. Monomials with $c_{i,j}\nu_k$ do only appear in $A_{14,i,j}, A_{15,i}$ since $A_{9,i} = 0$, so from Equations $c_{i,j}\nu_k, c_{i,j}t_j\nu_k$ all these coefficients are zero in a similar manner. Finally, we cancel $B_{6,i}$ by looking at $t_i\mu_k$ which gives us $A_{8,k}B_{6,i} = 0$.

Regarding Ψ_{12}, as in the non-simulation case, we look at Equation ($\delta\gamma$): $A_{13}B_3 - \Psi_{12} - C_{13} = 0$, which simplifies to $\Psi_{12} = -C_{13}$. This means that $\mathcal{A}$ can indeed add its own randomness to C, because it can cancel it out exactly with C_{13}.

Except for C_{13} we can cancel all the other second category terms of C. $C_{10,i}$ is zero because $\delta^2 s_i$ are only balanced by $A_{10,i}B_3$ but $A_{10,i} = 0$; the same thing happens to $C_{11,i}$ because of $\delta t_i(\alpha x + \beta x + x)$, to C_{12} ($\delta^2 t_0$ and $\delta^2 t_i s_i$), to $C_{14,i,j}$ ($\delta c_{i,j}$), and to $C_{15,i}$ ($\delta c_{i,j}t_j$).

We have now cancelled all the second category terms of A, B, C except for C_{13} balancing out Ψ_{12}. Now it is possible to proceed with the reduction exactly as in the second branch of the Groth16 weak SE proof, having in mind the similar difference with $A_{5,i}, A_{6,i}$ elements explained in the first branch of the current proof. In particular, we argue that $\Psi_{11,j}$ cancel from the third critical equation:

$$0 = \sum_{i=0}^{l} \mathsf{a}_i u_i(x) + \sum_{i=l+1}^{l+l_{\mathsf{w}}} \Psi_{11,i-l} u_i(x) + \sum_{i=l+l_{\mathsf{w}}+1}^{m} C_{6,i} u_i(x) - \sum_{i=0}^{l} \mathsf{a}_{k,i} u_i(x)$$

Similarly to Groth16 weak SE, $\sum_{i=l+l_{\mathsf{w}}+1}^{m} C_{6,i}u_i(x) = 0$ because it is linearly independent from all the $0 \ldots l + l_{\mathsf{w}}$ input wires. Then, because input wires are independent, all $\Psi_{11,i-1} = 0$ (which forbids adding nonzero honest ciphertexts), and $\mathsf{a}_i = \mathsf{a}_{k,i}$. Hence, $\mathcal{A}$ has reused the simulated proof number k, but potentially with ciphertext randomization (Ψ_{12}) additionally to the (A, B, C) randomization of Groth16.

□

Lemma 3.4.3. *The Ext-Groth16 NIZK is rerandomizable with* Rand *in Fig. 3.3.*

Proof. Follows directly from rerandomizability of SAVER in [Lee et al., 2019]. □

3.5 Performance

In this section, we evaluate the efficiency of Int-Groth16 and Ext-Groth16. First, in Table 3.1, we give a high-level comparison of Groth16 and (the most efficient) C∅C∅ black-box SE transformation [Kosba et al., 2015, Section 4]. It shows the asymptotic dependence of the performance metrics on the witness size l_{w} and the blow-up of the QAP size due to the use of cryptographic primitives for the transformation. $\mathsf{Enc}_{l_{\mathsf{w}}}$ denotes an encryption scheme with sufficiently large plaintext size to encrypt the witness. We note that even for Ext-Groth16 a small circuit modification is required, and therefore m grows by $2l_{\mathsf{w}}$ bits, and n grows by l_{w}; additionally, l_{w} wires for Ext-Groth16 have 6 times less capacity than for Int-Groth16 and C0C0. Clearly, in Table 3.1, an overhead of C∅C∅ in CRS size and prover time is strictly bigger than in both constructions we suggest, due to the use of PRF and commitment scheme, and Ext-Groth16 encryption overhead (thus proof size and verification time) is bigger than in first two transformations because of the expansion factor.

We also estimate the *concrete* performance of our two black-box constructions, along the same four performance parameters defined in Table 3.2, as depending on the bit-size of the encrypted witness. For both NIZKs we will use a 255-bit BLS12-381 curve, defined over a 381 bit prime field. Let us assume that witness size is B_{w} bits, and it is provided in bit-decomposed form in the original circuit. We aim to optimize proof size, which is important for SNARKs, and thus will only consider encrypting secret inputs at the maximum possible capacity (e.g. we do not encrypt individual bits); the two approaches have different block capacities, so the number of plaintext (and ciphertext) blocks is different in both cases. For Int-Groth16, block size is 248 bits, where the 6 remaining bits are reserved for Koblitz [Koblitz, 1987] message embedding padding. For Ext-Groth16 we split the plaintext in 43-bit blocks, thus assuming that we can solve 43-bit discrete logarithm for black-box extraction. This explains Ext-Groth16 expansion factor of $6 = \lceil 248/43 \rceil$. We base our circuit design estimates, which are especially relevant to Int-Groth16, on zcash implementation, description of which is provided in [Hopwood et al., 2022] (Section "Circuit Design").

Table 3.1: A comparison of Groth16 with the overhead of C∅C∅ framework and our constructions. Constants are omitted in the case of CRS size and prover's computation. $\mathbb{G}_1$ and $\mathbb{G}_2$: group elements, E: exponentiations and P: pairings. l_w is the number of secret input wires. $e(l_\mathsf{w}) = |\mathsf{Enc}_{l_\mathsf{w}}|$, $c(l_\mathsf{w}) = e(l_\mathsf{w}) + |\mathsf{Com}| + |\mathsf{Prf}|$, where $|\mathsf{Op}|$ denotes the number of *constraints* required for the operation $\mathsf{Op} \in \{\mathsf{Enc}_{l_\mathsf{w}}, \mathsf{Com}, \mathsf{Prf}\}$. $e_i(l_\mathsf{w}), c_i(l_\mathsf{w})$ denote an additional increase in *input wires* (counted in m, but not in n). k_e is Enc expansion factor, can be assumed ≤ 2. The highlighted cells indicate the best efficiency. Top-level parentheses between expressions and units are omitted for better readability.

Construction	Security	CRS	Proof	Prover	Verifier
Groth16, Sec. 3.2	KS, Weak WB-SE	$m + 2n\ \mathbb{G}_1$ $n\ \mathbb{G}_2$	$2\ \mathbb{G}_1$ $1\ \mathbb{G}_2$	$m + 3n - l\ E_1$ $n\ E_2$	$l\ E_1$ $3\ P$
Groth16 + [Kosba et al., 2015]	Weak BB-SE	$+3c(l_\mathsf{w}) + c_i(l_\mathsf{w})\ \mathbb{G}_1$ $+c(l_\mathsf{w})\ \mathbb{G}_2$	$+k_e l_\mathsf{w}\ \mathbb{G}_1$	$+4c(l_\mathsf{w}) + c_i(l_\mathsf{w}) - O(l_\mathsf{w})\ E_1$ $+c(l_\mathsf{w})\ E_2$	$+k_e l_\mathsf{w}\ E_1$
Int-Groth16, Sec. 3.4.1	Weak BB-SE	$+3e(l_\mathsf{w}) + e_i(l_\mathsf{w})\ \mathbb{G}_1$ $+e(l_\mathsf{w})\ \mathbb{G}_2$	$+k_e l_\mathsf{w}\ \mathbb{G}_1$	$+4e(l_\mathsf{w}) + e_i(l_\mathsf{w}) - O(l_\mathsf{w})\ E_1$ $+e(l_\mathsf{w})\ E_2$	$+k_e l_\mathsf{w}\ E_1$
Ext-Groth16, Sec 3.4.2	Weak BB-SE	$+36 l_\mathsf{w}\ \mathbb{G}_1$ $+12 l_\mathsf{w}\ \mathbb{G}_2$	$+6 l_\mathsf{w} + 2\ \mathbb{G}_1$	$+42 l_\mathsf{w}\ E_1$ $+12 l_\mathsf{w}\ E_2$	$+6 l_\mathsf{w} + 2\ P$

3.5.0.1 Concrete Performance of Int-Groth16

The performance overhead of Int-Groth16 compared to Groth16 depends mostly on the increase in circuit size — we must implement the encryption scheme itself, and also the infrastructure that converts the input (plaintext) data to the desired form, which we discuss first. We remind that JubJub forms a 252 bit group over 255 bit prime field, which is equal to the group size of BLS, allowing seamless integration of one into another.

Plaintext embedding into the JubJub curve — that is, converting plaintext blocks into JubJub points — is the main technical challenge, which we solve using the approach of Koblitz [Koblitz, 1987] to overcome it. To embed a plaintext block w_i of bit-length $(254 - \log_2 \kappa)$ into the curve we reserve a padding p_i for a nonce of length $\log_2 \kappa$: with probability $1 - 2^{\kappa}$ the concatenation $w_i \| p_i$ is a valid x-coordinate for some p. Because of how lengths are chosen, $m_i \| p_i$ is always smaller than the prime field of JubJub. For practical purposes it is enough to reserve 6 bits for the padding, which gives $\kappa = 64$, leaving 248 bits for the message block, and thus $l_{\mathsf{w}} = \lceil B_w / 248 \rceil$. To avoid issues with completeness — since now it is possible, with negligible probability, that some w_i does not have a suitable padding — we allow a fallback mechanism [Muralidhara and Sen, 2007], in which a random blinding b_i is chosen, and the algorithm is repeated for $w_i + b_i$, and b_i is attached to the ciphertext in clear. To avoid attacks where $\mathcal{A}$ finds non-encodable witnesses, we generate this b_i every single time (otherwise the presence of nonzero b_i may leak something about the message). From the circuit side, one extra wire per plaintext block is required to contain b_i, but it takes no extra mul-gates, since any gate that uses w_i can use $w_i + b_i$ for free. We will denote this procedure by $\mathsf{Embed}(w_i, b_i, p_i) = ((w_i \oplus b_i) \parallel p_i, y)$, where y coordinate is computed from x. Instead of computing y in the circuit, we pass it through the intermediate secret wires, and just check that (x, y) is on the curve, which takes just 4 constraints.

The second issue is that we must verify that circuit inputs corresponding to w_i are of a right bit-size in order to guarantee correctness of decryption (used in extraction). The standard way of solving this is to represent the values as bit-vectors of the needed size. That is, for each encrypted element $e = w_i$ we supply its bit decomposition $\{e_i\}_{i=0}^{n-1}$ explicitly and assert that $e = \sum 2^i e_i$, which certifies that e is a n bit value. This takes n gates to ensure each e_i is a binary value, and then one gate is needed to combine them all. Converting from bit-decomposed representation to any other base is inexpensive, and takes one gate per digit in the chosen base, so we assume that application (as opposed to the encryption-correctness check) is taking each w_i in bit-decomposed representation. To simplify the comparison, we assume that this representation is used in the original non-transformed circuit too, so we omit boolean-checks on w_i from both estimates.

We now discuss the encryption scheme itself. To verify the ciphertext $c_0, c_1, \ldots, c_{l_\mathsf{w}}$ we must check that $c_0 = r[1]_1$ and $c_i = r[s_i]_1 + \mathsf{Embed}(w_i, b_i, p_i)$. All the JubJub exponentiations here take 750 multiplication gates for each full-bit exponent — we assume that public key points are embedded into the circuit, thus we use an algorithm for fixed-base exponentiation.

- Checking c_0 requires 254 constraints for bit-range check, 750 gates for fixed-base exponentiation, 2 constraints to compare both coordinates of c_0 to the computed ones, and to perform $\mathsf{onCurve}$ check for c_0 coordinates (4 gates); together it takes 1010 constraints. The number of *additional input wires* that we introduce (for r and c_i) is 256, which we include in computation of m but not n.

- For the c_i checks we must additionally perform embedding and sum the embedded point with $r[s_i]$. For the embedding, our circuit accepts bit-decomposed w_i, b_i, p_i and y (second coordinate of embedded point). Bit-checking b_i, p_i (w_i is excluded, r is bit-checked once and counted for c_0 check already) takes $248 \cdot 2 + 6$ constraints. In order to optimize l we will pass b_i as field elements through public inputs and perform an equality-check with the bit-decomposed witness-passed b_i; this takes 1 constraint. Next, we must bit-add w_i and b_i, which takes 250 constraints. We can now pack the bits corresponding to $x = (w_i \oplus b_i) \parallel p_i$ into a field element with a single gate, and it takes 4 gates to verify $\mathsf{onCurve}(x, y)$. We sum the value with $r[s_i]$ (this takes 750 constraints to compute), which takes extra 6 constraints. On-curve check for c_i and comparison of c_i with the computed points takes 6 gates. In total, we get 1273 constraints. The number of additional input wires is 258 (b_i, p_i, y, c_i).

Combining it all together, n increases by $\Delta_n = 1010 + 1273 l_\mathsf{w}$, m increases by $\Delta_m = \Delta_n + 256 + 258 l_\mathsf{w} = 1266 + 1531 l_\mathsf{w}$, and l increases by $\Delta_l = 3 l_\mathsf{w}$ because of public blinding factors (l_w field elements) and two values for c_i.

We now analyse the NIZK parameters we compare along:

1. *CRS size.* We get extra $\Delta_m + 2\Delta_n = 3286 + 4077 l_{\mathsf{w}}\ \mathbb{G}_1$, and $\Delta_n = 1010 + 1273 l_{\mathsf{w}}\ \mathbb{G}_2$. Converting it to B_w, extra $3286 + 16.4 B_w\ \mathbb{G}_1$ and $1010 + 5.1 B_w\ \mathbb{G}_2$.
2. *Proof size.* Using ElGamal we have $l_{\mathsf{w}} + 1$ points per message block, and l_{w} 248-bit blinding factors. This results in additional $\lceil B_w/248 \rceil + 1\ \mathbb{G}_1$ plus B_w bits.
3. *Prover time.* $\Delta_m + 3\Delta_n - \Delta_l = 4296 + 5347 l_{\mathsf{w}} = 4296 + 21.6 B_w\ E_1$. As in the CRS, $1010 + 5.1 B_w\ E_2$.
4. *Verifier time.* Extra $\Delta_l = 3 l_{\mathsf{w}} \approx 0.012 B_w$ exponentiations, plus time to decode c_i (finding second point coordinate), which we ignore in our comparison.

3.5.0.2 Concrete Performance of Ext-Groth16

As we mentioned before, efficient black-box extraction from Ext-Groth16 is only possible if encrypted plaintext values are small enough, since the decryption algorithm needs to solve DLP for each ciphertext element. We assume that it is feasible to solve 43 bit DLP, which splits every 128 bits into 3 blocks.

Compared to Groth16, Ext-Groth16 has two types of overhead: on the first, structural layer, it has additional CRS elements (for the public key), ciphertext proof elements, prover exponentiations, and verifier pairings; on the second infrastructural layer, the circuit should be changed to assert that encrypted wire values fit into 43 bits, and then to convert from this representation to the desired one. We will denote the number of secret input wires by $l'_{\mathsf{w}} = \lceil B_w/43 \rceil$ to distinguish it from the number of wires l_{w} in the more efficienly-packed Int-Groth16.

Regarding infrastructure, since we compare to the circuit which already uses binary-decomposed witness, all the bits are checked to be binary, so the only real overhead is to pack them into field values and compare to the plaintext values that are plugged-in externally. Each comparison takes just one constraint, so we have extra $\Delta_n = l_{\mathsf{w}}$ constraints. We also have an additional input wires for ciphertexts, so $\Delta_m = \Delta_n + l_{\mathsf{w}} = 2 l_{\mathsf{w}}$. Although we connect the ciphertexts externally, formally they are not counted as public inputs, so $\Delta_l = 0$.

This gives, for the four parameters:

1. *CRS size.* $\Delta_m + 2\Delta_n + 2 l_{\mathsf{w}} = 6 l_{\mathsf{w}} \approx 0.14 B_w\ \mathbb{G}_1$. For second group elements,

Table 3.2: Overhead comparison of our constructions over plain Groth16. $\mathbb{G}^{\mathbb{J}}$ stands for bit-size of an encoded JubJub point, and $\mathbb{G}_i$ is the size of an encoded BLS12-381 point. Highlighted cells indicate efficiency improvement.

Construction	Int-Groth16	Ext-Groth16
CRS	$3286 + 16.4B_{\mathsf{w}}\ \mathbb{G}_1$ $1010 + 5.1B_{\mathsf{w}}\ \mathbb{G}_2$	$0.14B_{\mathsf{w}}\ \mathbb{G}_1$ $0.05B_{\mathsf{w}}\ \mathbb{G}_2$
Proof	$\left(\lceil \frac{B_{\mathsf{w}}}{248} \rceil + 1\right) \mathbb{G}^{\mathbb{J}} + B_{\mathsf{w}}$	$\left(\lceil \frac{B_{\mathsf{w}}}{43} \rceil + 2\right) \mathbb{G}_1$
Prover	$4296 + 21.6B_{\mathsf{w}}\ E_1$ $1010 + 5.1B_{\mathsf{w}}\ E_2$	$0.16B_{\mathsf{w}}\ E_1$ $0.05B_{\mathsf{w}}\ E_2$
Verifier	$3\lceil \frac{B_{\mathsf{w}}}{248} \rceil\ E_1$	$\left(\lceil \frac{B_{\mathsf{w}}}{43} \rceil + 2\right) P$

$\Delta_n + l_{\mathsf{w}} = 2l_{\mathsf{w}} = 0.05B_w\ \mathbb{G}_2$.

2. *Proof size.* We produce $l'_{\mathsf{w}} + 2$ extra ciphertext points in $\mathbb{G}_1$.

3. *Prover time.* For E_1, $\Delta_m + 3\Delta_n - \Delta_l + 2l_{\mathsf{w}} = 7l_{\mathsf{w}} = 0.16B_w\ E_1$. The overhead for E_2 is the same as of $\mathbb{G}_2$ in CRS size.

4. *Verifier time.* We need to compute $l'_{\mathsf{w}} + 2$ more pairings than in Groth16, and no additional exponentiations, since $\Delta_l = 0$.

3.5.0.3 Performance Comparison

Our estimates, summarized in Table 3.2, suggest that both constructions are quite efficient practically. Ext-Groth16 achieves better prover time and CRS size at the expense of slightly bigger proofs and verification time. CRS size and prover time of Ext-Groth16 incur a very small overhead, and are asymptotically *much* smaller than the same numbers for Int-Groth16, giving almost a $100 - 135\times$ performance gain. Hence, we focus our detailed analyses on the proof size and verifier time:

1. *Proof size.* Assuming that encoded BLS12-381 $\mathbb{G}_1$ takes 381 bits, and that JubJub point $\mathbb{G}^{\mathbb{J}}$ takes 256 bits, Int-Groth16 overhead is $\left(\lceil \frac{B_{\mathsf{w}}}{248} \rceil + 1\right)256 + B_{\mathsf{w}} \approx 2.03B_{\mathsf{w}} + 256$ bits, and for Ext-Groth16 it is $\left(\lceil \frac{B_{\mathsf{w}}}{43} \rceil + 2\right)381 \approx 8.86B_{\mathsf{w}} + 762$ bits. Asymptotically, Int-Groth16 proof size is $\times 4.4$ times smaller.

2. *Verifier time.* To compare the increase in exponentiations in Int-Groth16 with

the increase in pairings in Ext-Groth16, we use the estimation that micro benchmarks ([Atapoor and Baghery, 2019, Fig 2], also consistent with [Fauzi et al., 2017, Table 3] for BN-254) show pairings to be approximately $N = 35$ times slower than processing one element of a multi-exponentiation. Thus, the verification overhead of Int-Groth16 is small for practical witnesses, e.g. $1600 \cdot 3/248 \approx 20$ wires for encrypting 200 bytes, comparing to tens of thousands circuit constraints. And the overhead of Ext-Groth16 therefore is about $70\times$ more than for Int-Groth16, although for real-world witnesses it takes less than just a few tens milliseconds, and becomes immaterial for bigger public input sizes.

3.6 Open Questions

We prove two important theorems about [Groth, 2016] and [Lee et al., 2019] enabling the composable analysis of provable secure protocols. We conjecture that both our white-box and black-box results generalize to other SNARKs. In fact, we first showed white-box weak SE in a modification of [Groth and Maller, 2017] with the second equation removed. We decided to focus on Groth16 as the most important SNARK in this family to give a targeted proof and performance analysis. Besides improving performance, we expect weak SE and proof randomization to also have positive cryptographic applications that would be impossible with strong SE — just as for Groth-Sahai proofs [Groth and Sahai, 2008, Belenkiy et al., 2009].

Chapter 4

Secure Non-Malleable Ceremonies for SNARKs

This chapter is based on the work "Snarky Ceremonies", published in Asiacrypt 2021, and co-authored by Markulf Kohlweiss, Mary Maller, and Janno Siim.

Every application using Groth16 must run a separate trusted setup ceremony in order to ensure security, and even small errors in the setup could result a complete break of the system. Indeed, the paper of the original Zcash SNARK [Ben-Sasson et al., 2014c] contained a small typo which resulted in a bug that would allow an attacker to print unlimited funds in an undetectable manner [Gabizon, 2019]. Some would use this example as a reason to avoid any SNARK with a trusted setup ceremony at all costs. And yet Groth16 is not only still being used, but many protocols are being actively designed on top of it, potentially for the reasons listed above. Thus we believe that if this SNARK ceremony is going to be used anyway, it is important to put significant effort on simplifying its description and verifying its security.

The primary purpose of this chapter is to take a formal approach to proving the security of the Groth16 setup ceremony of Bowe, Gabizon, and Miers [Bowe et al., 2017b] that is currently commonly used in practice. The first prominent application of the protocol was the Zcash Sapling ceremony, but it was also run by many other projects, for example Aztec protocol, Filecoin, Semaphore, Loopring, Tornado Cash, Plumo Ceremony, and Hermez. Some of these ceremonies are based on the project called Perpetual Powers of Tau (PPoT), which implements the first phase of [Bowe et al., 2017b], that is not specialized to any circuit — this implies that the project planning to run a ceremony can fork off the PPoT, reducing its own setup cost. In other words, [Bowe et al., 2017b] is by far the most popular ceremony protocol used in practice; but it is also modified, specialized, and re-implemented by many independent projects. We simplify the original protocol, specifically we remove the need for a random beacon. Our security proofs equally apply to the version of the protocol with a beacon already used in practice.

A number of different works have analysed the setup security of zk-SNARKs. The works of [Ben-Sasson et al., 2015, Bowe et al., 2017a, Abdolmaleki et al., 2019] (see also [Aggelakis et al., 2020]) propose specialized multi-party computation protocols for SRS generation ceremonies. A common feature of these protocols is that they are secure if at least one of the parties is honest. However, these schemes are not robust in the sense that all parties must be fixed before the beginning of the protocol and be active throughout the whole execution. In other words if a single party goes offline between rounds then the protocol will not terminate.

Bowe, Gabizon, and Miers [Bowe et al., 2017b] showed that the latter problem could be solved if there is access to a random beacon — an oracle that periodically produces bitstrings of high entropy — which can be used to rerandomize the SRS after each protocol phase. Unfortunately, obtaining a secure random beacon is, by itself, an extremely challenging problem [Kiayias et al., 2017, Boneh et al., 2018, Han et al., 2020]. Secure solutions include unique threshold signatures [Hanke et al., 2018], which themselves require complex setup ceremonies as well as verifiable delay functions [Boneh et al., 2018, Pietrzak, 2019, Wesolowski, 2019] that require the design and use of specialized hardware. Practical realizations have instead opted for using a hash function applied to a recent blockchain block as a random beacon. This is not an ideal approach since the blockchain miners can bias the outcome.[1]

The work of Groth, Kohlweiss, Maller, Meiklejohn, and Miers [Groth et al., 2018] takes a different approach and directly constructs a SNARK where the SRS is updatable, that is, anyone can update the SRS and knowledge soundness and zero-

[1] It is desirable for a setup ceremony to avoid dependence on setups as much as possible—we spurn random beacons but embrace random oracles.

knowledge are preserved if at least one of the updaters was honest.[2] Subsequent updatable SNARKS like Sonic [Maller et al., 2019], Marlin [Chiesa et al., 2020], and PLONK [Gabizon et al., 2019] have improved the efficiency of updatable SNARKs, but they are still less efficient than for example [Groth, 2016]. Mirage [Kosba et al., 2020] modifies the original Groth16 by making the SRS universal, that is the SRS works for all relations up to some size bound. The latter work can be seen as complementary to the results of this work as it amplifies the benefits of a successfully conducted ceremony.

The key contributions of this chapter are as follows:

Designing a security framework. We formalize the notion of non-interactive zero-knowledge (NIZK) argument with a multi-round SRS ceremony protocol, which extends the framework of updatable NIZKs in [Maller et al., 2019]. Our definitions fix a syntax for ceremonies with Update and $\mathsf{VerifySRS}$ algorithms and take a game-based approach. This is less rigid than a multi-party computation definition (see for example [Abdolmaleki et al., 2019] for a UC-functionality). Our security notion says that an adversary cannot forge a SNARK proofs even if they can participate in the setup ceremony. We call such a SNARK ceremonial. This notion is more permissible for the setup ceremony than requiring simulatability and is therefore easier to achieve. In particular, using our definitions we do not require the use of a random beacon (as is needed in [Bowe et al., 2017b]) or additional setup assumptions ([Ben-Sasson et al., 2015] assumes a common random string and [Abdolmaleki et al., 2019] assumes a trusted commitment key), whereas it is not clear that those could be avoided in the MPC setting. Our definitions are applicable to SNARKs with a multiple round setup ceremony as long as they are ceremonial.

Proving security without a random beacon. We prove the security of the Groth16 SNARK with a setup ceremony of [Bowe et al., 2017b] in our new security framework. We intentionally try not change the original ceremony protocol too much so that our security proof would apply to protocols already used in practice. Security is proven with respect to algebraic adversaries [Fuchsbauer et al., 2018] in the random oracle model. We require a single party to

[2] Note that one can independently prove subversion ZK [Abdolmaleki et al., 2017, Fuchsbauer, 2018].

be honest in each phase of the protocol in order to guarantee that knowledge soundness and subversion zero-knowledge hold. Unlike [Bowe et al., 2017b], our security proof does not rely on the use of a random beacon. However, our security proof does apply to protocols that have been implemented using a (potentially insecure) random beacon because the beacon can just be treated as an additional malicious party. We see this as an important security validation of real-life protocols that cryptocurrencies depend on.

Revisiting the discrete logarithm argument. The original paper of [Bowe et al., 2017b] used a novel discrete logarithm argument Π_{dl} to prove knowledge of update contributions. They showed that the argument has knowledge soundness under the knowledge of exponent assumption in the random oracle model. While proving the security of the ceremony protocol, we observe that even stronger security properties are necessary. The discrete logarithm argument must be zero-knowledge and straight-line simulation extractable, i.e., knowledge sound in the presence of simulated proofs. Furthermore, simulation-extractability has to hold even if the adversary obtains group elements as an auxiliary input for which he does not know the discrete logarithm. We slightly modify the original argument to show that those stronger properties are satisfied if we use the algebraic group model with random oracles.

Thus, in this chapter we simplify the widely used protocol of [Bowe et al., 2017b] and puts it onto firmer security foundations.

4.1 Technical Overview

We provide a high-level technical overview of the results in this chapter.

Security framework. Our security framework assumes that the SRS is split into φ_{max} distinct components $\mathsf{srs} = (\mathsf{srs}_1, \ldots, \mathsf{srs}_{\varphi_{max}})$ and in each phase of the ceremony protocol one of the components gets finalized. We formalize this by enhancing the standard definition of NIZK with an Update and $\mathsf{VerifySRS}$ algorithms. Given srs and the phase number φ, the Update algorithm updates srs_φ and produces a proof ρ that the update was correct. The verification algorithm $\mathsf{VerifySRS}$ is used to check that srs and update proofs $\{\rho_i\}_i$ are valid.

We obtain the standard updatability model of [Maller et al., 2019] if $\varphi_{max} = 1$.

When modelling the Groth16 SNARK we set $\varphi_{max} = 2$. In that scenario, we split the SRS into a *universal* component $\mathsf{srs}_1 = \mathsf{srs}_u$ that is independent of the specific relation that we want to prove[3] and to a specialized component $\mathsf{srs}_2 = \mathsf{srs}_s$, which depends on a concrete relation $\mathcal{R}$. Both srs_u and srs_s are updatable; however, the initial srs_s has to be derived from srs_u and the relation $\mathcal{R}$. Thus, parties need first to update srs_u, and only after a sufficient number of updates can they start to update srs_s. The universal srs_u can potentially be reused for other relations.

In our definition of update knowledge soundness, we require that no adversary can convince an honest verifier of a statement unless either (1) they know a valid witness; (2) the SRS does not pass the setup ceremony verification $\mathsf{VerifySRS}$; or (3) one of the phases did not include *any* honest updates. Completeness and zero-knowledge hold for any SRS that passes the setup ceremony verification, even if there were no honest updates at all. The latter notions are known as subversion completeness and subversion zero-knowledge [Bellare et al., 2016].

Security proof of setup ceremony. We must prove subversion zero-knowledge and update knowledge-soundness. Subversion zero-knowledge follows from the previous work in [Abdolmaleki et al., 2017, Fuchsbauer, 2018], which already proved it for Groth16 under knowledge assumptions. The only key difference is that we can extract the simulation trapdoor with a discrete logarithm proof of knowledge argument Π_{dl} used in the ceremony protocol.

Our security proof of update knowledge-soundness uses a combination of the algebraic group model and the random oracle (RO) model. As was recently shown by Fuchsbauer, Plouviez, and Seurin [Fuchsbauer et al., 2020] the mixture of those two models can be used to prove powerful results (tight reductions of Schnorr-based schemes in their case) but it also introduces new technical challenges. Recall that the algebraic group model (AGM) is a relaxation of the generic group model proposed by Fuchsbauer, Kiltz, and Loss [Fuchsbauer et al., 2018]. They consider algebraic adversaries $\mathcal{A}_{alg}$ that obtain some group elements $G_1, \ldots, G_n$ during the execution of the protocol and whenever $\mathcal{A}_{alg}$ outputs a new group element E, it also has to output a linear representation $\vec{C} = (c_1, \ldots, c_n)$ such that $E = G_1^{c_1} G_2^{c_2} \ldots G_n^{c_n}$. Essentially, $\mathcal{A}_{alg}$ can only produce new group elements by

[3]Similarly to the universal updatability notions that share the same “independence”, e.g. [Maller et al., 2019], srs_u still formally depends on the maximum size of the circuit, which can nevertheless be made large enough to be practically universal.

applying group operations to previously known group elements. In contrast to the generic group model, the representation of group elements is visible to $\mathcal{A}_{alg}$, and thus security proofs in AGM are typically reductions to some group-assumptions (e.g. the discrete logarithm assumption).

Already the original AGM paper [Fuchsbauer et al., 2018] proved knowledge soundness of the Groth16 SNARK in the AGM model (assuming trusted SRS). They proved it under the q-discrete logarithm assumption, i.e., a discrete logarithm assumption where the challenge is $(G^z, G^{z^2}, \ldots, G^{z^q})$. The main idea for the reduction is that we can embed G^z in the SRS of the SNARK. Then when the algebraic adversary $\mathcal{A}_{alg}$ outputs a group-based proof π, all the proof elements are in the span of the SRS elements, and $\mathcal{A}_{alg}$ also outputs the respective algebraic representation. We can view the verification equation as a polynomial Q that depends on the SRS and π such that $Q(SRS, \pi) = 0$ when the verifier accepts. Moreover, since π and SRS depend on z, we can write $Q(SRS, \pi) = Q'(z)$. Roughly, the proof continues by looking at the formal polynomial $Q'(Z)$, where Z is a variable corresponding to z, and distinguishing two cases: (i) if $Q'(Z) = 0$, it is possible to argue based on the coefficient of Q' that the statement is valid and some of the coefficients are the witness, i.e., $\mathcal{A}_{alg}$ knows the witness, or (ii) if $Q'(Z) \neq 0$, then it is possible to efficiently find the root z of Q' and solve the discrete logarithm problem.

Our proof of update knowledge soundness follows a similar strategy, but it is much more challenging since the SRS can be biased, and the $\mathcal{A}_{alg}$ has access to all the intermediate values related to the updates. Furthermore, $\mathcal{A}_{alg}$ also has access to the random oracle, which is used by the discrete logarithm proof of knowledge Π_{dl}. Firstly, since the SRS of the Groth16 SNARK contains one trapdoor that is inverted (that is δ), we need to use a novel extended discrete logarithm assumption where the challenge value is $(\{G^{z^i}\}_{i=0}^{q_1}, \{H^{z^i}\}_{i=0}^{q_2}, r, s, G^{\frac{1}{rz+s}}, H^{\frac{1}{rz+s}})$ where G and H are generators of pairing groups and r, s, z are random integers. We prove that this new assumption is very closely related (equivalent under small change of parameters) to the q-discrete logarithm assumption. In the case with an honest SRS [Fuchsbauer et al., 2018] it was possible to argue that by multiplying all SRS elements by δ we get an equivalent argument which does not contain division, but it is harder to use the same reasoning when the adversary biases δ. The reduction still follows a similar high-level idea, but we need to introduce intermediate games that create a simplified environment before we can use the polynomial Q. For these games we rely on the zero-knowledge property and simulation extractability of Π_{dl}. Moreover, we have to consider that $\mathcal{A}_{alg}$ sees and adaptively affects intermediate states of the SRS on which the proof by π can depend on. Therefore the polynomial Q' takes a significantly more complicated form, but the simplified environment will reduce this complexity.

Revisiting the discrete logarithm argument. One of the key ingredients in the [Bowe et al., 2017b] ceremony is the discrete logarithm proof of knowledge Π_{dl}. Each updater uses this to prove that it knows its contribution to the SRS. The original [Bowe et al., 2017b] proved only knowledge soundness of Π_{dl}. While proving the security of the setup ceremony in our framework, we observe that much stronger properties are needed. Firstly, Π_{dl} needs to be zero-knowledge since it should not reveal the trapdoor contribution. Secondly, Π_{dl} should be knowledge sound, but in an environment where the adversary also sees simulated proofs and obtains group elements (SRS elements) for which it does not know the discrete logarithm. For this, we define a stronger notion simulation-extractability where the adversary can query oracle $\mathcal{O}_{\mathsf{se}}$ for simulated proofs and oracle $\mathcal{O}_{\mathsf{poly}}$ on polynomials $f(X_1, \ldots, X_n)$ that get evaluated at some random points $x_1, \ldots, x_n$ such that the adversary learns $G^{f(x_1,\ldots,x_n)}$ or $H^{f(x_1,\ldots,x_n)}$.

We show that proofs can be trivially simulated when the simulator has access to the internals of the random oracle and thus Π_{dl} is zero-knowledge. We once again use AGM, this time to prove simulation-extractability. Since in this proof we can embed the discrete logarithm challenge in the random oracle responses, we do not need different powers of the challenge and can instead rely on the standard discrete logarithm assumption. We also slightly simplify the original Π_{dl} and remove the dependence on the public transcript T_{Π} of the ceremony protocol, that is, the sequence of messages broadcasted by the parties so far. Namely, the original protocol hashes T_{Π} and the statement to obtain a challenge value. This turns out to be a redundant feature, and removing it makes Π_{dl} more modular.

Implementation and Optimization. Partners in a joint research project have developed a Rust implementation[4] of our Update and VerifySRS algorithms for Groth16 building on the arkworks library with various optimizations such as batch-

[4] `https://github.com/grnet/snarky`

ing and parallelization. This validates the correctness of our algorithms and intends to serve as an independent implementation to measure other solutions. We describe batched SRS update verification in 4.7.

4.2 Extended Discrete Logarithm Assumption

We presented the (q_1, q_2)-discrete logarithm assumption [Fuchsbauer et al., 2018] previously in Definition 2.3.4. In our main theorem it is more convenient to use a slight variation of it.

Definition 4.2.1 ((q_1, q_2)-**edlog**). *The* (q_1, q_2)-extended *discrete logarithm assumption holds for* BGen *if for any PPT* $\mathcal{A}$*:*

$$\Pr\left[\begin{array}{ll}\mathsf{bp} \xleftarrow{\$} \mathsf{BGen}(1^\lambda) & \\ z, r, s \xleftarrow{\$} \mathbb{Z}_p \textit{ s.t. } rz + s \neq 0 & : \quad z = z' \\ z' \xleftarrow{\$} \mathcal{A}(\mathsf{bp}, \{G^{z^i}\}_{i=1}^{q_1}, \{H^{z^i}\}_{i=1}^{q_2}, r, s, G^{\frac{1}{rz+s}}, H^{\frac{1}{rz+s}}) & \end{array}\right] = \mathsf{negl}(\lambda)$$

The assumption is an extension of (q_1, q_2)-**dlog**, where we additionally give $\mathcal{A}$ the challenge z in denominator (in both groups), blinded by s, r, which $\mathcal{A}$ is allowed to see. Later this helps to model fractional elements in Groth16's SRS. Notice that (q_1, q_2)-**edlog** trivially implies (q_1, q_2)-**dlog**, since $\mathcal{A}$ for the latter does not need to use the extra elements of the former. The opposite implication is also true (except for a slight difference in parameters) as we state in the following theorem.

Theorem 4.2.1. *If* $(q_1 + 1, q_2 + 1)$*-**dlog** assumption holds, then* (q_1, q_2)*-**edlog** assumption holds.*

Proof. Suppose that a PPT adversary $\mathcal{A}$ breaks (q_1, q_2)-**edlog** assumption with a probability ε. We will construct an adversary $\mathcal{B}$ that breaks $(q_1 + 1, q_2 + 1)$-**dlog** assumption with the same probability.

The adversary $\mathcal{B}$ gets as an input a challenge $(\mathsf{bp}, \{G^{z^i}\}_{i=1}^{q_1+1}, \{H^{z^i}\}_{i=1}^{q_2+1})$. Firstly, $\mathcal{B}$ samples $r, s \xleftarrow{\$} \mathbb{Z}_p$ and we implicitly define x such that $z = rx + s$; the value of x is unknown to $\mathcal{B}$. After this $\mathcal{B}$ constructs a pairing description bp^* which is exactly like bp but the generator G is changed to $\hat{G} := G^z$ and H to $\hat{H} = G^z$.[5] Now, let us observe that $\hat{G}^{\frac{1}{rx+s}} = \hat{G}^{1/z} = G$ and $\hat{G}^{x^i} = \hat{G}^{((z-s)/r)^i} = G^{z((z-s)/r)^i}$ for

[5] We implicitly assume that generators in bp are uniformly random. This might not always be the case in a real-life pairing library.

$i = 1, \ldots, q_1$ are all values that $\mathcal{B}$ either already knows or can compute from r, s and $\{G^{z^i}\}_{i=0}^{q_1+1}$. Considering that the same is true for $\mathbb{G}_2$ elements, $\mathcal{B}$ is able to run $\mathcal{A}$ on an input $(\mathsf{bp}, \{\hat{G}^{x^i}\}_{i=1}^{q_1}, \{\hat{H}^{x^i}\}_{i=1}^{q_2}, r, s, \hat{G}^{\frac{1}{rx+s}}, \hat{H}^{\frac{1}{rx+s}})$ and obtain some output x'. Finally, $\mathcal{B}$ returns $rx' + s$.

The adversary $\mathcal{A}$ will output $x' = x$ with a probability ε since the input to $\mathcal{A}$ is indistinguishable from an honest (q_1, q_2)-**edlog** challenge. If this happens, then $\mathcal{B}$ will succeed in computing z. Thus, $\mathcal{B}$ will break the (q_1+1, q_2+1)-**dlog** assumption with the same probability ε. Given the statement of our theorem, ε must be negligible and it follows that (q_1, q_2)-**edlog** assumption holds. $\square$

4.3 Ceremonial SNARKs

We present our definitions for NIZKs that are secure with respect to a setup ceremony. We discuss the new notions of update completeness and update soundness that apply to ceremonies that take place over many rounds. We also define subversion zero-knowledge which is adjusted to our ceremonial setting.

Compared to standard MPC definitions, our definition of (update) knowledge soundness is not simulation-based and the final SRS may not be uniformly random. We believe that the attempt to realise standard MPC definitions is what led prior works to make significant practical sacrifices e.g. random beacons or players that cannot go offline. This is because a rushing adversary that plays last can manipulate the bit-decomposition, for example to enforce that the first bit of the SRS is always 0. We here choose to offer an alternative protection: we allow that the final SRS is not distributed uniformly at random provided that the adversary does not gain any meaningful advantage when attacking the soundness of the SNARK. This is in essence an extension of updatability definitions [Groth et al., 2018] to ceremonies that require more than one round.

As usual, we consider NP-languages $\mathcal{L}$ and their corresponding relations $\mathcal{R} = \{(\mathsf{x}, \mathsf{w})\}$ where w is an NP-witness for the statement $\mathsf{x} \in \mathcal{L}$.

Definition 4.3.1. *A* NIZK with a ceremony protocol *for a relation $\mathcal{R}$ is a* NIZK *with the following additional algorithms:*

$\mathsf{UpdateSRS}(\varphi, \mathsf{srs}, \{\rho_i\}_i) \xrightarrow{\$} \mathsf{srs}'$**:** *A PPT SRS update algorithm that takes as input a phase number $\varphi \in \{1, \ldots, \varphi_{max}\}$, the current SRS* srs*, and proofs of pre-*

vious updates $\{\rho_i\}_i$*, and outputs a new SRS* srs' *and an update proof* ρ'*. It is expected that* $\mathsf{UpdateSRS}$ *itself forces a certain phase order, e.g. the sequential one.*

$\mathsf{VerifySRS}(\mathsf{srs}, \{\rho_i\}_i) \to 0/1$**:** *A DPT SRS verification algorithm that takes as an input a SRS* srs *and update proofs* $\{\rho_i\}_i$*, and outputs 0 or 1.*

The description of NIZK *also fixes a default* $\mathsf{srs}^{\mathsf{d}} = (\mathsf{srs}^{\mathsf{d}}_1, \ldots, \mathsf{srs}^{\mathsf{d}}_{\varphi_{max}})$*. The algorithms* $\mathsf{Prove}, \mathsf{Verify}, \mathsf{Sim}$ *are as before, and, semantically, can accept any* srs *. The* Setup *algorithm is also as before, but it is not used, since it is essentially replaced by the multi-round setup algorithm* $\mathsf{UpdateSRS}$*.*

As before, we implicitly assume $\mathsf{bp} \xleftarrow{\$} \mathsf{BGen}(1^\lambda)$ being run before all the algorithms, and bp being passed to them implicitly. We disallow subversion of bp in this work but in real life systems also this part of the setup needs scrutiny. This is arguable easier since usually bp is trapdoor free.

As usual, we require that a secure NIZK satisfies the following flavours of completeness, zero-knowledge, and knowledge soundness. All our definitions are in the (implicit) random oracle model, since our final SRS update protocol will be using RO-dependent proof of knowledge. Therefore, all the algorithms in this section have access to RO, if some sub-components of NIZK require it. In addition, we will require definitions

Completeness of NIZK additionally requires that $\mathsf{UpdateSRS}$ and Prove always satisfy verification.

Definition 4.3.2 (Perfect Completeness)**.** *A* NIZK *with a ceremony protocol for* $\mathcal{R}$ *is* perfectly complete *if for any adversary* $\mathcal{A}$*, it has the following properties:*

1. *Update completeness:*

$$\Pr\left[\begin{array}{l}(\varphi, \mathsf{srs}, \{\rho_i\}_i) \xleftarrow{\$} \mathcal{A}(1^\lambda), (\mathsf{srs}', \rho') \xleftarrow{\$} \mathsf{UpdateSRS}(\varphi, \mathsf{srs}, \{\rho_i\}_i) : \\ \mathsf{VerifySRS}(\mathsf{srs}, \{\rho_i\}_i) = 1 \land \mathsf{VerifySRS}(\mathsf{srs}', \{\rho_i\}_i \cup \{\rho'\}) = 0\end{array}\right] = 0.$$

2. *Prover completeness:*

$$\Pr\left[\begin{array}{l}(\mathsf{srs}, \{\rho_i\}_i, \mathsf{x}, \mathsf{w}) \xleftarrow{\$} \mathcal{A}(1^\lambda), \pi \xleftarrow{\$} \mathsf{Prove}(\mathsf{srs}, \mathsf{x}, \mathsf{w}) : \\ \mathsf{VerifySRS}(\mathsf{srs}, \{\rho_i\}_i) = 1 \land (\mathsf{x}, \mathsf{w}) \in \mathcal{R} \land \mathsf{Verify}(\mathsf{srs}, \mathsf{x}, \pi) \neq 1\end{array}\right] = 0.$$

We will need a strengthening of the standard definition of zero-knowledge (Definition 2.6.5), by modelling potential *subversion* of the SRS; our definition follows [Abdolmaleki et al., 2017]. Intuitively it says that an adversary that outputs a well-formed SRS knows the simulation trapdoor $\mathtt{td}$ and thus could simulate a proof himself even without the witness. Therefore, proofs do not reveal any additional information. On a more technical side, we divide the adversary into an efficient SRS subverter $\mathcal{Z}$ that generates the SRS (showing knowledge of $\mathtt{td}$ makes sense only for an efficient adversary) and into an unbounded distinguisher $\mathcal{A}$. We let $\mathcal{Z}$ send st to communicate with $\mathcal{A}$.

Definition 4.3.3 (Subversion Zero-Knowledge)**.** *A* NIZK *with a ceremony protocol for* $\mathcal{R}$ *is* subversion zero-knowledge (sub-ZK) *if for all PPT subverters* $\mathcal{Z}$*, there exists a PPT extractor* $\mathsf{Ext}_{\mathcal{Z}}$*, such that for all (unbounded)* $\mathcal{A}$*,* $|\varepsilon_0 - \varepsilon_1|$ *is negligible in* λ*, where*

$$\varepsilon_b := \Pr\left[\begin{array}{l} (\mathsf{srs}, \{\rho_i\}_i, \mathsf{st}) \xleftarrow{\$} \mathcal{Z}(1^\lambda) \\ \mathsf{td} \xleftarrow{\$} \mathsf{Ext}_{\mathcal{Z}}(\mathsf{view}_{\mathcal{Z}}) \end{array} : \begin{array}{l} \mathsf{VerifySRS}(\mathsf{srs}, \{\rho_i\}_i) = 1 \wedge \\ \mathcal{A}^{\mathcal{S}_{b,\mathsf{srs},\mathsf{td}}(\cdot)}(\mathsf{st}) = 1 \end{array}\right]$$

The simulation oracle $\mathcal{S}_{b,\mathsf{srs},\mathsf{td}}$ *is the first variant as defined in Fig. 2.2; it uses the* srs *returned by* $\mathcal{Z}$*.*

Bellare et al. [Bellare et al., 2016] showed that it is possible to achieve soundness and subversion zero-knowledge at the same time, but also that subversion soundness is incompatible with (even non-subversion) zero-knowledge. Updatable knowledge soundness from [Groth et al., 2018] can be seen as a relaxation of subversion soundness to overcome the impossibility result.

We generalize the notion of update knowledge soundness to multiple SRS generation phases. SRS is initially empty (or can be thought to be set to a default value $\mathsf{srs}^{\mathsf{d}}$). In each phase φ, the adversary has to fix a part of the SRS, denoted by srs_φ, in such a way building the final srs. The adversary can ask honest updates for his own proposal of srs^*_φ, however, it has to pass the verification $\mathsf{VerifySRS}$. The adversary can query honest updates using UPDATE query through a special oracle $\mathcal{O}_{srs}$, described in Fig. 4.1. Eventually, adversary can propose some srs^*_φ with update proofs Q^* to be finalized through FINALIZE query. The oracle does it if Q^* contains at least one honest update proof obtained from the oracle for the current phase. If that is the case, then srs_φ cannot be changed anymore and the phase $\varphi + 1$ starts. Once the whole SRS has been fixed, $\mathcal{A}$ outputs a statements x and a proof π. The adversary wins if $(\mathsf{srs}, \mathsf{x}, \pi)$ passes verification, but there is no PPT extractor $\mathsf{Ext}_{\mathcal{A}}$ that can extract a witness even when given the view of $\mathcal{A}$.

$\underline{\mathcal{O}_{srs}(\mathsf{intent}, \mathsf{srs}^*, Q^*)}$:

% Initially $Q_1 = \cdots = Q_{\varphi_{max}} \leftarrow \emptyset; \varphi \leftarrow 1$
% SRS already finalized for all phases:
if $\varphi > \varphi_{max}$ **then return** $\perp$
$\mathsf{srs}_{\mathsf{new}} \leftarrow (\mathsf{srs}_1, \ldots, \mathsf{srs}_{\varphi-1}, \mathsf{srs}^*_\varphi, \ldots, \mathsf{srs}^*_{\varphi_{max}})$
% Invalid SRS:
if $\mathsf{VerifySRS}(\mathsf{srs}_{\mathsf{new}}, Q^*) = 0$ **then return** $\perp$
if $\mathsf{intent} = \textsc{update}$ **then**
    $(\mathsf{srs}', \rho') \xleftarrow{\$} \mathsf{UpdateSRS}(\varphi, \mathsf{srs}_{\mathsf{new}}, Q^*)$;
    $Q_\varphi := Q_\varphi \cup \{\rho'\}$
    return (srs', ρ')
if $\mathsf{intent} = \textsc{finalize} \wedge Q_\varphi \cap Q^* \neq \emptyset$ **then**
    $\mathsf{srs}_\varphi := \mathsf{srs}^*_\varphi; \varphi := \varphi + 1$

Figure 4.1: SRS update oracle $\mathcal{O}_{srs}$ given to the adversary in Definition 4.3.4. UPDATE returns $\mathcal{A}$ an honest update for φ, and FINALIZE finalizes the current phase. Current phase φ and current SRS srs are shared with the KS challenger. $\{Q_{\varphi_i}\}_i$ is a local set of proofs for honest updates, one for each phase.

Definition 4.3.4 (Update Knowledge Soundness)**.** *An* NIZK *with a ceremony protocol for $\mathcal{R}$ is update knowledge-sound if for all PPT adversaries $\mathcal{A}$, there exists a PPT extractor* $\mathsf{Ext}_\mathcal{A}$ *such that* $\Pr[\mathcal{G}^{\mathcal{A},\mathsf{Ext}_\mathcal{A}}_{\mathsf{uks}}(1^\lambda) = 1]$ *is negligible in λ, where*

$$\mathcal{G}^{\mathcal{A},\mathsf{Ext}_\mathcal{A}}_{\mathsf{uks}}(1^\lambda) := \left[\begin{array}{l} (\mathsf{x}, \pi) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{srs}(\cdot)}(1^\lambda) \\ \textit{Obtain } (\mathsf{srs}, \varphi) \textit{ from } \mathcal{O}_{srs} \\ \mathsf{w} \xleftarrow{\$} \mathsf{Ext}_\mathcal{A}(\mathsf{view}_\mathcal{A}) \\ \textbf{return } \mathsf{Verify}(\mathsf{srs}, \mathsf{x}, \pi) = 1 \wedge \\ \qquad\qquad (\mathsf{x}, \mathsf{w}) \notin \mathcal{R} \wedge \varphi > \varphi_{max} \end{array}\right],$$

The SRS update oracle $\mathcal{O}_{srs}$ is described in Fig. 4.1.

If $\varphi_{max} = 1$, we obtain the standard notion of update knowledge soundness. In the rest of the chapter, we only consider the case where $\varphi_{max} = 2$. In particular, in the first phase we will generate a universal SRS $\mathsf{srs}_u = \mathsf{srs}_1$ that is independent of the relation and in the second phase we generate a specialized SRS $\mathsf{srs}_s = \mathsf{srs}_2$ that depends on the concrete relation. We leave it as an open question whether ceremony protocols with $\varphi_{max} > 2$ can provide any additional benefits. We also note that we do not model the possibility of the protocol running for several relations honestly simultaneously, although $\mathcal{A}$ can construct such SRS variants on its own.

It is important to explain the role of the default SRS in the definition. Our definition allows $\mathcal{A}$ to start its chain of SRS updates from any SRS, not just from the default one; the only condition is the presence of a single honest update in the chain. The default $\mathsf{srs}^{\mathsf{d}}$ is only used as a reference, for honest users. This has positive real-world consequences: since the chain is not required to be connected to any "starting point", clients only need to verify the suffix of Q^*, if they are confident it contains an honest update. In particular, clients that contribute to the SRS update can start from the corresponding proof of update.

We again note that when using the random oracle model in a sub-protocol, we assume that all of the above algorithms in our security model have access to RO.

4.4 Update Proofs of Knowledge

One of the primary ingredients in the setup ceremony is a proof of update knowledge whose purpose is to ensure that adversary knows which values they used for updating the SRS. In this section, we discuss the proof of knowledge given by Bowe et al [Bowe et al., 2017b]. Bowe et al. only proved this proof of knowledge secure under the presence of an adversary that can make random oracle queries. This definition is not sufficient to guarantee security (at least in our framework), because the adversary might be able to manipulate other users proofs or update elements in order to cheat. We therefore define a significantly stronger property that suffices for proving security of our update ceremony.

4.4.1 White-box Simulation-Extraction with Oracles

In this section, we provide definitions for the central ingredient of the ceremony protocol — the *update proof of knowledge* that ensures validity of each sequential SRS update. The proof of knowledge (PoK) protocol does not rely on reference string but employs a random oracle as a setup. Hence we will extend the standard NIZK definitions with $\mathsf{RO}_t(\cdot)$, defined in Fig. 2.1.

Since NIZK proof of knowledge is used in our ceremony protocol, we require it to satisfy a stronger security property than knowledge soundness or even simulation extraction. Instead of the standard white-box simulation-extractability (SE), we need a property that allows to compose the proof system more freely with other protocols while still allowing the adversary to extract. This is somewhat similar to idea of universal composability (UC, [Canetti, 2001]), but contrary to the standard UC, our extractor is still white-box. Another way would be to use an augmented UC model which allows white-box assumptions (see [Kerber et al., 2021a]). We follow the more minimal and commonly used game-based approach.

We model influence of other protocols by considering a polynomial oracle $\mathcal{O}_{\mathsf{poly}}$ in the SE game of the update PoK.

The adversary can query the oracle $\mathcal{O}_{\mathsf{poly}}$ on Laurent polynomials $f_i(Z_1, \ldots, Z_n)$ and it will output $G^{f_i(z_1,\ldots,z_n)}$ for $z_1, \ldots, z_n$ pre-sampled from a uniform distribution, and unknown to $\mathcal{A}$. We use Laurent polynomials since SRS elements, the access to which the oracle models, may have negative trapdoor powers.[6] By $\mathsf{deg}(f)$ we will denote the maximum absolute degree of its monomials, where by absolute degree of the monomial we mean the sum of all its degrees taken as absolute values. Formally, $\mathsf{deg}(c \cdot \prod_i Z_i^{a_i}) := \sum_i |a_i|$, and $\mathsf{deg}(f(Z_1, \ldots, Z_n)) = \mathsf{deg}(\sum_i M_i) := \max\{\mathsf{deg}(M_i)\}$, where M_i are monomials of f. For example, $\mathsf{deg}(3x^2\alpha\delta^{-2}+y) = 5$. This notion is used to limit the degree of input to $\mathcal{O}_{\mathsf{poly}}$ — we denote the corresponding degree $d(\lambda)$ (or d, interchangeably).

This empowered adversary still should not be able to output a proof of knowledge unless it knows a witness. Note that $\mathcal{O}_{\mathsf{poly}}$ is independent from the random oracle RO_t and cannot provide the adversary any information about the random oracle's responses. In general, $\mathcal{O}_{\mathsf{poly}}$ adds strictly more power to $\mathcal{A}$. The intention of introducing $\mathcal{O}_{\mathsf{poly}}$ is to account for the SRS of the Groth's SNARK later on.

In addition, our ceremony protocol for Groth16 requires NIZK to be straight-line simulation extractable, i.e., that extraction works without rewinding and is possible even when the adversary sees simulated proofs. Below, we define such a NIZK in the random oracle model.

As usual, let $\mathcal{L}$ be a language and $\mathcal{R}$ the corresponding relation. The argument NIZK for $\mathcal{R}$ in the *random oracle model* consists of the standard three algorithms with the following RO access: the prover $\mathsf{Prove}^{\mathsf{RO}(\cdot)}$, the verifier $\mathsf{Verify}^{\mathsf{RO}(\cdot)}$, and the simulator $\mathsf{Sim}^{\mathsf{RO}_1(\cdot)}$.

[6] See the description of Groth16 SRS, which has $1/\delta$ in some SRS elements.

$\underline{\mathcal{O}_{\mathsf{se}}(\mathsf{x})}$:	$\underline{\mathcal{O}^{\mathbb{G}_1}_{\mathsf{poly}}(f(Z_1,\dots,Z_{d(\lambda)}))}$:	$\underline{\mathcal{O}^{\mathbb{G}_2}_{\mathsf{poly}}(g(Z_1,\dots,Z_{d(\lambda)}))}$:
`% On init` $Q = \emptyset$	**if** $\deg(f) > d(\lambda)$ **then**	**if** $\deg(g) > d(\lambda)$ **then**
$\pi \xleftarrow{\$} \mathsf{Sim}^{\mathsf{RO}_1(\cdot)}(\mathsf{x})$	**return** $\perp$	**return** $\perp$
$Q := Q \cup \{(\mathsf{x}, \pi)\}$	**else return** $G^{f(z_1,\dots,z_d(\lambda))}$	**else return** $H^{g(z_1,\dots,z_d(\lambda))}$
return π		

Figure 4.2: Simulation-extraction oracle and two $d-$Poly oracles — for $\mathbb{G}_1$ and $\mathbb{G}_2$. All used in $\mathcal{G}_{\mathsf{sSE}}$.

We assume that NIZK in the ROM satisfies the following variations of the standard definitions, where the main difference is the new oracle algorithms, including RO, that we add to the NIZK algorithms and parties.

Definition 4.4.1. *An* NIZK *for* $\mathcal{R}$ *is* perfectly complete *in the random oracle model, if for any adversary* $\mathcal{A}$*,*

$$\Pr\left[\begin{array}{l}(\mathsf{x},\mathsf{w}) \xleftarrow{\$} \mathcal{A}^{\mathsf{RO}(\cdot)}(1^\lambda) \\ \pi \xleftarrow{\$} \mathsf{Prove}^{\mathsf{RO}(\cdot)}(\mathsf{x},\mathsf{w})\end{array} : \begin{array}{l}\mathsf{Verify}^{\mathsf{RO}(\cdot)}(\mathsf{x},\pi) = 1 \wedge \\ (\mathsf{x},\mathsf{w}) \notin \mathcal{R}\end{array}\right] = 0.$$

Definition 4.4.2. *A* NIZK *for* $\mathcal{R}$ *is* straight-line simulation extractable *in the (RO, $d-$Poly)-model, if for all PPT $\mathcal{A}$, there exists a PPT extractor* $\mathsf{Ext}_{\mathcal{A}}$ *such that* $Pr[\mathcal{G}^{\mathcal{A}}_{\mathsf{sSE}}(1^\lambda) = 1] = \mathsf{negl}(\lambda)$*, where*

$$\mathcal{G}^{\mathcal{A}}_{\mathsf{sSE}}(1^\lambda) := \left[\begin{array}{l} Q \leftarrow \emptyset; z_1,\dots,z_{d(\lambda)} \xleftarrow{\$} \mathbb{Z}_p \\ (\mathsf{x},\pi) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{se}},\mathsf{RO},\mathcal{O}^{\mathbb{G}_1}_{\mathsf{poly}},\mathcal{O}^{\mathbb{G}_2}_{\mathsf{poly}}}(1^\lambda) \\ \mathsf{w} \leftarrow \mathsf{Ext}_{\mathcal{A}}(\mathsf{view}_{\mathcal{A}}) \end{array} : \begin{array}{l}\mathsf{Verify}^{\mathsf{RO}(\cdot)}(\mathsf{x},\pi) = 1 \wedge \\ (\mathsf{x},\mathsf{w}) \notin \mathcal{R} \wedge (\mathsf{x},\pi) \notin Q\end{array}\right]$$

The oracles $\mathcal{O}_{\mathsf{se}}, \mathcal{O}^{\mathbb{G}_1}_{\mathsf{poly}}, \mathcal{O}^{\mathbb{G}_2}_{\mathsf{poly}}$ *are defined on Fig. 4.2.*

Roughly speaking, the adversary wins if it can output a verifying statement and proof for which it does not know a witness, such that this proof has not been obtained from a simulation oracle. There are also up to $d(\lambda)$ random variables chosen at the start such that the adversary can query an oracle for arbitrary polynomial evaluations with maximum degree $d(\lambda)$ of these values in the group. With respect to the relation of this definition to more standard one we note two things. First, our definition is white-box (since $\mathsf{Ext}_{\mathcal{A}}$ requires $\mathsf{view}_{\mathcal{A}}$), and strong (in the sense that proofs are not randomizable). Second, our notion implies strong-SE in the presence of RO, which is the special case of $\mathcal{G}_{\mathsf{sSE}}$ with $\mathcal{O}_{\mathsf{poly}}$ removed, and thus is very close to the standard non-RO strong-SE variant.

Definition 4.4.3. *A* NIZK *for the relation $\mathcal{R}$ is* perfectly zero-knowledge *in the random oracle model if for all PPT adversaries $\mathcal{A}$, $\varepsilon_0 = \varepsilon_1$, where*

$$\varepsilon_b := \Pr\left[\ \mathcal{A}^{\mathcal{S}_{b,\perp,\perp,\mathsf{RO}}(\cdot),\mathsf{RO}(\cdot)}(1^\lambda) = 1\ \right].$$

$\mathcal{S}_{b,\perp,\perp,\mathsf{RO}}$ is a variant of $\mathcal{S}_{b,\perp,\perp}$ on Fig. 2.2, that passes RO *to the subroutines: i.e. after the assertion it returns either* $\mathsf{Prove}^{\mathsf{RO}(\cdot)}(\mathsf{x}, \mathsf{w})$ *or* $\mathsf{Sim}^{\mathsf{RO}_1(\cdot)}(\mathsf{x})$*. We do not pass $\boldsymbol{\sigma}$ and* $\mathtt{td}$ *to $\mathcal{S}$ in this case since our RO-NIZK will not have a CRS.*

Note that Sim is allowed to have access to RO discrete logarithms.

4.4.2 On the Security of BGM Update Proofs

We now prove that the proof system of [Bowe et al., 2017b] satisfies this stronger property.

Bowe et al. [Bowe et al., 2017b] proved that the proof system is secure under a Knowledge-of-Exponent assumption. Their analysis does not capture the possibility that an attacker might use additional knowledge obtained from the ceremony to attack the update proof. Our analysis is more thorough and assumes this additional knowledge. This means that we cannot use a simple Knowledge-of-Exponent assumption. Instead we rely on the algebraic group model; the AGM is to date the weakest idealized model in which Groth16 has provable security and thus we do not see this as being a theoretical drawback. The proof of knowledge is for the discrete logarithm relation

$$\mathcal{R}_{\mathsf{dl}} := \left\{(\mathsf{x} = (m, G^{y_1}, H^{y_2}), \mathsf{w}) \mid y_1 = y_2 = \mathsf{w}\right\},$$

where m is an auxiliary input that was used in the original [Bowe et al., 2017b] proof of knowledge. The auxiliary input is redundant as we will see, but we still model it to have consistency with the original protocol. We recall that one of our goals is also to confirm the security of ceremony protocols already used in practice.

The protocol is given formally in Fig. 4.3. First the prover queries the random oracle on the instance x. The oracle returns a fresh random group element H^r. The prover returns $\pi = H^{rw}$. The verifier checks that the instance is well-formed ($y_1 = y_2$), and then checks that $\hat{e}(\pi, H) = \hat{e}(\mathsf{RO}(\mathsf{x}), H^{y_2})$ which ensures knowledge of y_2. Intuition for the last equation is that $\mathsf{RO}(\mathsf{x})$ acts as a fresh random challenge for x and the only way to compute $\pi = \mathsf{RO}(\mathsf{x})^{y_2}$ and H^{y_2} is by knowing y_2. The fact that in $\mathcal{R}_{\mathsf{dl}}$

$\underline{\mathsf{Prove}_{\mathsf{dl}}^{\mathsf{RO}(\cdot)}(\mathsf{x},\mathsf{w})}$:	$\underline{\mathsf{Sim}_{\mathsf{dl}}^{\mathsf{RO}_1(\cdot)}(\mathsf{x} = (\cdot, G^{y_1}, H^{y_2}))}$:
$G^r \leftarrow \mathsf{RO}(\mathsf{x})$	**assert** $\hat{e}(G^{y_1}, H) = (G, H^{y_2})$
return $G^{r\mathsf{w}}$	$r_{\mathsf{x}} \leftarrow \mathsf{RO}_1(\mathsf{x})$
	return $\pi := (G^{y_1})^{r_{\mathsf{x}}}$

$\underline{\mathsf{Verify}_{\mathsf{dl}}^{\mathsf{RO}(\cdot)}(\mathsf{x} = (\cdot, G^{y_1}, H^{y_2}), \pi)}$:

$G^r \leftarrow \mathsf{RO}(\mathsf{x})$

return $\hat{e}(G^{y_1}, H) = (G, H^{y_2}) \wedge \hat{e}(\pi, H) = \hat{e}(G^r, H^{y_2})$

Figure 4.3: The discrete logarithm proof of knowledge Π_{dl}.

every x with $y_1 = y_2$ belongs to $\mathcal{L}_{\mathsf{dl}}$ (the exponent w always exists) justifies that we will call the correspondent equation "well-formedness check"; subsequently, we will refer to the other check as "the main verification equation".

Here we have moderately simplified the description from [Bowe et al., 2017b]:

- We allow the message m to be unconstrained. Thus if one were to hash the public protocol view, as current implementations do, our security proof demonstrates that this approach is valid. However, we can also allow m to be anything, including the empty string.
- The original protocol has the proof element in $\mathbb{G}_2$. We switched it to $\mathbb{G}_1$ to have shorter proofs.
- Our protocol includes the pairing based equality check for y in G^y and H^y in the verifier rather than relying on this being externally done in the ceremony protocol. The value G^y is needed by the simulator.

We are now ready to state the security theorem for Π_{dl}.

Theorem 4.4.1. *The argument* $\Pi_{\mathsf{dl}} = (\mathsf{Prove}_{\mathsf{dl}}^{\mathsf{RO}(\cdot)}, \mathsf{Verify}_{\mathsf{dl}}^{\mathsf{RO}(\cdot)}, \mathsf{Sim}_{\mathsf{dl}}^{\mathsf{RO}_1(\cdot)})$ *is (i) complete, (ii) perfect zero-knowledge in the random oracle model, and (iii) straight-line SE in the (RO,*$d-$*Poly)-model against algebraic adversaries under the* $(1,0)$*-**dlog** assumption in* $\mathbb{G}_1$.

Completeness and perfect zero-knowledge follow directly from the construction of the prover, verifier, and simulator algorithms. The proof of straight-line simulation extractability, presented below, is more challenging, and its general idea is as follows. We consider security against algebraic adversaries $\mathcal{A}$. Both statement x

elements (G^y, H^y) and proof $\pi \in \mathbb{G}_1$ that $\mathcal{A}$ outputs are going to be in the span of elements that $\mathcal{A}$ queried from oracles. Coefficients of those spans can we white-box extracted from $\mathcal{A}$'s view $\mathsf{view}_{\mathcal{A}}$ due to $\mathcal{A}$ being algebraic. We construct an extractor $\mathsf{Ext}_{\mathcal{A}}$ that gets $\mathsf{view}_{\mathcal{A}}$ as an input and returns the coefficient k corresponding to the element $\mathsf{RO}(\mathsf{x}) = G^r$. Rest of the proof focuses on proving that k is the witness y. Roughly speaking, the idea is to construct a discrete logarithm adversary $\mathcal{C}$ that embeds (a randomized) discrete logarithm challenge G^c into each of the random oracle queries that $\mathcal{A}$ makes. We show that unless $k = y$, $\mathcal{C}$ is able to compute the discrete logarithm c from $\mathsf{view}_{\mathcal{A}}$ with an overwhelming probablity.

Proof. **(i) Completeness:** Holds straightforwardly.

(ii) Zero-Knowledge: It is easy to see that Π_{dl} is perfect zero-knowledge with respect to Sim in Fig. 4.3. When the simulator gets an input $\mathsf{x} = (m, G^w, H^w)$ (note that $\mathsf{x} \in \mathcal{L}$ by definition, so the exponent w is equal in G^w and H^w), it queries r for $G^r = \mathsf{RO}(\mathsf{x})$ using RO_1, and returns G^{wr}. No adversary can distinguish between honest and simulated proofs since they are equal.

(iii) Strong Simulation Extractability: Let $\mathcal{A}$ be an algebraic adversary playing $\mathcal{G}_{\mathsf{sSE}}$, and let us denote $\vec{z} = (z_1, \ldots, z_{d(\lambda)})$. As $\mathcal{A}$ is algebraic, at the end of $\mathcal{G}_{\mathsf{sSE}}$ it returns a statement and a proof (x, π) such that $\mathsf{x} = (m, G^{y'}, H^y)$ for some unknown variables y, y', and $\pi \in \mathbb{G}_1$. The fact that $y' = y$ immediately follows from the instance well-formedness pairing equation in Verify, and implies $\mathsf{x} \in \mathcal{L}$ (although does not affect the proof in any other way). For the elements H^y and π, $\mathcal{A}$ returns their representations $(\rho, b_1, \ldots, b_{q_2})$ and $(\alpha, a_1, \ldots, a_{q_1}, k_1, \ldots k_{q_3}, p_1, \ldots p_{q_4})$ that satisfy, correspondingly,

$$H^y = H^{\rho + b_1 g_1(\vec{z}) + \cdots + b_{q_2} g_{q_2}(\vec{z})} \tag{4.1}$$

and

$$\pi = G^{\alpha + a_1 f_1(\vec{z}) + \cdots + a_{q_1} f_{q_1}(\vec{z})} \cdot \prod_{j=1}^{q_3} K_j^{k_j} \cdot \prod_{j=1}^{q_4} P_j^{p_j} \tag{4.2}$$

In the former, ρ stands for the power of H, and b_i are linear coefficients of the polynomial evaluations returned by $\mathcal{O}^{\mathbb{G}_2}_{\mathsf{poly}}$. Similarly, for π, the representation is split into powers of the generator G, and coefficients of $\mathcal{O}^{\mathbb{G}_1}_{\mathsf{poly}}$, but it also accounts for the answers to hash queries $K_j, 1 \le j \le q_3$, and for the proof elements $P_j, 1 \le j \le q_4$, returned by the simulation oracle.

Let $S \subset [1, \dots, q_3]$, replacing $[1, \dots, q_4]$, be a set of indices denoting queries made *by the simulator* to the random oracle; $|S| = q_4$, and we know $q_3 \geq q_4$ since every simulation query produces one RO query. Also in the following, we let r^* and r_j be such that RO(x) $= G^{r^*}$ and RO(x$_j$) $= G^{r_j}$ for $1 \leq j \leq q_3$. RO responses $\{G^{r_j}\}$, corresponding to the second set of elements $\{r_j\}$, exist in view$_{\mathcal{A}}$ (in the list of queries and responses to RO), since these values were generated by RO during the game. On the other hand, G^{r^*} may not exist in view$_{\mathcal{A}}$, but then the probability that π verifies is negligible, as fresh G^{r^*} will be generated during the verification. Therefore, since we assume that $\mathcal{A}$ wins $\mathcal{G}_{\mathsf{sSE}}$, $r^* \in \{r_j\}_{j \in [1,q_3] \setminus S}$. S is excluded from the set of indices, since $\mathcal{A}$ also must not query Sim on x.

Thus, $K_j^{k_j}$ in the previously mentioned linear representations is just $G^{r_j k_j}$. In order to give algebraic representation of the simulated proofs P_j we must consider algebraic representations of inputs to Sim first. Because the simulated proof is constructed as $(G^{y_1})^r$ where G^{y_1} is an input provided by $\mathcal{A}$, G^{y_1} is the only input element that must be viewed algebraically. Notice that since we have a $\hat{e}(G^{y_1}, H) = \hat{e}(G, H^{y_2})$ check in the simulator too, the algebraic representation of y_1 must be consistent with the one of y_2, i.e. whatever $\mathcal{A}$ uses to construct G^{y_1} it must also have in $\mathbb{G}_2$ to construct H^{y_2}. In particular, this means that $\mathcal{A}$ cannot include (previous) direct RO responses and (previous) Sim responses into G^{y_1}, since these both contain r_i which $\mathcal{A}$ does not have in $\mathbb{G}_2$. Therefore, $P_j = G^{r_j y_j}$ is algebraically represented as $P_j = G^{r_j(\hat{\rho}_j + \sum_{i=1}^{q_1} \hat{a}_{j,i} f_i(\vec{z}))}$. Note that if $\mathcal{A}$ has not yet performed all the q_1 queries to $\mathcal{O}_{\mathsf{poly}}^{\mathbb{G}_1}$, then we can assume that $\hat{a}_{j,i} = 0$ for the subsequent queries. Finally, it is important to emphasize that $f_i(\vec{z})$ do not have any further algebraic decomposition: $\mathcal{A}$ specifies these polynomials to $\mathcal{O}_{\mathsf{poly}}$ in terms of $f_{i,j} \in \mathbb{Z}_p$, so these elements are just assumed to be standard public variables in our reasoning.

Because of the verification equation we have RO(x)$^y = \pi$. We thus have the two equations describing challenge values G^y and π, corresponding to Equations 4.1 and 4.2, in the exponent form: $y = \rho + \sum_{i=1}^{q_2} b_i g_i(\vec{z})$ and

$$yr^* = \alpha + \sum_{j=1}^{q_1} a_j f_j(\vec{z}) + \sum_{j=1}^{q_3} k_j r_j + \sum_{j \in S} p_j r_j (\hat{\rho}_j + \sum_{i=1}^{q_1} \hat{a}_{j,i} f_i(\vec{z}))$$

where in the second we used algebraic representations of K_j and P_j.

Let Ext$_{\mathcal{A}}$ be the SE extractor with the following logic. First it obtains the set S of (indices of) simulated queries; this can be deduced from the interaction pattern with the oracles, which is a part of $\mathsf{view}_{\mathcal{A}}$. Then, in the adversarial view $\mathsf{view}_{\mathcal{A}}$ find such an RO query index $j \in [1, q_3] \setminus S$ that RO input is equal to x; if successful, return k_j, and otherwise fail, returning 0. The intuition behind the extractor is the following. Since honest proofs are $\mathsf{RO}(\mathsf{x})^w$ for *direct* RO queries $\mathcal{A}$ makes, we expect k_j to be the witness. If $j \in S$, $\mathcal{A}$ re-used the *simulation* query and does not win.[7] When $G^{r^*} \neq G^{r_j}$ (which implies $r^* \neq r_j$) for all $j \in [1, q_3] \setminus S$, $\mathcal{A}$ did not query RO, and thus cannot win except with negligible probability.

We emphasize two limitations that any $\mathsf{Ext}_{\mathcal{A}}$ has, which shape the algorithm that we have just presented. First, the extractor does not have access to exponent values r_i themselves, since they are embedded inside RO, but $\mathsf{Ext}_{\mathcal{A}}$ only sees *interaction* with the oracle via $\mathsf{view}_{\mathcal{A}}$; therefore, it works only with G^{r_i} and S. Second, $\mathsf{Ext}_{\mathcal{A}}$ cannot compute exponent y right away merely from the algebraic representation of H^y passed as a part of x. Even though the coefficients $(\rho, b_1, \ldots, b_{q_2})$ are available to $\mathsf{Ext}_{\mathcal{A}}$ in the SE game, it does not have access to the trapdoor $\vec{z}$ of $\mathcal{O}^{\mathbb{G}_1}_{\mathsf{poly}}$, which is intended to model the external honest SRS setup procedure.

To prove that $\mathsf{Ext}_{\mathcal{A}}$ is a valid SE extractor for $\mathcal{A}$, we shall describe the behaviour of an adversary $\mathcal{C}$ that succeeds against the discrete logarithm assumption whenever $\mathsf{Ext}_{\mathcal{A}}$ fails to return a valid witness for $\mathcal{A}$. Thus if $\mathcal{A}$ has non-negligible advantage in the SE game with respect to $\mathsf{Ext}_{\mathcal{A}}$, then $\mathcal{C}$ also succeeds with non-negligible probability. As usual, $\mathcal{C}$ will simulate the SE game to $\mathcal{A}$, and it will succeed when $\mathcal{A}$ succeeds in the simulated game.

The adversary $\mathcal{C}$ takes as input a challenge C and aims to return c such that $C = G^c$. To begin it samples $(z_1, \ldots, z_d) \xleftarrow{\$} \mathbb{Z}_p$ and then runs $\mathcal{A}$ on input bp. $\mathcal{C}$ simulates the oracles for $\mathcal{A}$ in the following way:

- When $\mathcal{A}$ queries $\mathcal{O}^{\mathbb{G}}_{\mathsf{poly}}$ with $\mathbb{G} = \mathbb{G}_1$ on $f(\vec{Z})$, $\mathcal{C}$ returns $G^{f(z_1,\ldots,z_d)}$; on $\mathbb{G} = \mathbb{G}_2$ and $g(\vec{Z})$ it returns $H^{g(z_1,\ldots,z_d)}$.
- When $\mathcal{A}$ queries RO on x_j then $\mathcal{C}$ checks whether $(\mathsf{x}_j, G^{ct_j+s_j}, (t_j, s_j)) \in Q_{\mathsf{RO}}$ and if yes returns $G^{ct_j+s_j}$.

  Otherwise $\mathcal{C}$ samples $t_j, s_j \xleftarrow{\$} \mathbb{Z}_p$, adds $(\mathsf{x}_j, G^{ct_j+s_j}, (t_j, s_j))$ to Q_{RO} and returns $G^{ct_j+s_j}$, thus *embedding the challenge* into the response.
- When $\mathcal{A}$ queries simulation oracle $\mathcal{O}_{\mathsf{se}}$ on $\mathsf{x}_j = (m_j, G^{y_j}, H^{y_j})$ then its al-

[7] We exclude RO collision as they only happen with negligible probability.

gebraic extractor outputs representations $(\hat{\rho}_j, \hat{a}_{j,1}, \ldots, \hat{a}_{j,q_1})$ such that $y_j = \hat{\rho}_j + \sum_{i=1}^{q_1} \hat{a}_{j,i} f_i(\vec{z})$ for $f_i(Z)$ being ith query to $\mathcal{O}^{\mathbb{G}_1}_{\mathsf{poly}}$ (the representation is, as previously for y, due to the well-formedness verification equation). In this case $\mathcal{C}$ obtains $K_j = \mathsf{RO}(\mathsf{x}_j)$ and returns $K_j^{\hat{\rho}_j + \sum_{i=1}^{q_1} \hat{a}_{j,i} f_i(\vec{z})}$ (notice that $\mathcal{C}$, unlike $\mathsf{Ext}_{\mathcal{A}}$, knows $\vec{z}$ but not $ct_j + s_j$, thus the simulation strategy is different from Sim).

When, finally, $\mathcal{A}$ returns $(\mathsf{x} = (\cdot, \cdot, H^y), \pi)$, $\mathcal{C}$ obtains $(\rho, \{a_j\}, \{b_j\}, \{k_j\}, \{p_j\})$ such that $y = (\rho + \sum_{j=1}^{q_2} b_j g_j(\vec{z}))$ and

$$y(ct^* + s^*) = \alpha + \sum_{j=1}^{q_1} a_j f_j(\vec{z}) + \sum_{j=1}^{q_3} k_j(ct_j + s_j) + \sum_{j \in S} p_j(ct_j + s_j)(\hat{\rho}_j + \sum_{i=1}^{q_1} \hat{a}_{j,i} f_i(\vec{z})).$$

This is the same representation as $\mathsf{Ext}_{\mathcal{A}}$ obtains, with the previous randomness now depending on the challenge c. Additionally we assume that $G^{r^*} = \mathsf{RO}(\mathsf{x})$ is of form $r^* = ct^* + s^*$ and that it is determined by the j^*th RO query of $\mathcal{A}$ (thus t^* and s^* are, too). This is, again, because $\mathcal{A}$ cannot succeed without querying x to RO during the game. Substituting y from the first equation into the second equation gives us a polynomial equation in c which it is possible to solve. Note that c enters the last equation in three different places. Now $\mathcal{C}$ sets

$$\xi = \left((\rho + \sum_{j=1}^{q_2} b_j g_j(\vec{z}))t^* - \sum_{j=1}^{q_3} k_j t_j - \sum_{j \in S} p_j t_j (\hat{\rho}_j + \sum_{i=1}^{q_1} \hat{a}_{j,i} f_i(\vec{z})) \right)$$

and returns

$$c = \xi^{-1} \left(\alpha + \sum_{j=1}^{q_1} a_j f_j(\vec{z}) + \sum_{j=1}^{q_3} k_j s_j + \sum_{j \in S} p_j s_j (\hat{\rho}_j + \sum_{i=1}^{q_1} \hat{a}_{j,i} f_i(\vec{z})) - s^*(\rho + \sum_{j=1}^{q_2} b_j g_j(\vec{z})) \right).$$

Observe that $\mathcal{C}$ succeeds (returns c) whenever ξ^{-1} exists i.e. whenever $\xi \neq 0$. Recall that since $\mathcal{A}$ succeeds, $t^* \neq t_j$ for any $j \in S$. Consider the coefficients of ξ that include t^* in the monomials:

$$\xi = t^* \left[(\rho + \sum_{j=1}^{q_2} b_j g_j(\vec{z})) - k_{j^*} \right] + \ldots$$

If $\xi = 0$ then this expression is equal to zero with overwhelming probability bounded below by $1 - \frac{1}{p}$ by the Schwartz-Zippel Lemma. This is because the adversary learns no information about the secret values, including t_j, due to the presence of the s_j randomizers, thus ξ must be zero as a polynomial in all t_j, and in particular in $t_{j^*} = t^*$. And for a zero polynomial, for all its monomial the related coefficients are zero. However, if $(\rho + \sum_{j=1}^{q_2} b_j g_j(\vec{z})) - k_{j^*} = 0$, then $\mathsf{Ext}_\mathcal{A}$ succeeds (since then $k_{j^*} = y$), which we assumed to be false. Therefore, $\xi \neq 0$ and $\mathcal{C}$ succeeds.

Finally observe that if r^* is not determined by any adversarial query ($\mathcal{A}$ passing x that was not sent to RO before), then $(\rho + \sum_{j=1}^{q_2} b_j g_j(\vec{z})) = 0$ except with negligible probability by the same Schwartz-Zippel argument since $\mathcal{A}$ does not see RO exponents. Therefore $y = 0$ is the only possible valid witness, so $\mathsf{Ext}_\mathcal{A}$ succeeds. $\square$

4.5 Groth16 is Ceremonial

We show that Groth16 is ceremonial for a setup ceremony similar to the one proposed in [Bowe et al., 2017b]. In this section, we start by giving an intuitive overview of the [Bowe et al., 2017b] ceremony protocol. After that, we recall the Groth16 argument and carefully model the ceremony protocol in our security framework.

4.5.1 Ceremony Overview

We briefly remind the main idea of the [Bowe et al., 2017b] ceremony protocol.

- The SRS contains elements of the form e.g. $(A_1, \ldots, A_n, T) = (G^x, G^{x^2}, \ldots, G^{x^n}, G^{\delta p(x)})$ where $p(X)$ is a public polynomial known to all parties, and x and δ are secret trapdoors.[8]
- Parties initialize the SRS to $(A_1, \ldots, A_n, T) = (G, \ldots, G, G)$.
- In the first phase any party can update $(A_1, \ldots, A_n)$ by picking a random $x' \in \mathbb{Z}_p$ and computing $(A_1^{x'}, \ldots, A_n^{(x')^n})$. They must provide a proof of knowledge of x'.
- The value T is publicly updated to $G^{p(x)}$ given $A_1, \ldots, A_n$.
- In the second phase any party can update T by picking a random $\delta' \in \mathbb{Z}_p$ and computing $T^{\delta'}$. They must provide a proof of knowledge of δ'.

In order to prove knowledge of x' they assume access to a random oracle RO : $\{0,1\}^* \to \mathbb{G}_2$ and proceed as follows:

[8] The polynomial $p(X)$ is introduced only in the scope of this example, and is not related to QAP.

- The prover computes $R \leftarrow \mathsf{RO}(\mathsf{T}_\Pi \| G^x)$ as a challenge where T_Π is the public transcript of the protocol.
- Then prover outputs $\pi \leftarrow R^x$ as a proof which can be verified by recomputing R and checking that $\hat{e}(G, \pi) = \hat{e}(G^x, R)$. The original protocol is knowledge sound under (a variation of) the knowledge of exponent assumption, which states that if given a challenge R, the adversary outputs (G^x, R^x), then the adversary knows x.

Our protocol differs from the [Bowe et al., 2017b] in a few aspects related to both performance and security. Additionally to the RO switch to $\mathbb{G}_1$ and optionality of including T_Π in evaluation of RO, which we described in Section 4.4, we remove the update with the random beacon in the end of each phase. That means that SRS can be slightly biased, but we prove that it is not sufficient to break the argument's security. We consider this to be the biggest contribution of this chapter since obtaining random beacons is a significant challenge both in theory and practice. Our approach completely side-steps this issue by directly proving the protocol without relying on the random beacon model.

4.5.2 Formal Description

We present the variant of Groth's SNARK [Groth, 2016] from [Bowe et al., 2017b] and adjust the ceremony protocol to our security framework by defining $\mathsf{UpdateSRS}$ and $\mathsf{VerifySRS}$ algorithms which follow the intuition of the previous section.

We already described Groth16 NIZK [Groth, 2016] in Section 2.6.3. Bowe et al. [Bowe et al., 2017b] modified original argument's SRS to make it consistent with their distributed SRS generation protocol. The full description of the latter argument, variant of Groth16, is in Fig. 4.4.

We adjust the SRS in Fig. 4.4 to our model with a ceremony protocols: the default SRS, update algorithm, and a SRS specialization algorithm are described in Fig. 4.5. [9] We obtain the default SRS from the trapdoor $\mathsf{td} = (1, 1, 1, 1)$. The algorithm $\mathsf{UpdateSRS}$ samples new trapdoors and includes them in the previous SRS by exponentiation as was described in Section 4.5.1. For example, to update

[9] Our Groth16 SRS follows [Bowe et al., 2017b] and not the original [Groth, 2016]. It additionally contains $\{H^{x^i}\}_{i=n-2}^{2n-2}$, $\{H^{\alpha x^i}\}_{i=1}^{n-1}$, and $\{H^{\beta x^i}\}_{i=1}^{n-1}$. This simplifies our presentation, but also strengthens the security result as it shows that contrary to what happened with a different extended SRS of Zcash [Gabizon, 2019] adding these elements does not break soundness.

$\underline{\mathsf{Setup}(\mathcal{R}_{\mathsf{QAP}})\text{:}}$

1. Sample $\mathsf{td} = (\alpha, \beta, \delta, x) \xleftarrow{\$} (\mathbb{Z}_p^*)^4$
2. $\mathsf{srs}_u \leftarrow \left(\{G^{x^i}, H^{x^i}\}_{i=0}^{2n-2}, \{G^{\alpha x^i}, G^{\beta x^i}, H^{\alpha x^i}, H^{\beta x^i}\}_{i=0}^{n-1}\right)$
3. $\mathsf{srs}_s \leftarrow \left(G^{\delta}, H^{\delta}, \{G^{\frac{\beta u_i(x)+\alpha v_i(x)+w_i(x)}{\delta}}\}_{i=\ell+1}^{m}, \{G^{\frac{x^i t(x)}{\delta}}\}_{i=0}^{n-2}\right)$
4. **return** $(\mathsf{srs} := (\mathsf{srs}_u, \mathsf{srs}_s), \mathsf{td})$

$\underline{\mathsf{Prove}(\mathcal{R}_{\mathsf{QAP}}, \mathsf{srs}, \{\mathsf{a}_i\}_{i=0}^{m} := (\mathsf{x} \parallel \mathsf{w}))\text{:}}$

1. Sample $r, s \xleftarrow{\$} \mathbb{Z}_p$
2. $A \leftarrow \alpha + \sum_{i=0}^{m} \mathsf{a}_i u_i(x) + r\delta$
3. $B \leftarrow \beta + \sum_{i=0}^{m} \mathsf{a}_i v_i(x) + s\delta$
4. $C \leftarrow \frac{\sum_{i=\ell+1}^{m} \mathsf{a}_i(\beta u_i(x)+\alpha v_i(x)+w_i(x))+h(x)t(x)}{\delta} + As + Br - rs\delta$
5. **return** $\pi := (G^A, H^B, G^C)$

$\underline{\mathsf{Verify}(\mathcal{R}_{\mathsf{QAP}}, \mathsf{srs}, \{\mathsf{a}_i\}_{i=1}^{\ell} := x, \pi)\text{:}}$

1. Parse π as (G^A, H^B, G^C)
2. **return** $\hat{e}(G^A, H^B)$
3. $\quad = \hat{e}(G^{\alpha}, H^{\beta}) \cdot \hat{e}(\prod_{i=0}^{\ell} G^{\mathsf{a}_i(\beta u_i(x)+\alpha v_i(x)+w_i(x))}, H) \cdot \hat{e}(G^C, H^{\delta})$

$\underline{\mathsf{Sim}(\mathcal{R}_{\mathsf{QAP}}, \mathsf{srs}, \mathsf{td}, \{\mathsf{a}_i\}_{i=1}^{\ell})\text{:}}$

1. $A, B \xleftarrow{\$} \mathbb{Z}_p$,
2. $C \leftarrow \frac{AB - \alpha\beta - \left(\sum_{i=0}^{\ell} \mathsf{a}_i(\beta u_i(x)+\alpha v_i(x)+w_i(x))\right)}{\delta}$
3. **return** (G^A, H^B, G^C)

Figure 4.4: The variant of Groth16 zk-SNARK used in Chapter 4. Compare with the naive variant in Fig. 2.3 used in Chapter 3.

G^ι, where ι is some trapdoor, the updater will sample ι' and computes $(G^\iota)^{\iota'}$. Depending on the phase number $\varphi \in \{1, 2\}$, the algorithm will either update srs_u or srs_s. When updating srs_u, we also derive a consistent srs_s using the $\mathsf{Specialize}$ algorithm[10] which essentially computes srs_s with $\delta = 1$. This fixes a sequential phase update scenario, since updating srs_u after srs_s overwrites the latter.

Each update is additionally accompanied with an update proof ρ, which allows us to verify update correctness. For each trapdoor update ι', ρ contains $G^{\iota\iota'}$ (the element of the new SRS), $G^{\iota'}$, $H^{\iota'}$, and a NIZK proof of knowledge $\pi_{\iota'}$ for ι'. Since G^ι is part of the previous update proof, we can use pairings to assert well-formedness of $G^{\iota\iota'}$, $G^{\iota'}$, and $H^{\iota'}$. The first element of the update proof duplicates the element of the new SRS, but since we do not store every updated SRS but only update proofs, we must keep these elements.

Finally, we have a SRS verification algorithm $\mathsf{VerifySRS}$ in Fig. 4.6, that takes as an input srs and a set of update proofs Q, and then (i) uses pairing-equations to verify that srs is well-formed respect to some trapdoors, (ii) checks that each update proof $\rho \in Q$ contains a valid NIZK proof of discrete logarithm, and (iii) uses pairing-equations to verify that update proofs in Q are consistent with srs. In Section 4.7, we show how to make $\mathsf{VerifySRS}$ more efficient by using batching techniques. This will allow to substitute most of pairings in $\mathsf{VerifySRS}$ with significantly cheaper small-exponent multi-exponentiations.

4.6 Security

We prove the security of Groth16 from Section 4.5 in our NIZK with a ceremony framework of Section 4.3.

Theorem 4.6.1 (Completeness)**.** *Groth16 variant in Fig. 4.4 has perfect completeness, i.e., it has update completeness and prover completeness.*

Proof. Let us first make a general observation that if some bitstring $s = (\mathsf{srs}, \{\rho_i\}_i)$ satisfies $\mathsf{VerifySRS}(s) = 1$, then there exists a unique $\alpha, \beta, x, \delta \in \mathbb{Z}_p^*$ that define a well-formed srs. See Lemma 4.8.1, Section 4.8.1.

[10] This generality simplifies our model. In practice srs_s can be derived using $\mathsf{Specialize}$ only once just before starting phase 2.

Default SRS: Run Setup in Fig. 4.4 with $\mathsf{td} = (1,1,1,1)$ to obtain $\mathsf{srs}^{\mathsf{d}}$.

$\mathsf{UpdateSRS}(\mathcal{R}_{\mathsf{QAP}}, \varphi \in \{1,2\}, (\mathsf{srs} = (\mathsf{srs}_u, \mathsf{srs}_s), Q))$:

if $\varphi = 1$ **then**

  Parse $\mathsf{srs}_u = \left(\{G_{x:i}, H_{x:i}\}_{i=0}^{2n-2}, \{G_{\alpha x:i}, G_{\beta x:i}, H_{\alpha x:i}, H_{\beta x:i}\}_{i=0}^{n-1}\right)$

  Sample $\alpha', \beta', x' \xleftarrow{\$} \mathbb{Z}_p^*$

  For $\iota \in \{\alpha, \beta, x\}$: $\pi_{\iota'} \leftarrow \mathsf{Prove}_{\mathsf{dl}}^{\mathsf{RO}(\cdot)}(G^{\iota'}, H^{\iota'}, \iota')$

  $\rho_{\alpha'} \leftarrow (G_{\alpha x:0}^{\alpha'}, G^{\alpha'}, H^{\alpha'}, \pi_{\alpha'})$

  $\rho_{\beta'} \leftarrow (G_{\beta x:0}^{\beta'}, G^{\beta'}, H^{\beta'}, \pi_{\beta'})$

  $\rho_{x'} \leftarrow (G_{x:1}^{x'}, G^{x'}, H^{x'}, \pi_{x'})$

  $\rho \leftarrow (\rho_{\alpha'}, \rho_{\beta'}, \rho_{x'})$

  $\mathsf{srs}'_u \leftarrow \left(\{G_{x:i}^{(x')^i}, H_{x:i}^{(x')^i}\}_{i=0}^{2n-2}, \{G_{\alpha x:i}^{\alpha'(x')^i}, G_{\beta x:i}^{\beta'(x')^i}, H_{\alpha x:i}^{\alpha'(x')^i}, H_{\beta x:i}^{\beta'(x')^i}\}_{i=0}^{n-1}\right)$

  $\mathsf{srs}'_s \leftarrow \mathsf{Specialize}(\mathsf{QAP}, \mathsf{srs}'_u)$

  return $((\mathsf{srs}'_u, \mathsf{srs}'_s), \rho)$

else if $\varphi = 2$ **then**

  Parse $\mathsf{srs}_s = \left(G_\delta, H_\delta, \{G_{sum:i}\}_{i=\ell+1}^{m}, \{G_{t(x):i}\}_{i=0}^{n-2}\right)$;

  Sample $\delta' \xleftarrow{\$} \mathbb{Z}_p^*$

  $\pi_{\delta'} \leftarrow \mathsf{Prove}_{\mathsf{dl}}^{\mathsf{RO}(\cdot)}(G^{\delta'}, H^{\delta'}, \delta')$

  $\rho \leftarrow (G_\delta^{\delta'}, G^{\delta'}, H^{\delta'}, \pi_{\delta'})$

  $\mathsf{srs}'_s \leftarrow \left(G_\delta^{\delta'}, H_\delta^{\delta'}, \{G_{sum:i}^{1/\delta'}\}_{i=\ell+1}^{m}, \{G_{t(x):i}^{1/\delta'}\}_{i=0}^{n-2}\right)$

  return $((\mathsf{srs}_u, \mathsf{srs}'_s), \rho)$

$\mathsf{Specialize}(\mathcal{R}_{\mathsf{QAP}}, \mathsf{srs}_u)$:

% Computes srs_s with $\delta = 1$

Parse $\mathsf{srs}_u = \left(\{G_{x:i}, H_{x:i}\}_{i=0}^{2n-2}, \{G_{\alpha x:i}, G_{\beta x:i}, H_{\alpha x:i}, H_{\beta x:i}\}_{i=0}^{n-1}\right)$;

$\mathsf{srs}_s \leftarrow \left(G, H, \{\prod_{j=0}^{n-1} G_{\beta x:j}^{u_{ij}} \cdot G_{\alpha x:j}^{v_{ij}} \cdot G_{x:j}^{w_{ij}}\}_{i=\ell+1}^{m}, \{\prod_{j=0}^{n} G_{x:(i+j)}^{t_j}\}_{i=0}^{n-2}\right)$;

return srs_s;

Figure 4.5: Default SRS and update algorithm for Groth's SNARK

$\underline{\mathsf{VerifySRS}^{\mathsf{RO}(\cdot)}(\mathsf{QAP}, \mathsf{srs}, Q)}$:

Parse $\mathsf{srs} = (\mathsf{srs}_u, \mathsf{srs}_s)$ and $Q = (Q_u, Q_s) = \{\rho_{u,i}\}_{i=1}^{k_u} \cup \{\rho_{s,i}\}_{i=1}^{k_s}$
Parse $\mathsf{srs}_u = \left(\{G_{x:i}, H_{x:i}\}_{i=0}^{2n-2}, \{G_{\alpha x:i}, G_{\beta x:i}, H_{\alpha x:i}, H_{\beta x:i}\}_{i=0}^{n-1}\right)$
for $i = 1, \ldots, k_u$ **do**
    Parse $\rho_{u,i} = (\rho_{\alpha'}^{(i)}, \rho_{\beta'}^{(i)}, \rho_{x'}^{(i)})$
    for $\iota \in \{\alpha, \beta, x\}$ **do**
        Parse $\rho_{\iota'}^{(i)} = (G_{\iota}^{(i)}, G_{\iota'}^{(i)}, H_{\iota'}^{(i)}, \pi_{\iota'}^{(i)})$
        assert $\mathsf{Verify}_{\mathsf{dl}}^{\mathsf{RO}(\cdot)}(G_{\iota'}^{(i)}, H_{\iota'}^{(i)}, \pi_{\iota'}^{(i)}) = 1$
        if $i \neq 1$: **assert** $\hat{e}(G_{\iota}^{(i)}, H) = \hat{e}(G_{\iota}^{(i-1)}, H_{\iota'}^{(i)})$
assert $G_{x:1} = G_x^{(k_u)} \neq 1 \wedge G_{\alpha x:0} = G_{\alpha}^{(k_u)} \neq 1 \wedge G_{\beta x:0} = G_{\beta}^{(k_u)} \neq 1$
for $i = 1, \ldots, 2n-2$ **do**
    assert $\hat{e}(G_{x:i}, H) = \hat{e}(G, H_{x:i}) \wedge \hat{e}(G_{x:i}, H) = \hat{e}(G_{x:(i-1)}, H_{x:1})$
for $i = 0, \ldots, n-1$ and $\iota \in \{\alpha, \beta\}$ **do**
    assert $\hat{e}(G_{\iota x:i}, H) = \hat{e}(G, H_{\iota x:i}) \wedge \hat{e}(G_{\iota x:i}, H) = \hat{e}(G_{x:i}, H_{\iota x:0})$
Parse $\mathsf{srs}_s \leftarrow \left(G_\delta, H_\delta, \{G_{sum:i}\}_{i=\ell+1}^{m}, \{G_{t(x):i}\}_{i=0}^{n-2},\right)$
for $i = 1, \ldots, k_s$ **do**
    Parse $\rho_{s,i} = (G_{\delta}^{(i)}, G_{\delta'}^{(i)}, H_{\delta'}^{(i)}, \pi_{\delta'})$
    assert $\mathsf{Verify}_{\mathsf{dl}}^{\mathsf{RO}(\cdot)}(G_{\delta'}^{(i)}, H_{\delta'}^{(i)}, \pi_{\delta'}) = 1$
    if $i \neq 1$ **assert** $\hat{e}(G_{\delta}^{(i)}, H) = \hat{e}(G_{\delta}^{(i-1)}, H_{\delta'}^{(i)})$
assert $\hat{e}(G_\delta, H) = \hat{e}(G, H_\delta)$ and $G_\delta = G_\delta^{(k_s)} \neq 1$
for $i = \ell+1, \ldots, m$ **do**:
    assert $\hat{e}(G_{sum:i}, H_\delta) = \hat{e}(\prod_{j=0}^{n-1} G_{\beta x:j}^{u_{ij}} \cdot G_{\alpha x:j}^{v_{ij}} \cdot G_{x:j}^{w_{ij}}, H)$
for $i = 0, \ldots, n-2$ **do**:
    assert $\hat{e}(G_{t(x):i}, H_\delta) = \hat{e}(G_{t(x)}, H_{x:i})$, where $G_{t(x)} = \prod_{j=0}^{n} G_{x:j}^{t_j}$

Figure 4.6: SRS verification algorithm for Groth16.

Update completeness: Let $\mathcal{A}$ be an adversary that outputs $s = (\varphi, \mathsf{srs}, \{\rho_i\}_i)$ such that $\mathsf{VerifySRS}(s) = 1$. By the observation above, there exists some $\alpha, \beta, x, \delta \in \mathbb{Z}_p^*$ that map to a well-formed srs. It is easy to observe that by construction $\mathsf{UpdateSRS}(\mathsf{QAP}, \varphi, (\mathsf{srs}, \{\rho_i\}_i))$ picks a new $\alpha', \beta', x' \in \mathbb{Z}_p^*$ (or δ' if $\varphi = 2$) and rerandomizes srs such that the new srs' has a trapdoor $\alpha\alpha', \beta\beta', xx' \in \mathbb{Z}_p^*$ (or $\delta\delta' \in \mathbb{Z}_p^*$). Since the srs' is still well-formed and ρ is computed independently, $\mathsf{VerifySRS}(\mathsf{srs}', \{\rho_i\}_i \cup \{\rho'\}) = 1$. See details in Lemma 4.8.2, Section 4.8.1.

Prover completeness: Suppose that $\mathcal{A}$ output $(\mathsf{srs}, \{\rho_i\}_i, \mathsf{x}, \mathsf{w})$ such that $(\mathsf{x}, \mathsf{w}) \in \mathcal{R}_{\mathsf{QAP}}$, and $\mathsf{VerifySRS}(\mathsf{srs}, \{\rho_i\}_i) = 1$. It follows that srs is a well-formed SRS for Groth's SNARK. From here, the prover completeness follows from the completeness proof in [Groth, 2016]. □

Subversion zero-knowledge of Groth's SNARK was independently proven in [Abdolmaleki et al., 2017] and [Fuchsbauer, 2018] under slightly different knowledge assumptions. Our approach here differs only in that we extract the trapdoor from Π_{dl} proofs. For sake of completeness, we sketch the main idea below.

Theorem 4.6.2 (sub-ZK)**.** *If Π_{dl} is a non-interactive proof of knowledge, then Groth16 in Fig. 4.4 is subversion zero-knowledge.*

Proof (sketch). Let $\mathcal{Z}$ be a PPT subverter and $\mathcal{A}$ an unbounded adversary in the subversion zero-knowledge definition. We suppose that $\mathcal{Z}(1^\lambda)$ outputs $(\mathsf{srs}, \{\rho_i\}_i, st)$ such that $\mathsf{VerifySRS}(\mathsf{srs}, \{\rho_i\}_i) = 1$. The latter guarantees that srs is well-formed and that update proofs verify. To prove subversion zero-knowledge, we need to construct an extractor $\mathsf{Ext}_{\mathcal{Z}}$ that give $\mathsf{view}_{\mathcal{Z}}$ extracts the simulation trapdoor for srs. Idea behind $\mathsf{Ext}_{\mathcal{A}}$ is that we use straight-line extractability of Π_{dl} to extract $\iota_1, ..., \iota_m$ for $\iota \in \{x, \alpha, \beta, \delta\}$ from the proofs $\{\rho_i\}_i$ and then compute $\iota = \prod_i \iota_i$ to obtain the trapdoor $\mathsf{td} = (x, \alpha, \beta, \delta)$. Given that $\mathsf{Ext}_{\mathcal{A}}$ outputs the correct trapdoor td, proofs can be perfectly simulated as is proven in [Groth, 2016]. □

4.6.1 Update Knowledge Soundness

Theorem 4.6.3. *Let us assume the $(2n-1, 2n-2)$-**edlog** assumption holds. Then Groth16 in Fig. 4.4 has update knowledge soundness with respect to all PPT algebraic adversaries in the random oracle model.*

$\mathsf{Ext}_{\mathcal{A}}(\mathsf{view}_{\mathcal{A}})$

1. Extract the set of algebraic coefficients $T_\pi \leftarrow \mathsf{Ext}^{\mathsf{agm}}_{\mathcal{A}}(\mathsf{view}_{\mathcal{A}})$ and obtain $\{C_{i:x:j}\}_{i,j=(1,l+1)}^{m_1,m}$ from it, corresponding to the elements $\{(\beta u_i(x) + \alpha v_i(x) + w_i(x))/\delta\}$ in the second phase, where m_1 is the number of update queries made in the first phase, and m is the QAP parameter.
2. From $\mathsf{view}_{\mathcal{A}}$ deduce i_{crit_2} — $\mathcal{O}_{srs}$ query index that corresponds to the last honest update in the final SRS.
3. Return coefficients $\mathsf{w} = \{C_{i_{\mathsf{crit}_2}:x:j}\}_{j=l+1}^{m}$.

Figure 4.7: The extractor $\mathsf{Ext}_{\mathcal{A}}$ for update knowledge soundness

Proof. Let $\mathcal{A}$ be an algebraic adversary against update knowledge soundness and let us denote the update knowledge soundness game $\mathcal{G}_{\mathsf{uks}}$ by $\mathcal{G}_0$. We construct an explicit white-box extractor $\mathsf{Ext}_{\mathcal{A}}$ and prove it to succeed with an overwhelming probability. The theorem statement is thus $\mathsf{Adv}^{\mathcal{G}_0}_{\mathcal{A},\mathsf{Ext}_{\mathcal{A}}}(\lambda) = \mathsf{negl}(\lambda)$. We assume that $\mathcal{A}$ makes at most q_1 update queries in phase 1 and at most q_2 in phase 2. Often we will use ι to denote any of the elements x, α, β or δ.

Description of the extractor $\mathsf{Ext}_{\mathcal{A}}$**.** We present the extractor $\mathsf{Ext}_{\mathcal{A}}$ in Fig. 4.7. The extractor takes the adversarial view $\mathsf{view}_{\mathcal{A}}$ as an input and extracts AGM coefficients from $\mathsf{view}_{\mathcal{A}}$ when $\mathcal{A}$ produces a verifying proof. The goal of the extractor is to reconstruct the witness from this information.

The intuition behind its strategy is that, in Prove in Fig. 4.4, C is constructed as $\sum_i \mathsf{a}_i(\alpha u_i(x) + \beta v_i(x) + w_i(x))/\delta$, and we would like to obtain precisely these a_i as AGM coefficients corresponding to the $(\alpha u_i(x) + \ldots)/\delta$ elements of the *final* SRS. When $\mathcal{A}$ submits the final response $(\mathsf{x}, \pi = (A, B, C))$, the proof element $C \in \mathbb{G}_1$ has the algebraic representation, corresponding to following $\mathbb{G}_1$ elements: (1) SRS elements that the update oracle outputs, (2) corresponding update proofs, and (3) direct RO replies. These sets include *all* the SRS elements that were produced during the update KS game, not only those that were included in the final SRS. The coefficient of elements $(\alpha u_i(x) + \ldots)/\delta$ that the extractor needs belong to the the first category and in particular correspond to the second phase updates, since δ is updated there.

Let m_φ be the number of update queries that $\mathcal{A}$ makes in phase $\varphi \in \{1, 2\}$. We introduce the notion of the *critical* query — $i_{\mathsf{crit}_\varphi} \in \{1, \ldots, m_\varphi\}$ corresponds to the last honest update that $\mathcal{A}$ includes into the finalized SRS in phase φ. Technically, we define it in the following way. For every phase φ, the final SRS is associated with update proofs $\{\rho_{\varphi,i}\}_{i=1}^{k_\varphi}$ (contained in Q^* in Fig. 4.1) and at least one of them must be produced by honest update query for finalization to succeed. Suppose that $\rho_{\varphi,i_{max}}$ is the last honest update in that set, that is, the one with the largest index i. If $\rho_{\varphi,i_{max}}$ was obtained as the j-th update query, then we define $i_{\mathsf{crit}_\varphi} := j$.

The extractor $\mathsf{Ext}_\mathcal{A}$ can deduce $i_{\mathsf{crit}_\varphi}$, since $\mathsf{view}_\mathcal{A}$ includes $\mathcal{O}_{srs}$ responses and Q^*. When $\mathsf{Ext}_\mathcal{A}$ obtains i_{crit_2}, it merely returns the AGM coefficients (which it can obtain from $\mathsf{view}_\mathcal{A}$ since $\mathcal{A}$ is algebraic) corresponding to the $(\alpha u_i(x) + \ldots)/\delta$ elements of update oracle response number i_{crit_2}. For now, there is no guarantee that these elements are in any way connected to the final SRS, but later we show that $\mathsf{Ext}_\mathcal{A}$ indeed succeeds.

Description of $\mathcal{G}_1$. We describe $\mathcal{G}_1$ (see Fig. 4.9 for full details), that differs from $\mathcal{G}_0$ in that one of the honest updates in each phase is a freshly generated SRS instead of being an update of the input SRS. This simplifies further reasoning (Lemma 4.6.5), and also at a later step we build a reduction $\mathcal{B}$ that embeds the edlog challenge z into the trapdoors of the fresh SRS. For convenience, we describe $\mathcal{G}_1$ in terms of communication between the challenger $\mathcal{C}$ (top-level execution code of $\mathcal{G}_1$) and $\mathcal{A}$.

$\mathcal{C}$ of $\mathcal{G}_1$ maintains an update (current call) counter i_{call}, which is reset to zero in the beginning of each phase. Before the game starts, $\mathcal{C}$ uniformly samples two values i_{guess_1} and i_{guess_2}, ranging from $1, \ldots, q_1$ and $1, \ldots, q_2$ (upperbounds on the number of queries) correspondingly, in such a way attempting to guess critical queries $\{i_{\mathsf{crit}_\varphi}\}_\varphi$. In case the actual number of queries m_φ in a particular execution of $\mathcal{A}$ is less than $i_{\mathsf{guess}_\varphi}$, $\mathcal{C}$ will just execute as in $\mathcal{G}_0$ for phase φ. $\mathcal{C}$ will generate fresh SRS for at most two (randomly picked) update queries through $\mathcal{O}_{srs}$, and it will respond to all the other update requests from $\mathcal{A}$ honestly. The successful guess formally corresponds to the event **lucky**, set during SRS finalization in $\mathcal{G}_1$ (see Fig. 4.9).

It is not possible for $\mathcal{C}$ to generate an update proof for a fresh SRS as in $\mathcal{G}_0$ because it does not know the update trapdoors $\hat{\iota}'$ for critical queries — these values do not exist explicitly, since instead of updating an SRS, $\mathcal{C}$ generated a

$\mathcal{O}_{srs}(\mathsf{intent}, \mathsf{srs}^* = (\mathsf{srs}_u^*, \mathsf{srs}_s^*), Q^* = \{\rho_u^{(i)}\}_{i=1}^{k_u} \cup \{\rho_s^{(i)}\}_{i=1}^{k_s})$:

% Update $i_{\mathsf{call}} \leftarrow i_{\mathsf{call}} + 1$ on each successful return
if $\varphi > 2$ **then return** $\perp$
$\mathsf{srs}_{\mathsf{new}} \leftarrow$ **if** $\varphi = 1$ **then** srs^* **else** $(\mathsf{srs}_u, \mathsf{srs}_s^*)$
if $\mathsf{VerifySRS}^{\mathsf{RO}(\cdot)}(\mathsf{srs}_{\mathsf{new}}, Q^*) = 0$ **then return** $\perp$
if intent $= \mathsf{UPDATE} \land \varphi = 1 \land i_{\mathsf{call}} = i_{\mathsf{guess}_1}$ **then** % Simulated update
  $\mathsf{srs}_u' \leftarrow \left(\{G^{z_x^i}, H^{z_x^i}\}_{i=0}^{2n-2}, \{G^{z_\alpha z_x^i}, G^{z_\beta z_x^i}, H^{z_\alpha z_x^i}, H^{z_\beta z_x^i}\}_{i=0}^{n-1}\right)$;
  $\mathsf{srs}_s' \leftarrow \mathsf{Specialize}(\mathcal{R}_{\mathsf{QAP}}, \mathsf{srs}_u')$
  for $\iota \in \{x, \alpha, \beta\}$ **do** $\rho_{\iota'} \leftarrow \mathsf{SimUpdProof}(z_\iota, \varphi = u)$;
  return $(\mathsf{srs}', (\rho_{\alpha'}, \rho_{\beta'}, \rho_{x'}))$
if intent $= \mathsf{UPDATE} \land \varphi = 2 \land i_{\mathsf{call}} = i_{\mathsf{guess}_2}$ **then** % Simulated update
  Let $\{\hat{z}_\iota\}_{\iota \in x,\alpha,\beta}$ correspond to the trapdoors at the end of phase 1
  $\mathsf{srs}_s' \leftarrow \left(G^{z_\delta}, H^{z_\delta}, \{G^{\frac{\hat{z}_x^i t(\hat{z}_x)}{z_\delta}}\}_{i=0}^{n-2}, \{G^{\frac{\hat{z}_\beta u_i(\hat{z}_x) + \hat{z}_\alpha v_i(\hat{z}_x) + w_i(\hat{z}_x)}{z_\delta}}\}_{i=\ell+1}^{m}\right)$
  $\rho_\delta' \leftarrow \mathsf{SimUpdProof}(z_\delta, \varphi = s)$ **return** $((\mathsf{srs}_u^*, \mathsf{srs}_s'), \rho_\delta')$
if intent $= \mathsf{UPDATE}$ **then** % Honest update
  $(\mathsf{srs}', \rho') \leftarrow \mathsf{UpdateSRS}(\varphi, \mathsf{srs}_{\mathsf{new}}, Q^*)$; $Q_\varphi \leftarrow Q_\varphi \cup \{\rho'\}$
  return (srs', ρ')
if intent $= \mathsf{FINALIZE} \land Q_\varphi \cap Q^* \neq \emptyset$ **then**
  if $\varphi = 1$ **then** $\mathsf{srs}_u \leftarrow \mathsf{srs}_u^*$ **else** $\mathsf{srs}_s \leftarrow \mathsf{srs}_s^*$
  $\varphi \leftarrow \varphi + 1$; $i_{\mathsf{call}} \leftarrow 0$
if $\varphi > 2$ **then**
  Deduce $\{i_{\mathsf{crit}_\varphi}\}_\varphi$ from Q^* as last honest updates in phase φ
  lucky $:= \left(i_{\mathsf{guess}_1} = i_{\mathsf{crit}_1} \land i_{\mathsf{guess}_2} = i_{\mathsf{crit}_2}\right)$

$\mathsf{SimUpdProof}(z_\iota, \varphi)$:

% PoKs may correspond both to honest and malicious updates
$\{\hat{\iota}_j\}_{j=1}^{k_\varphi} \leftarrow$ extract trapdoors from $\{\rho_\varphi^{(i)}\}_{i=1}^{k_\varphi}$ PoKs using $\mathsf{view}_\mathcal{A}$
$\hat{\iota} \leftarrow \prod^{k_\varphi} \hat{\iota}_j$; $G^{\hat{\iota}'} \leftarrow (G^{z_\iota})^{\hat{\iota}^{-1}}$; $H^{\hat{\iota}'} \leftarrow (H^{z_\iota})^{\hat{\iota}^{-1}}$
$\pi_{\iota'} \leftarrow \mathsf{Sim}_{\mathsf{dl}}^{\mathsf{RO}_1(\cdot)}(\mathsf{x}_{\mathsf{dl}} = (\perp, G^{\hat{\iota}'}, H^{\hat{\iota}'}))$
$\rho_{\iota'} \leftarrow (G^{z_\iota}, G^{\hat{\iota}'}, H^{\hat{\iota}'}, \pi_{\iota'})$; **return** $\rho_{\iota'}$

Figure 4.8: Oracles used in $\mathcal{G}_1$ (Fig. 4.9), a modified update KS game.

$\underline{\mathcal{G}_1^{\mathcal{A},\mathsf{Ext}_\mathcal{A}}(1^\lambda):}$

$\mathsf{srs} \leftarrow \mathsf{srs}^\mathsf{d}, \varphi = 1$

$Q_1, Q_2 \leftarrow \emptyset; i_\mathsf{call} \leftarrow 0; i_{\mathsf{guess}_1} \xleftarrow{\$} [0, q_1]; i_{\mathsf{guess}_2} \xleftarrow{\$} [0, q_2]$

$\{z_\iota\}_{\iota \in \{x,\alpha,\beta,\delta\}} \xleftarrow{\$} \mathbb{Z}_p$

Initialize $\mathsf{RO}_t(\cdot)$

$(\mathsf{x}, \pi) \leftarrow \mathcal{A}^{\mathcal{O}_{srs},\mathsf{RO}};\ w \leftarrow \mathsf{Ext}_\mathcal{A}(\mathsf{view}_\mathcal{A})$

return $\mathsf{Verify}(\mathsf{srs}, \mathsf{x}, \pi) = 1 \wedge (\mathsf{x}, \mathsf{w}) \notin \mathcal{R} \wedge \varphi > 2$

Figure 4.9: Description of $\mathcal{G}_1$, a modified update KS game. Helper oracles are presented in Fig. 4.8.

new one. Therefore, it uses a specific technique to simulate update proofs using the procedure SimUpdProof(see Fig. 4.9). The task of SimUpdProof is to create $\rho_{\hat{\iota}'} = (G_{\hat{\iota}}^{\hat{\iota}'}, G^{\hat{\iota}'}, H^{\hat{\iota}'}, \pi_{\hat{\iota}'})$, which is a valid update proof from srs^* to a freshly generated srs'. Since $\mathcal{C}$ does not actually update srs^*, but creates a completely new one with z_ι trapdoors, we have $G^{z_\iota} = G^{\hat{\iota}\hat{\iota}'}$ where $\hat{\iota}$ is the trapdoor value of srs^* and $\hat{\iota}'$ is the new update trapdoor. Given the value $\hat{\iota}$ in clear, we can reconstruct $G^{\hat{\iota}'}$ by computing $(G^{\hat{\iota}\hat{\iota}'})^{\hat{\iota}^{-1}} = (G^{z_\iota})^{\hat{\iota}^{-1}}$.

This is the strategy of $\mathcal{C}$: it uses $\mathsf{view}_\mathcal{A}$ to extract the trapdoors ι_j for all the k_u updates that led to srs^*_φ, and thus obtains $\hat{\iota}$. Notice that these updates can be both honest and adversarial, but importantly, none of them are simulated (because we perform this procedure only once per phase), which guarantees that extraction succeeds. Next, SimUpdProof computes a product $\hat{\iota}$ of these extracted values, and using its inverse produces $(G^{\hat{\iota}'}, H^{\hat{\iota}'})$, which are the second and third elements of the update proof. The first element of $\rho_{\hat{\iota}'}$ is just an element of the new SRS (e.g. for $\iota = x$, it is $G_{x:1}^{\iota'}$, and for $\iota \in \{\alpha, \beta\}$ it is $G_{\iota x:0}^{\iota'}$), so we set the value to G^{z_ι}. The last element, the proof-of-knowledge of $\hat{\iota}'$, we create by black-box simulation, since Π_dl is perfectly ZK. Namely, since the challenger already has $\mathsf{x}_\mathsf{dl} = (\perp, G^{\hat{\iota}'}, H^{\hat{\iota}'})$, it passes it into Sim_dl, and attaches the resulting $\pi_{\iota'}$ to the update proof. Since we know z_ι in $\mathcal{G}_1$ (and therefore know x_dl exponent $\hat{\iota}'$), it is not necessary to simulate the proof in $\mathcal{G}_1$ — technically, the procedure only requires G^{z_ι}. However, simulation will be critical in the final part of our theorem, reduction to edlog, since in that case z_ι contains embedded edlog challenge for which the challenger does not know the exponent. This is why we introduce it here in $\mathcal{G}_1$.

We prove in Section 4.8.2.1 that the game $\mathcal{G}_1$ that we introduced is indistinguishable from $\mathcal{G}_0$ for $\mathcal{A}$ by relying on the zero-knowledge and simulation-extractability properties of Π_{dl}. We recall that $(1,0)$-**dlog** assumption is implied by $(2n-1, 2n-2)$-**edlog** assumption.

Lemma 4.6.4. *Assuming $(1,0)$-**dlog**, the difference between advantage of $\mathcal{A}$ in winning $\mathcal{G}_0$ and $\mathcal{G}_1$ is negligible:* $\mathsf{Adv}^{\mathcal{G}_0}_{\mathcal{A},\mathsf{Ext}_{\mathcal{A}}}(\lambda) \leq \mathsf{Adv}^{\mathcal{G}_1}_{\mathcal{A},\mathsf{Ext}_{\mathcal{A}}}(\lambda) + \mathsf{negl}(\lambda)$.

Reconstructing the proof algebraically. For the next steps of our proof we will need to be able to reconstruct the proof elements, and the verification equation generically from the AGM coefficients we extract from $\mathcal{A}$. Almost all the elements that $\mathcal{A}$ sees depend on certain variables $\vec{\Psi}$ that are considered secret for the adversary (update trapdoors, RO exponents, critical query honest trapdoors). Since $\mathcal{A}$ can describe proof elements A, B, C as linear combinations of elements it sees, that depend on $\vec{\Psi}$, we are able to reconstruct the proof elements as functions $A(\vec{\Psi}), B(\vec{\Psi}), C(\vec{\Psi})$ (Laurent polynomials, as we will show later). That is, for the particular values $\vec{\psi}$ that we chose in some execution in $\mathcal{G}_1$, $A(\vec{\psi}) = A$ (but we can also evaluate $A(\vec{\Psi})$ on a different set of trapdoors). From these functions $A(\vec{\Psi}), B(\vec{\Psi}), C(\vec{\Psi})$ one can reconstruct a SNARK verification equation $Q(\vec{\Psi})$, such that $\mathsf{Verify}(\psi, \pi) = 1 \iff Q(\vec{\psi}) = 0$.

We note that it is not trivial to obtain the (general) form of these functions, because it depends on $\mathsf{view}_{\mathcal{A}}$ — different traces produce different elements that $\mathcal{A}$ sees, which affects with which functions these elements are modelled. Therefore, we start by defining which *variables* are used to model elements that $\mathcal{A}$ sees.

We denote by $\vec{\Psi}$ this set of variables which are unknown to $\mathcal{A}$. This includes, first and foremost, the set of trapdoors that are used for the (critical) simulation update queries: $Z_x, Z_\alpha, Z_\beta, Z_\delta$ (these abstract the corresponding trapdoors $\{z_\iota\}$). To denote the expression that includes final adversarial trapdoors $\iota_j^{\mathcal{A}}$, we will use $\hat{Z}_\iota$ that is equal to the previously defined Z_ι, but now as a function of Z_ι: $\hat{Z}_\iota(Z_\iota) = Z_\iota \prod \iota_j^{\mathcal{A}}$ for $\iota \in \{x, \alpha, \beta\}$, and $\hat{Z}_\delta(Z_\delta) = Z_\delta / \prod \delta_j^{\mathcal{A}}$.[11]

The full list of variables that constitute $\vec{\Psi}$ is the following:

1. Critical honest trapdoor variables: $Z_\alpha, Z_\beta, Z_x, Z_\delta$.

[11] If $\hat{Z}_\iota$ is not equal $Z_\iota \prod \iota_j^{\mathcal{A}}$ as a function we have $\hat{Z}_\iota(\Psi) - Z_\iota \prod \iota_j^{\mathcal{A}} \neq 0$ but $\hat{Z}_\iota(\psi) - z_\iota \prod \iota_j^{\mathcal{A}} \equiv 0$ for $\iota \in \{x, \alpha, \beta, \delta\}$, and we break the $(2n-1, 2n-2)$-**edlog** problem as in Lemma 4.6.7.

2. Honest (non-critical) update trapdoors $\vec{T} = \{T_{i,\iota}\}$.
3. RO replies, which we, for convenience of indexing, split into three disjoint sets:
   - RO values for the critical queries $\vec{K} = \{K_\iota\}_{x,\alpha,\beta,\delta}$: these RO replies are used in PoK simulation by $\mathcal{G}_1$.
   - RO values for honest update proofs $\vec{R}_T = \{R_{T:i:\iota}\}_{i,\iota}$. First phase update query number $i \in \{1, \dots, m_1\}$ corresponds to three values $R_{T:i:x}$, $R_{T:i:\alpha}$, $R_{T:i:\beta}$, and second phase update query number $j \in \{1, \dots, m_2\}$ corresponds to $R_{T:j:\delta}$.
   - RO responses $\vec{R}_{\mathcal{A}}$ that $\mathcal{A}$ directly requests from RO. These are used by $\mathcal{A}$, in particular, but not only, to create PoKs for adversarial SRS updates.

We denote by $\vec{R} = \vec{R}_{\mathcal{A}} \cup \vec{R}_T$. Therefore, $\vec{\Psi} = (\{Z_\iota\}_\iota, \vec{K}, \vec{T}, \vec{R})$. Since we will be often working only with the first set of variables $\{Z_\iota\}$, we will denote it as $\vec{\Psi}_2$, and all other variables from $\vec{\Psi}$ as $\vec{\Psi}_1$.

Success in lucky executions. In general, the set structure of $Q(\vec{\Psi})$ can vary enormously, and it depends on many things, including the way $\mathcal{A}$ interacts with the challenger. Each interaction can present a different set of coefficients in $\mathcal{A}$ that will be modelled by different functions. Therefore, we would like to take advantage of the **lucky** event to simplify our reasoning and reduce the space of possible interactions.

We claim that **lucky** is independent from $\mathcal{A}$'s success in $\mathcal{G}_1$. In other words, in order to win $\mathcal{G}_1$ it suffices to only show the existence of a witness extractor in the case where the lucky indices correspond to $\mathcal{A}$'s critical queries.

$$\mathsf{Adv}^{\mathcal{G}_1}_{\mathcal{A},\mathsf{Ext}_{\mathcal{A}}}(\lambda) = \Pr[\mathcal{G}_1^{\mathcal{A},\mathsf{Ext}_{\mathcal{A}}}(1^\lambda) = 1] = \Pr[\mathcal{G}_1^{\mathcal{A},\mathsf{Ext}_{\mathcal{A}}}(1^\lambda) = 1 \mid \textbf{lucky}]$$

where q_1 and q_2 are polynomially bounded. Indeed, $\mathcal{A}$ is blind to whether we simulate or not, and so we can assume independence of events: $\Pr[\mathcal{G}_1^{\mathcal{A},\mathsf{Ext}_{\mathcal{A}}}(1^\lambda) = 1 \mid \textbf{sim}_i]$ is the same for all simulation strategies $\textbf{sim}_i$, including the lucky one.

$$\begin{aligned}
\mathsf{Adv}^{\mathcal{G}_1}_{\mathcal{A},\mathsf{Ext}_{\mathcal{A}}}(\lambda) &= \sum_{i=0}^{q_1 q_2} \Pr[\mathcal{G}_1^{\mathcal{A},\mathsf{Ext}_{\mathcal{A}}}(1^\lambda) = 1 \mid \textbf{sim}_i] \frac{1}{q_1 q_2} \\
&= \frac{1}{q_1 q_2} \sum_i \Pr[\mathcal{G}_1^{\mathcal{A},\mathsf{Ext}_{\mathcal{A}}}(1^\lambda) = 1 \mid \textbf{lucky}] = \Pr[\mathcal{G}_1^{\mathcal{A},\mathsf{Ext}_{\mathcal{A}}}(1^\lambda) = 1 \mid \textbf{lucky}]
\end{aligned}$$

Our choice of $\{i_{\text{guess}_\varphi}\}_\varphi$, and thus the chosen simulation strategy $\mathbf{sim}_i$ is independent from the success of $\mathcal{A}$. This does not imply that we ignore some traces of $\mathcal{A}$, which would break the reduction. Instead, for each possible trace of $\mathcal{A}$, and thus each possible way it communicates with the challenger and the oracles, we only consider those executions in which we guess the indices correctly.

Defining the function $Q(\vec{\Psi})$ for $\mathcal{G}_1$. Therefore, when in $\mathcal{G}_1$ the challenger guesses critical queries correctly (**lucky**), and $\mathcal{A}$ returns a verifying proof, the complexity is greatly simplified, and we can now define at least the high-level form of the function Q:

$$Q(\vec{\Psi}) := \left(A(\vec{\Psi})B(\vec{\Psi}) - \hat{Z}_\alpha \hat{Z}_\beta - \sum_{i=0}^{\ell} \mathsf{a}_i(\hat{Z}_\beta u_i(\hat{Z}_x) + \hat{Z}_\alpha v_i(\hat{Z}_x) + w_i(\hat{Z}_x)) - C(\vec{\Psi})\hat{Z}_\delta \right) \tag{4.3}$$

such that $G^{A(\vec{\psi})} = A$ and similarly for B and C, where $\vec{\psi}$ is the concrete set of secret values used for a particular execution.[12] The function $Q(\vec{\Psi})$ reconstructs verification equation of the proof in this particular game execution: in particular, $Q(\vec{\psi}) = 0 \iff \mathsf{Verify}(\mathsf{srs}, \mathsf{x}, \pi) = 1$.

Note that the form of functions $A(\vec{\Psi}), B(\vec{\Psi})$, and $C(\vec{\Psi})$ depends on the interaction with $\mathcal{A}$, and thus on the particular execution trace. But the general form of Q we have just specified is enough to argue the critical lemmas. The proof of the following Lemma, which shows exactly that, is deferred to Section 4.8.2.2.

Lemma 4.6.5. *In $\mathcal{G}_1$, conditioned on event* **lucky***, the general form of the function $Q(\vec{\Psi})$ reconstructing the main verification equation is as presented in Eq.* (4.3)*, under $(2n-1, 2n-2)$-**edlog**. Moreover, A, B, C are Laurent polynomials in $\vec{\Psi}_2$ when viewed over $\mathbb{Z}_p[\vec{C}, \vec{\Psi}_1]$, where $\vec{C}$ are AGM coefficients, abstracted as variables. In other words, $A, B, C \in (\mathbb{Z}_p[\vec{C}, \vec{\Psi}_1])[\vec{\Psi}_2]$ are Laurent. Therefore, Q also is Laurent when viewed as $(\mathbb{Z}_p[\vec{C}, \vec{\Psi}_1])[\vec{\Psi}_2]$ element.*

Description of $\mathcal{G}_2$. The following game, presented in Fig. 4.10 extends $\mathcal{G}_1$ with two additions. Firstly, it introduces the event **bad**. The condition that we are trying

[12] The form of the proof-independent parts of the verification equation (see Eq. (4.4)) is due to our critical-step-simulation strategy that we introduce in $\mathcal{G}_1$. That is, these values they only depend on the challenge variables Z_ι plus last adversarial trapdoors (e.g. $\prod \alpha_i^{\mathcal{A}}$ etc). This is where guessing the *last* query really helps: otherwise these terms would also depend on Ψ_1, e.g. on $\vec{T}$.

$\underline{\mathcal{G}_2^{\mathcal{A},\mathsf{Ext}_\mathcal{A}}(1^\lambda):}$

$\mathsf{srs} \leftarrow \mathsf{srs}^{\mathsf{d}}, \varphi = 1$
$Q_1, Q_2 \leftarrow \emptyset; i_{\mathsf{call}} \leftarrow 0; i_{\mathsf{guess}_1} \xleftarrow{\$} [0, q_1];\ i_{\mathsf{guess}_2} \xleftarrow{\$} [0, q_2]$
$\{z_\iota\}_{\iota \in \{x,\alpha,\beta,\delta\}} \xleftarrow{\$} \mathbb{Z}_p$
RO_t, $\mathcal{O}_{srs}$ and $\mathsf{SimUpdProof}$ are constructed as in $\mathcal{G}_1$
$(\mathsf{x}, \pi) \leftarrow \mathcal{A}^{\mathcal{O}_{srs},\mathsf{RO}}$
$w \leftarrow \mathsf{Ext}_\mathcal{A}(\mathsf{view}_\mathcal{A})$
bad $:= \Big(\textbf{lucky} \wedge Q(\psi_1, \{z_\iota\}) = 0 \wedge Q(\psi_1, \{Z_\iota\}) \not\equiv 0\Big)$
return $\mathsf{Verify}(\mathsf{srs}, \mathsf{x}, \pi) = 1 \wedge (\mathsf{x}, \mathsf{w}) \notin \mathcal{R} \wedge \varphi > \varphi_{max} \wedge$ **lucky**

Figure 4.10: Description of $\mathcal{G}_2$, an extension of $\mathcal{G}_1$ with **bad** event. $Q(\vec{\Psi}_1, \vec{\Psi}_2)$ is the function (Laurent polynomial in $\vec{\Psi}_2$) that corresponds to the way to reconstruct π and verification equation, where Ψ_2 corresponds to the trapdoor variables $\{Z_\iota\}$.

to capture is whether $\mathcal{A}$ uses the elements that depend on trapdoors z_ι blindly or not. When **bad** does not happen, the adversary is constructing π in such a way that it works for any value of z'_ι ($Q(\psi_1, \{Z_\iota\})$ is a zero as a polynomial). Otherwise, we can argue that $\mathcal{A}$'s cheating strategy depends on the specific value of z_ι, even though it is hidden in the exponent ($Q(\psi_1, \{z_\iota\}) = 0$, but $Q(\psi_1, \{Z_\iota\})$ is a non-zero polynomial).

Secondly, we require that adversary wins only if the event **lucky** happens. Since lucky is an independent event, then $\Pr[\mathcal{G}_2^{\mathcal{A},\mathsf{Ext}_\mathcal{A}}(1^\lambda) = 1] = \Pr[\mathcal{G}_1^{\mathcal{A},\mathsf{Ext}_\mathcal{A}}(1^\lambda) = 1 \wedge$ **lucky**$] = \Pr[\mathcal{G}_1^{\mathcal{A},\mathsf{Ext}_\mathcal{A}}(1^\lambda) = 1]/(q_1 q_2)$. The last transition is due to independence of winning $\mathcal{G}_1$ and **lucky** explained earlier ($\Pr[\mathcal{G}_1^{\mathcal{A},\mathsf{Ext}_\mathcal{A}}(1^\lambda) = 1] = \Pr[\mathcal{G}_1^{\mathcal{A},\mathsf{Ext}_\mathcal{A}}(1^\lambda) = 1 \mid$ **lucky**$]$). We can use the total probability formula to condition winning in $\mathcal{G}_2$ on the event **bad**.

$$\begin{aligned}
\Pr[\mathcal{G}_2^{\mathcal{A},\mathsf{Ext}_\mathcal{A}}(1^\lambda) = 1] &= \Pr[\mathcal{G}_2^{\mathcal{A},\mathsf{Ext}_\mathcal{A}}(1^\lambda) = 1 \mid \neg\textbf{bad}] \cdot \Pr[\neg\textbf{bad}] \\
&\quad + \Pr[\mathcal{G}_2^{\mathcal{A},\mathsf{Ext}_\mathcal{A}}(1^\lambda) = 1 \mid \textbf{bad}] \cdot \Pr[\textbf{bad}] \\
&\leq \Pr[\mathcal{G}_2^{\mathcal{A},\mathsf{Ext}_\mathcal{A}}(1^\lambda) = 1 \mid \neg\textbf{bad}] + \Pr[\textbf{bad}].
\end{aligned}$$

The next two lemmas will upperbound this probability. The Lemma 4.6.6 will bound the first term of the sum and the Lemma 4.6.7 bounds the second term.

Extractor succeeds in good executions. In this subsection we present a lemma, that states that whenever $\mathcal{C}$ guesses the critical indices correctly, and event **bad**

does not happen, the output of the extractor $\mathsf{Ext}_{\mathcal{A}}$ is a QAP witness. The proof of Lemma 4.6.6 is presented in Section 4.8.2.3.

Lemma 4.6.6. *In $\mathcal{G}_2$, when $\neg$**bad** happens and $\mathcal{A}$ produces a verifying proof, then* $\mathsf{Ext}_{\mathcal{A}}$ *succeeds:* $\Pr[\mathcal{G}_2^{\mathcal{A},\mathsf{Ext}_{\mathcal{A}}}(1^\lambda) = 1 \mid \neg\textbf{bad}] = \mathsf{negl}(\lambda)$.

Description of the EDLOG reduction. We show that the event **bad** can only happen with a negligible probability by making a reduction to the edlog assumption. If $\mathcal{A}$ triggers **bad**, then it could construct a proof in a manner that is specific to the SRS $\vec{\psi}_2$ and does not generalize to any other $\vec{\psi}'_2$. This means that $\mathcal{A}$ has knowledge of the exponent element, which is impossible assuming edlog. The proof of the following lemma is delayed to Section 4.8.2.4.

Lemma 4.6.7. *The probability of* **bad** *in $\mathcal{G}_2$ is negligible under the $(2n-1, 2n-2)$-**edlog** assumption.*

Now, combining the results of Lemma 4.6.6 and Lemma 4.6.7 with previous game transitions:

$$\begin{aligned}
\Pr[\mathcal{G}_0^{\mathcal{A},\mathsf{Ext}_{\mathcal{A}}}(1^\lambda) = 1] &\leq \Pr[\mathcal{G}_1^{\mathcal{A},\mathsf{Ext}_{\mathcal{A}}}(1^\lambda) = 1] + \mathsf{negl}(\lambda) \\
&= (q_1 q_2) \Pr[\mathcal{G}_2^{\mathcal{A},\mathsf{Ext}_{\mathcal{A}}}(1^\lambda) = 1] + \mathsf{negl}(\lambda) \\
&\leq (q_1 q_2)\big(\Pr[\mathcal{G}_2^{\mathcal{A},\mathsf{Ext}_{\mathcal{A}}}(1^\lambda) = 1 \mid \neg\textbf{bad}] + \Pr[\textbf{bad}]\big) + \mathsf{negl}(\lambda) \\
&= (q_1 q_2)(\mathsf{negl}(\lambda) + \mathsf{negl}(\lambda)) + \mathsf{negl}(\lambda) = \mathsf{negl}(\lambda)
\end{aligned}$$

This concludes the proof of the update knowledge soundness theorem. $\square$

4.7 Batched VerifySRS

We provide an optimized VerifySRS algorithm for Groth's SNARK. It follows closely the batching techniques used in [Abdolmaleki et al., 2017] for verifying the SRS for subversion zero-knowledge Groth's SNARK. Our approach only differs in that we also consider update proofs.

We briefly remind the main idea behind the batching technique. Suppose the verifier has to verify a set of pairing equations of the form $\hat{e}(G_i, H) = \hat{e}(G, H_i)$ for $i = 1, \ldots, n$. The naive way of checking those equations would require $2n$

pairings. Batching technique can be used substitute most of those pairings with small exponent multi-exponentiations which is much cheaper. Idea is to sample $s_1, \dots, s_n \xleftarrow{\$} \mathbb{Z}_p$ and instead verify a single equation

$$\prod_{i=1}^{n} \hat{e}(G_i, H)^{s_i} = \prod_{i=1}^{n} \hat{e}(G, H_i)^{s_i}.$$

By using bilinear properties, the latter equation can be simplified to

$$\hat{e}(\prod_{i=1}^{n} G_i^{s_i}, H) = \hat{e}(G, \prod_{i=1}^{n} H_i^{s_i}).$$

This equation requires only 2 n-wise multi-exponentions and 2 pairings. It can be shown using the Schwartz-Zippel lemma that the probability that one of the initial equations does not hold and $\hat{e}(\prod_{i=1}^{n} G_i^{s_i}, H) = \hat{e}(G, \prod_{i=1}^{n} H_i^{s_i})$ holds is bounded by $1/p$. Since this is a very low probability, we can even sample s_i from a much smaller set to further speed up the exponentiation. For example, we may sample $s_i \in \{0,1\}^{40}$, which will give an error $1/2^{40}$.

We apply this technique to $\mathsf{VerifySRS}$ in Fig. 4.11 to construct a batched $\mathsf{batchVerifySRS}$.

Theorem 4.7.1. *Take any (possibly malformed)* srs *and* Q *and any valid* QAP*. Then,*

$$\Pr[\mathsf{VerifySRS}(\mathsf{QAP}, \mathsf{srs}, Q) \neq \mathsf{batchVerifySRS}(\mathsf{QAP}, \mathsf{srs}, Q)] \leq 12/2^{\kappa},$$

where the probability is taken over random coin-tosses of $\mathsf{batchVerifySRS}$*.*

Proof. Let us consider a set of equations in a general form $\hat{e}(G^{a_i}, H^{b_i}) = \hat{e}(G^{c_i}, H^{d_i})$ for $i \in \{1, \dots, t\}$ and let $\prod_{i=1}^{t} \hat{e}(G^{a_i}, H^{b_i})^{s_i} = \prod_{i=1}^{t} \hat{e}(G^{c_i}, H^{d_i})^{s_i}$ be the respective batched equation, where $s_i \xleftarrow{\$} \{0,1\}^{\kappa}$. All of the batched equations in $\mathsf{batchVerifySRS}$ follow this form. It is clear that if the initial equations are satisfied, then also the batched equation is satisfied. Thus, $\mathsf{VerifySRS}(\mathsf{QAP}, \mathsf{srs}, Q) = 1$ implies $\mathsf{batchVerifySRS}(\mathsf{QAP}, \mathsf{srs}, Q) = 1$.

We can rewrite the batched equation as $\hat{e}(G, H)^{\sum_{i=1}^{t}(a_i b_i - c_i d_i) s_i} = \hat{e}(G, H)^0$. Let us now consider the polynomial $p(X_1, \dots, X_n) = \sum_{i=1}^{t}(a_i b_i - c_i d_i) X_i$. If one of the initial equations is not satisfied then p is a non-zero polynomial and the probability $p(s_1, \dots, s_t) = 0$ is bounded by $1/2^{\kappa}$. Given that we batch 12 sets of equations, $\Pr[\mathsf{VerifySRS}(\mathsf{QAP}, \mathsf{srs}, Q) = 0 \wedge \mathsf{batchVerifySRS}(\mathsf{QAP}, \mathsf{srs}, Q) = 1] \leq 12/2^{\kappa}$. $\square$

$\underline{\mathsf{batchVerifySRS}^{\mathsf{RO}(\cdot)}(\mathsf{QAP}, \mathsf{srs}, Q)}$:

Parse $\mathsf{srs} = (\mathsf{srs}_u, \mathsf{srs}_s)$ and $Q = (Q_u, Q_s) = \{\rho_{u,i}\}_{i=1}^{k_u} \cup \{\rho_{s,i}\}_{i=1}^{k_s}$
Parse $\mathsf{srs}_u = \left(\{G_{x:i}, H_{x:i}\}_{i=0}^{2n-2}, \{G_{\alpha x:i}, G_{\beta x:i}, H_{\alpha x:i}, H_{\beta x:i}\}_{i=0}^{n-1}\right)$
Sample $s_0, \ldots, s_{\mathsf{max}} \xleftarrow{\$} \{0,1\}^\kappa$ where $\mathsf{max} = \max\{2n-2, m, k_u, k_s\}$
for $i = 1, \ldots, k_u$ **do**
  Parse $\rho_{u,i} = (\rho_{\alpha'}^{(i)}, \rho_{\beta'}^{(i)}, \rho_{x'}^{(i)})$
  for $\iota \in \{\alpha, \beta, x\}$ **do**
    Parse $\rho_{\iota'}^{(i)} = (G_\iota^{(i)}, G_{\iota'}^{(i)}, H_{\iota'}^{(i)}, \pi_{\iota'}^{(i)})$
    $R_{\iota'}^{(i)} \leftarrow \mathsf{RO}(G_{\iota'}^{(i)}, H_{\iota'}^{(i)})$
for $\iota \in \{\alpha, \beta, x\}$ **do**
  assert $\hat{e}(\prod_{i=2}^{k_u}(G_\iota^{(i)})^{s_i}, H) = \prod_{i=2}^{k_u} \hat{e}((G_\iota^{(i-1)})^{s_i}, H_{\iota'}^{(i)})$
  assert $\hat{e}(\prod_{i=1}^{k_u}(G_{\iota'}^{(i)})^{s_i}, H) = \hat{e}(G, \prod_{i=1}^{k_u}(H_{\iota'}^{(i)})^{s_i})$
  assert $\hat{e}(\prod_{i=1}^{k_u}(\pi_{\iota'}^{(i)})^{s_i}, H) = \prod_{i=1}^{k_u} \hat{e}((R_{\iota'}^{(i)})^{s_i}, H_{\iota'}^{(i)})$
assert $G_{x:1} = G_x^{(k_u)} \neq 1 \wedge G_{\alpha x:0} = G_\alpha^{(k_u)} \neq 1 \wedge G_{\beta x:0} = G_\beta^{(k_u)} \neq 1$
assert $\hat{e}(\prod_{i=1}^{2n-2} G_{x:i}^{s_i}, H) = \hat{e}(G, \prod_{i=1}^{2n-2} H_{x:i}^{s_i}) \wedge$
    $\hat{e}(\prod_{i=1}^{2n-2} G_{x:i}^{s_i}, H) = \hat{e}(\prod_{i=1}^{2n-2} G_{x:(i-1)}^{s_i}, H_{x:1})$
for $\iota \in \{\alpha, \beta\}$ **do**
  assert $\hat{e}(\prod_{i=0}^{n-1} G_{\iota x:i}^{s_i}, H) = \hat{e}(G, \prod_{i=0}^{n-1} H_{\iota x:i}^{s_i}) \wedge$
    $\hat{e}(\prod_{i=0}^{n-1} G_{\iota x:i}^{s_i}, H) = \hat{e}(\prod_{i=0}^{n-1} G_{x:i}^{s_i}, H_{\iota x:0})$
Parse $\mathsf{srs}_s = \left(G_\delta, H_\delta, \{G_{sum:i}\}_{i=\ell+1}^m, \{G_{t(x):i}\}_{i=0}^{n-2},\right)$
for $i = 1, \ldots, k_s$ **do**
  Parse $\rho_{s,i} = (G_\delta^{(i)}, G_{\delta'}^{(i)}, H_{\delta'}^{(i)}, \pi_{\delta'})$; $R_{\delta'}^{(i)} \leftarrow \mathsf{RO}(G_{\delta'}^{(i)}, H_{\delta'}^{(i)})$
assert $\hat{e}(\prod_{i=2}^{k_s}(G_\delta^{(i)})^{s_i}, H) = \prod_{i=2}^{k_s} \hat{e}((G_\delta^{(i-1)})^{s_i}, H_{\delta'}^{(i)})$
assert $\hat{e}(\prod_{i=1}^{k_s}(G_{\delta'}^{(i)})^{s_i}, H) = \hat{e}(G, \prod_{i=1}^{k_s}(H_{\delta'}^{(i)})^{s_i})$
assert $\hat{e}(\prod_{i=1}^{k_s}(\pi_{\delta'}^{(i)})^{s_i}, H) = \prod_{i=1}^{k_s} \hat{e}((R_{\delta'}^{(i)})^{s_i}, H_{\delta'}^{(i)})$
assert $\hat{e}(G_\delta, H) = \hat{e}(G, H_\delta)$ and $G_\delta = G_\delta^{(k_s)} \neq 1$
assert $\hat{e}(\prod_{i=\ell+1}^m G_{sum:i}^{s_i}, H_\delta) = \hat{e}(\prod_{i=\ell+1}^m \left(\prod_{j=0}^{n-1} G_{\beta x:j}^{u_{ij}} \cdot G_{\alpha x:j}^{v_{ij}} \cdot G_{x:j}^{w_{ij}}\right)^{s_i}, H)$
assert $\hat{e}(\prod_{i=0}^{n-2} G_{t(x):i}^{s_i}, H_\delta) = \hat{e}(G_{t(x)}, \prod_{i=0}^{n-2} H_{x:i}^{s_i})$, where $G_{t(x)} = \prod_{j=0}^n G_{x:j}^{t_j}$

Figure 4.11: Batched SRS verification algorithm for Groth's SNARK where $\kappa \approx 2^{40}$

4.8 Deferred Proofs

4.8.1 Lemmas for Groth16 Completeness

This section presents the additional lemmas for the completeness proof of Theorem 4.6.1.

Lemma 4.8.1. *If SRS passes* VerifySRS*, then it forms a valid Groth's SNARK SRS.*

Proof. We prove the statement following VerifySRS line by line.

- Line 4 certifies that $G_{x:1} \neq [0]_1$, $G_{\alpha x:0} \neq [0]_1$, $G_{\beta x:0} \neq [0]_1$. Assume then then their values are x, α, β correspondingly.
- Line 5 ensures that (1) $G_{x:i}$ has the same exponent as $H_{x:i}$ (thus exponent of $H_{x:1}$ is x too), and that (2) exponent of $G_{x:i}$ is exponent of $G_{x:i-1}$ multiplied by x. Thus, $G_{x:i} = [x^i]_1$, and $H_{x:i} = [x^i]_2$.
- Similarly, line 6 ensures that (1) $G_{\iota x:i}$ has the same exponent as $H_{\iota x:i}$ (thus exponent of $H_{\iota x:0}$ is ι), and that (2) exponent of $G_{\iota x:i}$ is ιx^i. Therefore, the exponent of $H_{\iota x:i}$ is ιx^i too.
- Line 9 certifies that $G_\delta \neq [0]$ (thus let uss assume that its exponent is δ), and that exponent of H_δ is the same.
- Line 10 certifies that $G_{sum:i}$ is the ith x-power of $\sum_0^{n-1}(\beta u(x) + \alpha v(x) + w(x))/\delta$.
- Line 11 ensures that each $G_{t(x):i}$ is equal to $t(x)x^i/\delta$.

Therefore, SRS is in exactly the same form as in Setup presented in Fig. 4.4. □

Lemma 4.8.2. *Groth's SNARK has update completeness.*

Proof. Again, we are analysing UpdateSRS together with VerifySRS:

$\varphi = 1$ First, we will ensure that new SRS is well-formed. Line 8 first multiplies every G^{x^i} and H^{x^i} by x'^i replacing x with xx'. Next it updates each ιx^i to $\iota\iota'(xx')^i$ in $G^{\iota x^i}$ and $H^{\iota x^i}$ for $\iota \in \alpha, \beta$. Specialize merely recomputes srs_s from srs_u and its correctness is easy to verify. Thus, the new srs is well-formed. Second, the update proof is correct because for each ι: (1) on step 3.b.ii of VerifySRS the proof of knowledge created on line 3 will be correct, since it is applied to the same instance; and (2) for $i > 1$, assuming the previous update was correct, the verification equation will check that the exponent of $G_{\iota}^{(i)}$ (expected to be $\iota\iota'$) is equal to the exponent of $G_{\iota}^{(i-1)}$ (ι) multiplied by the exponent of $H_{\iota'}^{(i)}$ (ι').

$\varphi = 2$ Similarly. The SRS itself updates δ to $\delta\delta'$, and proofs are verified exactly in the same manner, but for δ instead of α, β, x.

□

4.8.2 Proofs for Update Knowledge Soundness

4.8.2.1 Proof of Lemma 4.6.4

Proof. We introduce the intermediate game $\mathcal{G}_{1/2}$, and prove the lemma in two steps, corresponding to the transitions between $\mathcal{G}_0$ and $\mathcal{G}_{1/2}$, and between $\mathcal{G}_{1/2}$ and $\mathcal{G}_1$, correspondingly. Both transitions are using security properties of the underlying Π_{dl} PoK (ZK and SE), which hold under $(1,0)$-**dlog**.

Step 1. In $\mathcal{G}_{1/2}$, we choose the critical queries, but we still update the SRS honestly. The only thing that we change is the PoK: instead of producing honest PoKs on critical queries, we simulate them. That is, we still have the update trapdoor $\hat{\iota}'$, but we use it to construct $\mathsf{x} = (\perp, G^{\hat{\iota}'}, H^{\hat{\iota}'})$, and simulate for this x. $\mathcal{G}_0$ and $\mathcal{G}_{1/2}$ are indistinguishable by perfect ZK of the PoK, thus $\mathsf{Adv}^{\mathcal{G}_0}_{\mathcal{A},\mathsf{Ext}_{\mathcal{A}}}(\lambda) \leq \mathsf{Adv}^{\mathcal{G}_{1/2}}_{\mathcal{A},\mathsf{Ext}_{\mathcal{A}}}(\lambda) + \mathsf{negl}(\lambda)$. The formal reduction breaking ZK uses $\mathcal{S}_{b,\perp,\perp,\mathsf{RO}}$ (the real prover, or the simulator) in the critical queries; every other part of the game is the same.

Step 2. Next, we recall $\mathcal{G}_1$ which, compared to $\mathcal{G}_{1/2}$, generates a fresh SRS with trapdoors $\{z_\iota\}_\iota$, and reconstructs x for PoKs in a different way. Because for critical queries we do not have the update trapdoor $\hat{\iota}$ in the clear (since we do not do the update, but pretend our fresh SRS is the outcome of the update), we extract the corresponding trapdoors $\hat{\iota}_i$ from honest and adversarial PoKs, and reconstruct $\hat{\iota}'$ from these and z_ι. Since fresh and updated trapdoors are identically distributed, this part of the transition is perfect. Similarly, our reversed computation outputs exactly the same value of the update trapdoor $\hat{\iota}'$ that the game was supposed to obtain by honest update, so instance x to PoK is the same in two games. Therefore, the only risk in the transition between the two games is that PoK extraction can fail, and in this case we abort the execution, which is noticeable by $\mathcal{A}$. But the PoK is simulation-extractable — even though $\mathcal{A}$ sees simulated PoKs already in $\mathcal{G}_{1/2}$, the probability for PoK extractor to fail is negligible by SE. Therefore, $\mathcal{G}_{1/2}$ is indistinguishable from $\mathcal{G}_1$: $\mathsf{Adv}^{\mathcal{G}_{1/2}}_{\mathcal{A},\mathsf{Ext}_\mathcal{A}}(\lambda) \le \mathsf{Adv}^{\mathcal{G}_1}_{\mathcal{A},\mathsf{Ext}_\mathcal{A}}(\lambda) + \mathsf{negl}(\lambda)$.

Technically, we need to explain two things: why we are allowed to use PoK SE here, and why it applies here, guaranteeing us extraction. First, by Theorem 4.4.1 our PoK is SE. Second, we must show that our current setting does not give $\mathcal{A}$ more power than it is considered in the SE game. Concretely, in the SE game $\mathcal{A}$ is given access to simulation oracle, RO, and two Poly oracles.

In our setting adversary also has access to RO, simulation oracle models update proofs, and other elements that adversary sees (SRS elements and non-PoK update proof elements) only depend on update trapdoors and fresh trapdoors, which are modelled with $\mathcal{O}_{\mathsf{poly}}$. The degree $d(\lambda)$ of $\mathcal{O}_{\mathsf{poly}}$ that we need is $q_1(2n-2)+q_2$. Let us recall that we defined the degree of a Laurent polynomial to be the degree of its highest degree momonial, where the degree of a monomial is the sum of absolute values of variable degrees. Given this definition, the highest degree element in the SRS is $x^{n-2}t(x)/\delta$, which has the degree $2n-1$, we obtain the degree $q_1(2n-2)+q_2$, if $\mathcal{A}$ updates a single SRS sequentially in all its queries. $\square$

4.8.2.2 Proof of Lemma 4.6.5

Proof. We will first argue why the form of $Q(\vec{\Psi})$, and concretely its proof-independent elements that are included in it ($\hat{Z}_\alpha \hat{Z}_\beta$ for instance), is as in Eq. (4.3). Consider the first phase for now. When $\mathcal{A}$ finalizes srs_u we locate in Q^*_u ($Q^* = (Q^*_u, Q^*_s)$, where $k_u := |Q^*_u|$) the critical update proofs for x, α, β — let their position be $j \in [1, k_u]$. Note that j is not equal to the $\mathcal{O}_{srs}$ *query index* i_{crit_1} since there can be many adversarial updates in Q^*_u. These update proofs are followed by a (potentially non-empty) set of adversarial proofs with indices $j+1, \ldots, k_u$ — honest proofs are not included in this suffix since critical proofs are the last honest ones in Q^*_u. Now, let us argue that the element $G_{\alpha x:0}$ in the final SRS corresponds to $Z_\alpha \prod \alpha_i^\mathcal{A}$, where $\alpha_i^\mathcal{A}$ are adversarial update trapdoors. In step 3.b.iii of SRS verification we do a cascade verification: in particular, on the $j+1$ step we check $\hat{e}(G_\alpha^{(j+1)}, H) = \hat{e}(G_\alpha^{(j)}, H_{\alpha'}^{(j+1)})$. First of all, the form of $G_\alpha^{(j)}$ is exactly G^{z_α}, since we assume that the proof number j, which consists of $\rho_{\alpha'}^{(j)} = (G_\alpha^{(j)}, G_{\alpha'}^{(j)}, H_{\alpha'}^{(j)}, \pi_{\alpha'}^{(j)})$, is the last honestly generated one. And since we assuming **lucky**, we know that the first tuple element of $\rho_{\alpha'}^{(j)}$ is exactly

G^{z_α} (and other three are simulated; see Fig. 4.9). So, if the exponent of $H_{\alpha'}^{(i)}$ is some $\alpha_j^{\mathcal{A}}$, and $G_\alpha^{(j)} = G^\alpha$, then we know after this loop ends that $G_\alpha^{(k_u)} = G^{z_\alpha \prod \alpha_i^{\mathcal{A}}}$. Finally, from line 4 of verification procedure it follows from $G_{\alpha x:0} = G_\alpha^{(k_u)}$. Same logic applies to $G_{x:1}, G_{\beta x:0}$. The next step is to use other VerifySRS equations, similarly to the style in Lemma 4.8.1, to show that every α related slot in the final SRS contains $z_\alpha \prod \alpha_i^{\mathcal{A}}$ (in other words, srs is consistent w.r.t. this value of α trapdoor). We can show similar form and prove consistency similarly for the second phase and δ slot being taken by $z_\delta \prod \delta_i^{\mathcal{A}}$, and srs_s being consistent w.r.t. this value. This argument explains the form of the proof-independent part of $Q(\vec{\Psi})$:

$$\left\{ \hat{Z}_\alpha \hat{Z}_\beta, \sum_{i=0}^{\ell} \mathsf{a}_i(\hat{Z}_\beta u_i(\hat{Z}_x) + \hat{Z}_\alpha v_i(\hat{Z}_x) + w_i(\hat{Z}_x)), \hat{Z}_\delta \right\} \tag{4.4}$$

One technical detail is that the statement of the current lemma suggests that executions of $\mathcal{G}_1$ possess a certain property (i.e. reconstructing the verification equation is represented by $Q(\vec{\Psi})$). But what really takes place is rather an AGM reduction: what we want to show is that the SRS elements $\mathcal{A}$ finalizes have such-and-such form. This we do, based on the verification equations that are used inside VerifySRS, and we base our reasoning critically on the assumption that $\mathcal{A}$ does not break the discrete logarithm. So the lemma, in fact, implies a simple game transition, and a reduction to the dlog (similar to the one that will be shown later in Lemma 4.6.7).

What we also need to show to proceed, is that the exponents $\alpha_i^{\mathcal{A}}$ (of elements $H_{\iota'}^{(i)}$, that take third tuple-place in adversarial update proofs) should be constants. Even though we can extract these values from PoKs, it is important that $H_{\iota'}^{(i)}$ are not constructed as a non-constant linear combination of elements $\mathcal{A}$ has seen; that is, we must have that the AGM coefficient matrix $\mathcal{A}$ returns together with $H_\iota^{(i)}$ has $\alpha_i^{\mathcal{A}}$ as the only non-zero coefficient, associated with the constant slot. This is guaranteed by the implicit reduction: the logic is very similar to the proof of Theorem 4.4.1, where we showed that $\mathcal{A}$ can either return a simulated proof, or create a honest one from a constant. Since $\mathcal{A}$ cannot reuse honest proofs (this is guaranteed by VerifySRS), the only option for it is to create $H_{\iota'}^{(i)}$ honestly (as constants).

Finally, to explain why $Q(\vec{\Psi})$ is a Laurent polynomial in $\vec{\Psi}_2$, it is enough to understand three things. First, the elements E that $\mathcal{O}_{srs}$ outputs on the critical queries are Laurent polynomials in $\vec{\Psi}_2$ — this can be verified by observing that the form of honest SRS consists of Laurent polynomials in its trapdoors. Second, no new elements depending on $\vec{\Psi}_2$ can be obtained by passing E into RO, since RO returns randomly sampled values that are independent of $\vec{\Psi}_2$. Third, $\mathsf{UpdateSRS}$ of SRS does not use any older trapdoors, and only introduces new ones: this means that for any set of elements E' (that are Laurent polynomials in $\vec{\Psi}_2$) being inputs of $\mathsf{UpdateSRS}$, it will merely produce linear combinations of E', which will be again Laurent in $\vec{\Psi}_2$. □

4.8.2.3 Proof of Lemma 4.6.6

Proof. Assume $\mathsf{Verify}(\mathsf{srs}, \mathsf{x}, \pi) = 1$, the event **lucky** happens since otherwise $\mathcal{A}$ cannot win $\mathcal{G}_2$. Because **bad** did not happen, we deduce that $Q(\psi_1, \vec{\Psi}_2) \equiv 0$ w.o.p., where $Q(\vec{\Psi})$ is as in the equation Eq. (4.3).

The problem is that we do not know the form of Q; we want to argue that if $Q(\psi_1, \vec{\Psi}_2) \equiv 0$ then AGM coefficients that $\mathcal{A}$ returns have some specific form, and contain witness wires. But we also do not know what is the most general form of Q — with AGM coefficients being treated as variables, and not as concrete values. For our proof to proceed in such generality, we will only care about those AGM base elements that depend on $\vec{\Psi}_2$ — all the other elements are considered constants in $Q(\psi_1, \vec{\Psi}_2)$. Now, we must determine which elements depend on $\vec{\Psi}_2$.

Observation 3. *Let E_1, E_2 be elements depending on $\vec{\Psi}_2$ that $\mathcal{A}$ sees as an output of critical queries in the first and second round correspondingly. Then, the proof elements A, B, C can only include these elements and linear coefficients of $E_1 \cup E_2$ with constant values potentially unknown to $\mathcal{A}$.*

1. In the first phase, $\{Z_x, Z_\alpha, Z_\beta\} \subset \vec{\Psi}_2$ appear in the update query number i_{crit_1}: in SRS elements and in the corresponding update proof, let us call these elements E_1. Now, since i_{crit_1} does not have to be the last query of the first round, nothing stops $\mathcal{A}$ from passing E_1 into other RO queries or update oracle queries (and not using them for final SRS). Passing these values into RO is generally useful both here and in the second phase: on any request $\mathcal{A}$ will receive an unrelated constant value, so no elements that depend on E_1 can be produced in such a way. Passing E_1 into SRS update oracle only mixes E_1 with some other values that are considered

constants over $\vec{\Psi}_2$. This is easy to see: $\mathsf{UpdateSRS}$ procedure is designed in such a way that no knowledge of internal SRS trapdoors in needed to perform the update. As a result, all output elements of $\mathsf{UpdateSRS}$ are of form $[k_0 + \sum k_i e]$, where $e \in E_1$, and k_i are constants (e.g. update trapdoors). This is equivalent to $\mathcal{A}$ producing the linear combination of E_1 elements on its own, but in this case k_i may not be known to $\mathcal{A}$. Therefore, in the first round, until $\mathcal{A}$ finalizes, it only sees E_1 and linear combinations of E_1 elements (with unknown coefficients potentially).

2. The same logic applies to the adversarial queries w.r.t. E_1 in the second round before the second round critical queries.

3. In the second round query i_{crit_2} adversary obtains elements that depend on $E_2 = \{Z_\delta\} \subset \vec{\Psi}_2$: second phase SRS elements and corresponding update proofs. Now, similarly, $\mathcal{A}$ cannot mix E_1 with E_2 (and within these sets) using update oracle, producing conceptually new elements that depend on E_2 and cannot be represented as linear combinations of E_1 and E_2 elements.

4. The second round ends and $\mathcal{A}$ submits the final SRS. It then can query RO (since update oracle is disabled after the second round finalization), and finally $\mathcal{A}$ submits the instance and the proof.

□

Then we can assume A, B, C to only contain linear combinations of both E_i, and some other constant values. The form of this constant value may be complex, since it is a linear (AGM) combination of constants, the form of which depends on the particular execution, interaction pattern and other things. Nevertheless, these values are constant factor in $Q(\psi_1, \vec{\Psi}_2)$. As we just argued, elements that depend on E_i and that are not direct outputs of update oracle on two critical queries are linear combinations $[\sum k_i e_i]_\iota$. So since these are in the span of $E_1 \cup E_2$, we will only consider A, B, C to consist of linear elements $E_1 \cup E_2$ and constant values.

We now formally state the list of elements that can be used in the algebraic base of A, B, C. We use a custom enumeration to simplify our notation.

$$A(\vec{\Psi}_2) = A_0 + \sum_{i=1}^{2n-2} A_{1:i} Z_x^i + \sum_{i=0}^{n-1} (A_{2:i} Z_\alpha Z_x^i + A_{3:i} Z_\beta Z_x^i) + A_4 Z_\delta$$

$$
\begin{aligned}
&+ \sum_{i=l+1}^{m} A_{5:i} \frac{\hat{Z}_\beta u_i(\hat{Z}_x) + \hat{Z}_\alpha v_i(\hat{Z}_x) + w_i(\hat{Z}_x)}{Z_\delta} + \sum_{i=0}^{n-2} A_{6:i} \frac{\hat{Z}_x^i t(\hat{Z}_x)}{Z_\delta} \\
&+ \sum_{\iota} (A_{7:\iota} \frac{Z_\iota}{(\prod_{\mathcal{I}_1} T_{i,\iota})(\prod_{\mathcal{I}_2} \iota_i^{\mathcal{A}})} + A_{8:\iota} \frac{K_\iota Z_\iota}{(\prod_{\mathcal{I}_1} T_{i,\iota})(\prod_{\mathcal{I}_2} \iota_i^{\mathcal{A}})})
\end{aligned}
$$

$$
\begin{aligned}
B(\vec{\Psi}_2) = B_0 &+ \sum_{i=1}^{2n-2} B_{1:i} Z_x^i + \sum_{i=0}^{n-1} (B_{2:i} Z_\alpha Z_x^i + B_{3:i} Z_\beta Z_x^i) + B_4 Z_\delta \\
&+ \sum_{i,\iota} (B_{7:\iota} \frac{Z_\iota}{(\prod_{\mathcal{I}_1} T_{i,\iota})(\prod_{\mathcal{I}_2} \iota_i^{\mathcal{A}})})
\end{aligned}
$$

C is constructed as A. The constant value G sometimes corresponds to x^0 and could be referred to as $A_{1:0}$, but we will give the coefficient a separate index 0 for clarity. Indices 1 to 6 correspond to outputs of critical queries. Elements number 7 are second and third elements of proof of update: they contain update trapdoors as exponents. Elements number 8 are corresponding PoKs. In both these last two types of elements the denominator contains some honest and adversarial trapdoors corresponding to the prefix of the update procedure before the critical query: these are the elements that are extracted in $\mathsf{SimUpdProof}$ of $\mathcal{G}_1$. Essentially, we divide the new trapdoor by the old one to reconstruct the update trapdoor (for the update the challenger did not do).

We can immediately simplify the representation even further: observe that elements number 10 and 11 already exist in the span of elements they are included into. For example, $A_{10:\iota} Z_\iota / (\prod_{\mathcal{I}_1} T_{i,\iota} \prod_{\mathcal{I}_2} \iota_i^{\mathcal{A}})$ is just Z_ι multiplied by a very specific constant that $\mathcal{A}$ knows only partially (because T_i is hidden from it). For $\iota = x$, there exists $A_{1:1}$, for $\iota = \alpha, \beta$ there exist, correspondingly, $A_{2:0}$ and $A_{3:0}$. Therefore, the coefficient of Z_x is now $A_{1:1} + A_{10:\iota} / (\prod_{\mathcal{I}_1} T_{i,\iota} \prod_{\mathcal{I}_2} \iota_i^{\mathcal{A}})$. It is more restrictive for $\mathcal{A}$ to use constants which it knows only partially, therefore without loss of generality we can assume that $A_{10:\iota} = 0$, and if adversary wants to include Z_x it will set $A_{1:1}$ to a nonzero value. Similarly, $A_{11:\iota} = B_{10:\iota} = 0$.

Which leads to the general form similar to the one we have in the original proof of Groth16 in [Bowe et al., 2017b], except our elements have extra adversarial trapdoors (hidden inside some variables with hats):

$$
\begin{aligned}
A(\vec{\Psi}_2) = A_0 &+ \sum_{i=1}^{2n-2} A_{1:i} Z_x^i + \sum_{i=0}^{n-1} (A_{2:i} Z_\alpha Z_x^i + A_{3:i} Z_\beta Z_x^i) + A_4 Z_\delta \\
&+ \sum_{i=l+1}^{m} A_{5:i} \frac{\hat{Z}_\beta u_i(\hat{Z}_x) + \hat{Z}_\alpha v_i(\hat{Z}_x) + w_i(\hat{Z}_x)}{Z_\delta} + \sum_{i=0}^{n-2} A_{6:i} \frac{\hat{Z}_x^i t(\hat{Z}_x)}{Z_\delta}
\end{aligned}
$$

$$B(\vec{\Psi}_2) = B_0 + \sum_{i=1}^{2n-2} B_{1:i} Z_x^i + \sum_{i=0}^{n-1} (B_{2:i} Z_\alpha Z_x^i + B_{3:i} Z_\beta Z_x^i) + B_4 Z_\delta$$

We follow a proof strategy similar to the one in [Bowe et al., 2017b]. One structural difference is that we will not try to deduce first which elements can be included into A, B, C and which can not — since we do not know whether this will be necessary for the result. Instead, we will start from the end, immediately locating the three critical equations from which we expect to extract — these are equations that correspond to the monomials of public verification equation elements. The corresponding monomials are: $Z_X^i, Z_\alpha Z_x^i, Z_\beta Z_x^i$. For $Z_\alpha Z_x^i$:

$$\begin{aligned}(\sum A_{2,i} Z_\alpha Z_x^i)(B_0 + \sum B_{1,i} Z_x^i) + (\sum A_{5,i} \hat{Z}_\alpha v_i(\hat{Z}_x)) B_4 + \\ (\sum B_{2,i} Z_\alpha Z_x^i)(A_0 + \sum A_{1,i} Z_x^i) - \sum \mathsf{a}_i \hat{Z}_\alpha v_i(\hat{Z}_x) - (\sum C_{5,i} \hat{Z}_\alpha v_i(\hat{Z}_x)) = 0\end{aligned}$$

For $Z_\beta Z_x^i$:

$$\begin{aligned}(\sum A_{3,i} Z_\beta Z_x^i)(B_0 + \sum B_{1,i} Z_x^i) + (\sum A_{5,i} \hat{Z}_\beta u_i(\hat{Z}_x)) B_4 + \\ (\sum B_{3,i} Z_\beta Z_x^i)(A_0 + \sum A_{1,i} Z_x^i) - \sum \mathsf{a}_i \hat{Z}_\beta u_i(\hat{Z}_x) - (\sum C_{5,i} \hat{Z}_\beta u_i(\hat{Z}_x)) = 0\end{aligned}$$

And for Z_x^i:

$$\begin{aligned}(B_0 + \sum B_{1,i} Z_x^i)(A_0 + \sum A_{1,i} Z_x^i) + (\sum A_{5,i} w_i(\hat{Z}_x) + \sum A_{6,i} \hat{Z}_x^i t(\hat{Z}_x)) B_4 - \\ \sum \mathsf{a}_i w_i(\hat{Z}_x) - \sum C_{5,i} w_i(\hat{Z}_x) - \sum C_{6,i} \hat{Z}_x^i t(\hat{Z}_x) = 0\end{aligned}$$

Our strategy now is to attempt to remove the elements which clutter these equations and prevent us from substituting the first two into the third one to obtain a QAP. Let us write out equations on monomials that include Z_α, Z_β, Z_x and see whether we can deduce any simplifying relations on the AGM coefficients involved.

$$Z_\alpha^2 Z_x^i : (\sum_{i=0}^{n-1} A_{2:i} Z_\alpha Z_x^i)(\sum_{i=0}^{n-1} B_{2:i} Z_\alpha Z_x^i) = 0 \implies$$

$$\forall i \in [0, 2n-2] : \sum_{j,k:(0,0);j+k=i}^{(n-1,n-1)} A_{2:j} B_{2:k} = 0,$$

$$Z_\beta^2 Z_x^i : (\sum_{i=0}^{n-1} A_{3:i} Z_\beta Z_x^i)(\sum_{i=0}^{n-1} B_{3:i} Z_\beta Z_x^i) = 0 \implies$$

$$\forall i \in [0, 2n-2] : \sum_{j,k:(0,0);j+k=i}^{(n-1,n-1)} A_{3:j} B_{3:k} = 0,$$

$$
\begin{aligned}
Z_\alpha Z_\beta Z_x^i &: (\sum_{i=0}^{n-1} A_{2:i} Z_\alpha Z_x^i)(\sum_{i=0}^{n-1} B_{3:i} Z_\beta Z_x^i) + (\sum_{i=0}^{n-1} A_{3:i} Z_\beta Z_x^i)(\sum_{i=0}^{n-1} B_{2:i} Z_\alpha Z_x^i) = \\
&\quad \alpha^{\mathcal{A}}\beta^{\mathcal{A}} (\neq 0), \\
Z_\alpha^2 Z_x^i / Z_\delta &: (\sum_{i=l+1}^{m} A_{5,i} \hat{Z}_\alpha v_i(\hat{Z}_x)/Z_\delta)(\sum_{i=0}^{n-1} B_{2,i} Z_\alpha Z_x^i) = 0, \\
Z_\beta^2 Z_x^i / Z_\delta &: (\sum_{i=l+1}^{m} A_{5,i} \hat{Z}_\beta u_i(\hat{Z}_x)/Z_\delta)(\sum_{i=0}^{n-1} B_{3,i} Z_\beta Z_x^i) = 0, \\
Z_\alpha Z_\beta Z_x^i / Z_\delta &: (\sum_{i=l+1}^{m} A_{5,i} \hat{Z}_\alpha v_i(\hat{Z}_x)/Z_\delta)(\sum_{i=0}^{n-1} B_{3,i} Z_\beta Z_x^i) + \\
&\qquad (\sum_{i=l+1}^{m} A_{5,i} \hat{Z}_\beta u_i(\hat{Z}_x)/Z_\delta)(\sum_{i=0}^{n-1} B_{2,i} Z_\alpha Z_x^i) = 0, \\
Z_\alpha Z_x^i / Z_\delta &: (\sum_{i=l+1}^{m} A_{5,i} \hat{Z}_\alpha v_i(\hat{Z}_x)/Z_\delta)(\sum_{i=0}^{2n-2} B_{1,i} Z_x^i) + \\
&\qquad (\sum_{i=l+1}^{m} A_{5,i} w_i(\hat{Z}_x)/Z_\delta + \sum_{i=0}^{n-2} A_{6,i} \hat{Z}_x^{\,i} t(\hat{Z}_x)/Z_\delta)(\sum_{i=0}^{n-1} B_{2,i} Z_\alpha Z_x^i) = 0, \\
Z_\beta Z_x^i / Z_\delta &: (\sum_{i=l+1}^{m} A_{5,i} \hat{Z}_\beta u_i(\hat{Z}_x)/Z_\delta)(\sum_{i=0}^{2n-2} B_{1,i} Z_x^i) + \\
&\qquad (\sum_{i=l+1}^{m} A_{5,i} w_i(\hat{Z}_x)/Z_\delta + \sum_{i=0}^{n-2} A_{6,i} \hat{Z}_x^{\,i} t(\hat{Z}_x)/Z_\delta)(\sum_{i=0}^{n-1} B_{3,i} Z_\beta Z_x^i) = 0.
\end{aligned}
$$

From the first equation, $Z_\alpha^2 Z_x^i$, we have $A_2 * B_2 = 0$, where $*$ denotes convolution product. From $Z_\beta^2 Z_x^i$, $A_3 * B_3 = 0$. From $Z_\alpha Z_\beta Z_x^i$, $A_2 * B_3 + A_3 * B_2 = (\alpha^{\mathcal{A}}\beta^{\mathcal{A}}, 0, \ldots, 0)^T$.

Convolution products have a property useful in this context which we explain now. Assume $a * b = 0$, then $a_0 b_0 = 0$, $a_1 b_0 + a_0 b_1 = 0$, $a_2 b_0 + a_1 b_1 + a_0 b_2 = 0$ and so on (the longest equation is for degree n, and then the number of elements decreases one-by-one until degree $2n$). It is easy to see that the product is symmetric: $a * b = b * a$. Importantly, if $a_0 \neq 0$, then all $b_i = 0$: from the first equation $b_0 = 0$, from the second equation $a_0 b_1 = 0$, so $b_1 = 0$ too, from the third equation similarly $a_0 b_2 = 0$ (the other two terms cancel because of $b_0 = b_1 = 0$), and thus $b_2 = 0$. This process is continued until the degree n (middle, longest) equation. Therefore, if $a * b = 0$, then $a_0 \neq 0 \implies b = 0$, or $b_0 \neq 0 \implies a = 0$.

In our case, the $Z_\alpha Z_\beta Z_x^i$ gives $A_{2:0} B_{3:0} + A_{3:0} B_{2:0} = \alpha^{\mathcal{A}}\beta^{\mathcal{A}}$. But at the same time, at least one from $\{A_{2:0}, B_{2:0}\}$ and $\{A_{3:0}, B_{3:0}\}$ must be zero. If both zero values are in both terms, it is impossible for their sum to be zero, therefore both zero values must be in one term. This leads us to the two options:

(a) $A_{2:0} = B_{3:0} = 0$ and both $A_{3:0}$ and $B_{2:0}$ are nonzero. From this, by the convolution property above, we immediately conclude $\forall i.\, A_{2:i} = B_{3:i} = 0$.

(b) Symmetrically, $A_{3:i} = B_{2:i} = 0$ for all i, but $A_{2:0}$ and $B_{3:0}$ are nonzero.

In the honest proof generation, $\beta \in B$, as in option (b), so let us assume option (a) first. We will later see that one can indeed construct a proof with B swapped with A; we will succeed with (a), so this choice is performed without loss of generality.

Now, the equation $Z_\alpha Z_\beta Z_x^i$ becomes $(\sum_{i=0}^{n-1} A_{3:i} Z_\beta Z_x^i)(\sum_{i=0}^{n-1} B_{2:i} Z_\alpha Z_x^i) = \alpha^{\mathcal{A}}\beta^{\mathcal{A}} \neq 0$ or $A_3 * B_2 = (\alpha^{\mathcal{A}}\beta^{\mathcal{A}}, 0 \ldots 0)^T$. By an argument similar to above we can argue that $A_{3,i} = B_{2,i} = 0$ for all $i > 0$. We examine the highest degree coefficient $A_{3,n}B_{2,n} = 0$, and assume $A_{3,n} \neq 0$ wlog, then $B_{2,n} = 0$. Then, from the previous equation $A_{3,n-1}B_{2,n} + A_{3,n}B_{2,n-1} = 0$ we derive $B_{2,n-1} = 0$. This process goes on until on the degree n equation $A_{3,0}B_{2,n} + \ldots + A_{3,n-1}B_{2,1} + A_{3,n}B_{2,0} = 0$ where we reach a contradiction since $B_{2,0} = 0$ but we assumed it is not. By a symmetric argument, $B_{2,n} \neq 0$ lead to $A_{3,0} = 0$ and contradiction too. So $B_{2,n} = A_{3,n} = 0$. The equation $2n-1$ is now immediately satisfied, but the equation for $2n-2$ becomes $A_{3,n-1}B_{2,n-1} = 0$. Here the proof idea repeats, but we reach contradiction on degree $n-1$ equation instead. Using this process we conclude that $A_{3,i} = B_{2,i} = 0$ for $i > 0$.

If $\forall i.\, A_{2:i} = B_{3:i} = 0$, $A_{3:0}B_{2:0} = \alpha^{\mathcal{A}}\beta^{\mathcal{A}}$, and $A_{3:i} = B_{2:i} = 0$ for $i > 0$, our system of equation becomes:

$$
\begin{aligned}
Z_\alpha Z_\beta Z_x^i &: A_{3:0}B_{2:0} = 1 \\
Z_\alpha^2 Z_x^i / Z_\delta &: (\sum_{i=l+1}^{m} A_{5,i}\hat{Z}_\alpha v_i(\hat{Z}_x)/Z_\delta) B_{2,0} Z_\alpha = 0 \\
Z_\alpha Z_\beta Z_x^i / Z_\delta &: (\sum_{i=l+1}^{m} A_{5,i}\hat{Z}_\beta u_i(\hat{Z}_x)/Z_\delta) B_{2,0} Z_\alpha = 0 \\
Z_\alpha Z_x^i / Z_\delta &: (\sum_{i=l+1}^{m} A_{5,i}\hat{Z}_\alpha v_i(\hat{Z}_x)/Z_\delta)(\sum_{i=0}^{2n-2} B_{1,i} Z_x^i) + \\
&\qquad (\sum_{i=l+1}^{m} A_{5,i} w_i(\hat{Z}_x)/Z_\delta + \sum_{i=0}^{n-2} A_{6,i}\hat{Z}_x{}^i t(\hat{Z}_x)/Z_\delta) B_{2,0} Z_\alpha = 0
\end{aligned}
$$

$$Z_\beta Z_x^i/Z_\delta : (\sum_{i=l+1}^{m} A_{5,i}\hat{Z}_\beta u_i(\hat{Z}_x)/Z_\delta)(\sum_{i=0}^{2n-2} B_{1,i}Z_x^i) = 0$$

The equations $Z_\alpha^2 Z_x^i$, $Z_\beta^2 Z_x^i$, $Z_\beta^2 Z_x^i/Z_\delta$ are now satisfied, so are not considered anymore. From $Z_\alpha^2 Z_x^i/Z_\delta$ we conclude that $\sum_{i=l+1}^{m} A_{5,i}v_i(\hat{Z}_x) = 0$ as a polynomial in Z_x, and same for $(\sum_{i=l+1}^{m} A_{5,i}u_i(\hat{Z}_x) = 0$. $Z_\alpha Z_x^i/Z_\delta$ reduces to

$$(\sum_{i=l+1}^{m} A_{5,i}w_i(\hat{Z}_x)/Z_\delta + \sum_{i=0}^{n-2} A_{6,i}\hat{Z}_x{}^i t(\hat{Z}_x)/Z_\delta)B_{2,0}Z_\alpha = 0$$

from which, since these two sets are of different powers, we conclude

$$\sum_{i=l+1}^{m} A_{5,i}w_i(\hat{Z}_x) = 0 \text{ and } \sum_{i=0}^{n-2} A_{6,i}\hat{Z}_x{}^i t(\hat{Z}_x) = 0$$

both as polynomials in Z_x.

We now return to the three critical equations which are now significantly simplified:

$$\begin{aligned}
Z_\alpha Z_x^i &: B_{2,0}(A_0 + \sum A_{1,i}Z_x^i) = \sum \mathsf{a}_i\alpha^{\mathcal{A}}v_i(\hat{Z}_x) + (\sum C_{5,i}\alpha^{\mathcal{A}}v_i(\hat{Z}_x)) \\
Z_\beta Z_x^i &: A_{3,0}(B_0 + \sum B_{1,i}Z_x^i) = \sum \mathsf{a}_i\beta^{\mathcal{A}}u_i(\hat{Z}_x) + (\sum C_{5,i}\beta^{\mathcal{A}}u_i(\hat{Z}_x)) \\
Z_x^i &: (B_0 + \sum B_{1,i}Z_x^i)(A_0 + \sum A_{1,i}Z_x^i) = \sum \mathsf{a}_i w_i(\hat{Z}_x) + \\
&\sum C_{5,i}w_i(\hat{Z}_x) + \sum C_{6,i}\hat{Z}_x^i t(\hat{Z}_x)
\end{aligned}$$

Express 1 and 2 and substitute into 3:

$$\begin{aligned}
\frac{\beta^{\mathcal{A}}\alpha^{\mathcal{A}}}{A_{3,0}B_{2,0}}(\sum_{i=0}^{l} \mathsf{a}_i u_i(\hat{Z}_x) + \sum_{i=l+1}^{m} C_{5,i}u_i(\hat{Z}_x))(\sum_{i=0}^{l} \mathsf{a}_i v_i(\hat{Z}_x) + \sum_{i=l+1}^{m} C_{5,i}v_i(\hat{Z}_x)) = \\
\sum_{i=0}^{l} \mathsf{a}_i w_i(\hat{Z}_x) + \sum_{i=l+1}^{m} C_{5,i}w_i(\hat{Z}_x) + \sum_{i=0}^{n-2} C_{6,i}\hat{Z}_x^i t(\hat{Z}_x)
\end{aligned}$$

$A_{3,0}B_{2,0} = \beta^{\mathcal{A}}\alpha^{\mathcal{A}}$, so the first term is equal to 1. Our result is a QAP in $\hat{Z}_x$: $C_{5,i}$ elements are witness wires, and $C_{6,i}$ are coefficients of $h(\hat{Z}_x)$ (such that $h(\hat{Z}_x)t(\hat{Z}_x)$ is equal to QAP left hand side). Therefore the extractor targeting $C_{5,i}$ succeeds in extracting the witness. □

4.8.2.4 Proof of Lemma 4.6.7

Proof. Recall that we denote $\vec{\Psi}_2 = \{Z_\iota\}_\iota$; similarly, let us say $\vec{\psi}_2 = \{z_\iota\}_\iota$. Let us define $Q_2(\vec{\Psi}_2) := Q(\psi_1, \vec{\Psi}_2) \not\equiv 0$. Also recall that **bad** implies **lucky**, so we are implicitly considering lucky traces in this lemma.

$\underline{\mathcal{B}(\{G^{z^i}\}_{i=1}^{2n-1}, \{H^{z^i}\}_{i=1}^{2n-2}, r_\delta, s_\delta, G^{\frac{1}{r_\delta z+s_\delta}}, H^{\frac{1}{r_\delta z+s_\delta}}):}$

Initialize $\mathsf{RO}_t(\cdot)$
$\{r_\iota, s_\iota\}_{\iota\in\{x,\alpha,\beta\}} \xleftarrow{\$} \mathbb{Z}_p$
Set *implicitly* $z_\iota \leftarrow r_\iota z + s_\iota$ for critical query embeddings for $\iota \in \{\alpha, \beta, x\}$
Similarly set $z_\delta \leftarrow \frac{1}{r_\delta z+s_\delta}$
Run $\mathcal{A}$ and Ext as in $\mathcal{G}_1$ using dlog challenge elements to embed z_ι
    into critical SRS updates, and modified $\mathsf{SimUpdProof}_{\mathcal{B}}$
assert $\mathsf{Verify}(\mathsf{srs}, \mathsf{x}, \pi) = 1 \wedge (\mathsf{x}, \mathsf{w}) \notin \mathcal{R}$
Reconstruct $Q(\vec{\psi}_1, \vec{\Psi}_2)$ using AGM matrix T and extracted trapdoors
    from srs PoKs
Reinterpret it as $Q'(Z)$; factor $Q'(Z)$, find z among the roots
return z

$\underline{\mathsf{SimUpdProof}_{\mathcal{B}}(\iota, \varphi):}$

Compute $G^{\hat{\iota}'}, H^{\hat{\iota}'}$ as before, except now we do not know exp of G^{z_ι}, H^{z_ι}
`% Notice: for` δ, $G^{\hat{\iota}'} = (G^{\frac{1}{r_\delta z+s_\delta}})^{\hat{\iota}^{-1}}$ `due to inverted embedding`
As in $\mathsf{SimUpdProof}$, create x and call $\mathsf{Sim}_{\mathsf{dl}}^{\mathsf{RO}_1(\cdot)}$ on it to obtain $\pi_{\iota'}$
return $(G^{z_\iota}, G^{\hat{\iota}'}, H^{\hat{\iota}'}, \pi_{\iota'})$

Figure 4.12: Adversary $\mathcal{B}$ against $(2n-1, 2n-2)$-extended dlog assumption in Theorem 4.6.3. It is parameterized by a full update knowledge soundness algebraic adversary $\mathcal{A}$, and the extractor $\mathsf{Ext}_{\mathcal{A}}$ as in Fig. 4.7. Its main task is to simulate $\mathcal{G}_1$ to $\mathcal{A}$, embedding the edlog instance z into SRS on critical queries.

Let $\mathcal{A}$ be a PPT adversary in $\mathcal{G}_2$. We want to show that it is computationally hard for $\mathcal{A}$ to come up with a non-zero polynomial Q_2 such that the verifier accepts, i.e. $Q_2(\vec{\psi_2}) = 0$. The idea of the proof is to construct an adversary $\mathcal{B}$ that simulates $\mathcal{G}_2$ for $\mathcal{A}$ and embeds $(2n-1, 2n-2)$-**edlog** challenge z into the update trapdoors z_ι ($\vec{\psi_2}$) at critical queries i_{crit_1} and i_{crit_2}. We show $Q_2(\vec{\Psi}) \not\equiv 0$ implies that a closely related univariate polynomial $Q'(Z) \not\equiv 0$ where $(2n-1, 2n-2)$-**edlog** challenge value z is one of the roots of Q'. Since Q' is a univariate polynomial, $\mathcal{B}$ can efficiently factor it and output z. It follows that $Q_2(\vec{\psi_2}) = 0$ and $Q_2(\vec{\Psi}) \not\equiv 0$ can only hold with negligible probability, thus event **bad** is negligibly rare.

We now explain in detail the embedding strategy of $\mathcal{B}$ in Fig. 4.12. Firstly, $\mathcal{B}$ obtains as a challenge $(\mathsf{bp}, \{G^{z^i}\}_{i=1}^{2n-1}, \{H^{z^i}\}_{i=1}^{2n-2}, r, s, G^{\frac{1}{rz+s}}, H^{\frac{1}{rz+s}})$. Instead of sampling critical trapdoor values z_ι randomly, we implicitly define $z_\iota := r_\iota z + s_\iota$ for $\iota \in \{x, \alpha, \beta\}$ and let $\mathcal{B}$ sample s_ι, r_ι randomly.

When $\mathcal{A}$ requests an update number i_{crit_1} in the first phase, $\mathcal{B}$ uses the challenge input and $(r_x, r_\alpha, r_\beta, s_x, s_\alpha, s_\beta)$ to set

$$\mathsf{srs}'_u = \begin{pmatrix} \{G^{(r_x z+s_x)^i}, H^{(r_x z+s_x)^i}\}_{i=0}^{2n-2}, \{G^{(r_\alpha z+s_\alpha)(r_x z+s_x)^i}, G^{(r_\beta z+s_\beta)(r_x z+s_x)^i}, \\ H^{(r_\alpha z+s_\alpha)(r_x z+s_x)^i}, H^{(r_\beta z+s_\beta)(r_x z+s_x)^i}\}_{i=0}^{n-1} \end{pmatrix}.$$

Similary $\mathsf{SimUpdProof}$ is computed exactly as in $\mathcal{G}_2$ except that $\mathcal{B}$ knows G^{z_ι} and H^{z_ι} instead of $z_\iota = r_\iota z + s_\iota$ itself.

When $\mathcal{A}$ finalizes the first phase 1, $\mathcal{B}$ sees the verifying proofs $(\pi^{\mathcal{A}}_{1:1}, \dots, \pi^{\mathcal{A}}_{1:t_1})$ for all updates after the last update query that $\mathcal{A}$ made. More precisely, $\mathcal{B}$ also receives other verifying proofs, corresponding to the previous honest updates and adversarial updates between them, but $\mathcal{B}$ can just discard them after verifying their validity, keeping only the last t_1 of them. Then $\mathcal{B}$ can extract $(\vec{\alpha^{\mathcal{A}}}, \vec{\beta^{\mathcal{A}}}, \vec{x^{\mathcal{A}}})$ such that

$$\mathsf{srs}_u = \begin{pmatrix} \{G^{\left((r_x z+s_x)\prod_j x^{\mathcal{A}}_j\right)^i}, H^{\left((r_x z+s_x)\prod_j x^{\mathcal{A}}_j\right)^i}\}_{i=0}^{2n-2}, \\ \{G^{\left((r_\alpha z+s_\alpha)\prod_j \alpha^{\mathcal{A}}_j\right)\left((r_x z+s_x)\prod_j x^{\mathcal{A}}_j\right)^i}, G^{\left((r_\beta z+s_\beta)\prod_j \beta^{\mathcal{A}}_j\right)\left((r_x z+s_x)\prod_j x^{\mathcal{A}}_j\right)^i}, \\ H^{\left((r_\alpha z+s_\alpha)\prod_j \alpha^{\mathcal{A}}_j\right)\left((r_x z+s_x)\prod_j x^{\mathcal{A}}_j\right)^i}, H^{\left((r_\beta z+s_\beta)\prod_j \beta^{\mathcal{A}}_j\right)\left((r_x z+s_x)\prod_j x^{\mathcal{A}}_j\right)^i}\}_{i=0}^{n-1} \end{pmatrix}.$$

where $j = 1, \dots, t_1$. The reasoning of why the form of srs_u is that is similar to Lemma 4.6.5: because the critical queries are guessed correctly, $\mathcal{A}$ can only add its own adversarial trapdoors, but not to change the general form of the last honest SRS elements. To simplify the notation, we, as before, us polynomials $Z_x(Z) = (r_x Z + s_x)\prod_j x^{\mathcal{A}}_j$ and $Z_\alpha(Z) = (r_\alpha Z + s_\alpha)\prod_j \alpha^{\mathcal{A}}_j$ and $Z_\beta(Z) = (r_\beta Z + s_\beta)\prod_j \beta^{\mathcal{A}}_j$.

The variable Z stands for the edlog challenge exponent z. We note that extraction of $(\vec{\alpha^{\mathcal{A}}}, \vec{\beta^{\mathcal{A}}}, \vec{x^{\mathcal{A}}})$ above is possible only due to the strong form of simulation extractability that we proved for Π_{dl} (under $(1,0)$-**dlog**, which is clearly implied by $(2n-1, 2n-2)$-**edlog**). Namely, in our scenario, $\mathcal{A}$ sees both honest and simulated proofs from $\mathcal{B}$ and also gets group-based auxiliary inputs that the strong simulation extractability modelled by $\mathcal{O}^{\mathbb{G}_1}_{\mathsf{poly}}, \mathcal{O}^{\mathbb{G}_2}_{\mathsf{poly}}$ oracles (the extraction success is argued similarly to how it is done in Lemma 4.6.4).

When $\mathcal{A}$ requests an honest update number i_{crit_2} in the second phase, $\mathcal{B}$ uses r_δ, s_δ from the challenge to set

$$\mathsf{srs}_s = \left(\begin{array}{l} G^{\frac{1}{r_\delta z + s_\delta}}, H^{\frac{1}{r_\delta z + s_\delta}}, \{G^{(r_\delta z + s_\delta)\left(Z_\beta(z) u_i(Z_x(z)) + Z_\alpha(z) v_i(Z_x(z)) + w_i(Z_x(z))\right)}\}_{i=\ell+1}^{m}, \\ \{G^{(r_\delta z + s_\delta)(Z_x(z))^i t(Z_x(z))}\}_{i=0}^{n-2} \end{array} \right).$$

Notice that $\mathcal{B}$ embeds $r_\delta z + s_\delta$ in an inverted way. This is due to the fact that we only have $G^{1/(r_\delta z + s_\delta)}$ and $H^{1/(r_\delta z + s_\delta)}$ in the dlog challenge, but when we do the second phase update we must construct the $G^{(\alpha u_i(x) + \ldots)/\delta}$ and $G^{t(x)x^i/\delta}$ elements which we cannot do if δ is in the denominator. The reason is that these elements are constructed from $G^{x^i/\delta}, G^{\alpha x^i/\delta}, G^{\beta x^i/\delta}$ monomials, and since $\mathcal{B}$ does not know δ, it cannot exponentiate the elements $\mathcal{A}$ provided as an input to the update query, so $\mathcal{B}$ must construct these problematic SRS parts from scratch using the edlog challenge. For example, x^i/δ would be represented as $(r_x z + s_x)^i/(r_\delta z + s_\delta)$, which is not a Laurent polynomial but a rational function in z. So we cannot build $G^{x^i/\delta}$ from our dlog challenge with the direct δ embedding strategy. At the same time, embedding $r_\delta z + s_\delta$ in an inverted way can be done: now x^i/δ is $G^{(r_x z + s_x)^i (r_\delta z + s_\delta)}$ which is a positive-power polynomial in z, so we can build it from $\{G^{z^j}\}$ which are available. Simpler SRS elements G^δ and H^δ can also be constructed: they are just $G^{1/(r_\delta z + s_\delta)}, H^{1/(r_\delta z + s_\delta)}$. Since if $r_\delta z + s_\delta$ is uniform, $1/(r_\delta z + s_\delta)$ is also uniform, and $\mathcal{A}$ cannot notice the inverted embedding.

The maximum degree polynomial here is in the fourth set of srs_s elements, $G^{(r_\delta z + s_\delta)(Z_x(z))^{n-2} t(Z_x(z))}$ equal to $2n-1$, which explains the $\mathbb{G}_1$ degree of edlog. As for $\mathbb{G}_2$, its maximum degree is in $H^{(r_x z + s_x)^{2n-2}}$ in srs_u, and thus equal to $2n-2$. Therefore, $(2n-1, 2n-2)$-**edlog** is enough for the embedding to succeed.

Then $\mathcal{B}$ simulates a proof of correctness by using $\mathsf{SimUpdProof}$ as in $\varphi = 1$ case, which again uses the PoK simulator in a black-box way after constructing an instance x. In this case, with the inverted embedding, we must set $G^{\hat{\iota}'} = (G^{\frac{1}{r_\delta z + s_\delta}})^{\hat{\iota}-1}$ and similarly for H, but we can still do it from the edlog challenge.

When $\mathcal{A}$ finalises in phase 2, $\mathcal{B}$ sees the verifying proofs $(\pi^{\mathcal{A}}_{2:1}, \ldots, \pi^{\mathcal{A}}_{2:t_2})$ for all updates after the last (critical) update query that $\mathcal{A}$ made. Again, the actual number of proofs in the SRS is higher, but $\mathcal{B}$ discards the prefix corresponding to the pre-critical execution. Then $\mathcal{B}$ can extract $\vec{\delta^{\mathcal{A}}}$ such that

$$\mathsf{srs}_s = \left(\begin{array}{l} G^{\frac{\prod_j \delta^{\mathcal{A}}_j}{r_\delta z + s_\delta}}, H^{\frac{\prod_j \delta^{\mathcal{A}}_j}{r_\delta z + s_\delta}}, \left\{ G^{\frac{(r_\delta z + s_\delta)\left(\hat{Z}_\beta(z) u_i(\hat{Z}_x(z)) + \hat{Z}_\alpha(z) v_i(\hat{Z}_x(z)) + w_i(\hat{Z}_x(z))\right)}{\prod_j \delta^{\mathcal{A}}_j}} \right\}_{i=\ell+1}^{m}, \\ \left\{ G^{\frac{(r_\delta z + s_\delta)(\hat{Z}_x(z))^i t(\hat{Z}_x(z))}{\prod_j \delta^{\mathcal{A}}_j}} \right\}_{i=0}^{n-2} \end{array} \right)$$

where $j = 1, \ldots, t_2$. We, as before, set $Z_\delta(Z) = \frac{r_\delta Z + s_\delta}{\prod_j \delta^{\mathcal{A}}_j}$.

We first define $Q_3(Z_x, Z_\alpha, Z_\beta, Z_\delta) = Q_2(Z_x, Z_\alpha, Z_\beta, 1/Z_\delta)$, which inverts the last coefficient, to account for the inverted embedding of δ trapdoor. From **bad** we know $Q_2 \not\equiv 0$, and $Q_2(\vec{\psi}_2) = 0$; Q_3 has similar properties. First, if $Q_2 \not\equiv 0$, then $Q_3 \not\equiv 0$, since if Q_2 includes some nonzero monomial MZ^i_δ for M monomial in Z_x, Z_α, Z_β, and some i, then in Q_3 there will be a nonzero coefficient of MZ^{-i}_δ. Second, if $Q_2(\vec{\psi}_2) = 0$, then $Q_3(z_x, z_\alpha, z_\beta, 1/z_\delta) = Q_2(\vec{\psi}) = 0$. We will denote $\vec{\psi}_3 := (z_x, z_\alpha, z_\beta, 1/z_\delta)$, so $Q_3(\vec{\psi}_3) = 0$.

Let us transform the Laurent polynomial Q_3 to a standard positive-power polynomial. We do this by defining $Q_4(\{Z_\iota\}_\iota) := Q_3(\{Z_\iota\}_\iota) \cdot Z^2_\delta$, where Z_δ is a formal variable. Q_4 is a positive power polynomial since Q_3 can only have at most Z^{-2}_δ as a negative degree monomial: e.g. Z^{-1}_δ in both A and B, which is true even after Q_3 inversion on the previous step, since δ has powers 1 and -1 in the SRS. Moreover, since $Q_3(\{Z_\iota\}_\iota) \not\equiv 0$ and $Q_3(\vec{\psi}_3) = 0$, it follows that $Q_4(\{Z_\iota\}_\iota) \not\equiv 0$ and $Q_4(\vec{\psi}_3) = 0$.

Next we introduce $Q'(Z) := Q_4(r_x Z + s_x, r_\alpha Z + s_\alpha, r_\beta Z + s_\beta, r_\delta Z + s_\delta)$, which reinterprets Q_4 as a polynomial over Z instead of $\{Z_\iota\}$. Here, the last element $r_\delta Z + s_\delta$ is passed into Q_4 directly, since $r_\delta Z + s_\delta = 1/z_\delta$. From this it follows that $(r_x z + s_x, r_\alpha z + s_\alpha, r_\beta z + s_\beta, r_\delta z + s_\delta) = \vec{\psi}_3(z)$, and z is one of the roots of Q' since $Q'(z) = Q_4(\vec{\psi}_3(z)) = 0$.

If we can show that $Q'(Z) \neq 0$, then $\mathcal{B}$ can factor it to find z. To show this, let us first define an intermediate polynomial $Q'_3(Z) = Q_4(\{R_\iota Z + S_\iota\}_\iota)$ in variable Z over the ring of polynomials $\mathbb{Z}_p[R_\alpha, R_\beta, R_x, R_\delta, S_\alpha, S_\beta, S_x, S_\delta]$. Accoding to Lemma 2.3.1, the leading coefficient of $Q'_3(Z)$ is a polynomial $C(R_\alpha, R_\beta, R_x, R_\delta)$ with the same degree d as is the total degree of $Q_4(\{Z_\iota\}_\iota)$. Since the total degree of $Q_4(\{Z_\iota\}_\iota)$ is non-zero, then C is a non-zero polynomial. Values r_ι are information-theoretically hidden from $\mathcal{A}$ since $\mathcal{B}$ set critical trapdoors to be $z_\iota = r_\iota z + s_\iota$ (and for δ it is inverted). Therefore, $r_\alpha, r_\beta, r_x, r_\delta$ are chosen uniformly randomly and independently from C. According to the Schwartz-Zippel lemma (see Lemma 2.3.2), the probability that $c := C(r_\alpha, r_\beta, r_x, r_\delta) = 0$ is bounded by d/p. Hence, with an overwhelming probability $Q'(Z) \not\equiv 0$ since it has a non-zero leading coefficient c. This is sufficient for $\mathcal{B}$ to factor Q' and to find z.

It follows that the event **bad** can only happen with negligible probability. $\square$

4.9 Future Work

The proof of update soundness we present is quite complex structurally, and even we had to carefully structure it into different subsections and lemmas, it seems to be possible to simplify even further it and make more modular. However, we believe it is an inherent property of such proofs, especially in the AGM, and thus the question we would like to ask rather is *"how simpler proof structure can be achieved by adapting the proof model?"*. Another immediate question is whether it is possible to show simulation-extractable version of update (knowledge) soundness, similarly to how we do it in Chapter 3. This would allow (after a certain transformation to achieve black-box extractability) lifting our security properties to the UC framework.

Chapter 5

Multi-Asset Swaps from SNARKs

This chapter is based on the work "Zswap: zk-SNARK Based Non-Interactive Multi-Asset Swaps", published in PoPETS 2022, and co-authored by Felix Engelmann, Thomas Kerber, and Markulf Kohlweiss.

Pivacy-oriented cryptocurrencies, like Zcash or Monero, provide fair transaction anonymity and confidentiality, but lack important features compared to fully public systems, like Ethereum. Specifically, supporting assets of multiple types and providing a mechanism to atomically exchange them, which is critical for e.g. decentralized finance (DeFi), is challenging in the private setting.

An *atomic swap* is the exchange of different assets between multiple parties. Atomicity means that either all the participants get their desired output simultaneously or the transaction is aborted completely. A classical example is a foreign currency exchange, where a bank sells a foreign currency to a customer who pays in the local currency. There, atomicity is guaranteed by simultaneously handing over the assets.

By combining insights and security properties from Zcash and SwapCT ([Engelmann et al., 2021] an atomic swap system for Monero), we present a simple zk-SNARKs-based transaction scheme, called Zswap, which is carefully malleable to allow the merging of transactions, while preserving anonymity. Our protocol enables multiple assets and atomic exchanges by making use of sparse homomorphic commitments with aggregated open randomness, together with Zcash-friendly simulation-extractable non-interactive zero-knowledge (NIZK) proofs. This results in a provably secure privacy-preserving transaction protocol, with efficient swaps, and overall performance close to that of existing deployed private cryptocurrencies. It is similar to Zcash Sapling and benefits from existing code bases and implementation expertise.

The transactions in our atomic swap protocol only reveal a map of the imbalance for each asset where the sum of its inputs is unequal to the sum of its outputs. Hence, a balanced transaction is the one that does not reveal any amounts or types, it is fully private and semantically similar to a zcash transactions. Otherwise it is imbalanced and can be viewed as an *exchange offer*, the imbalance of which encodes exactly the exchange order conditions. Next, these unbalanced transactions can be merged off-chain by untrusted parties. The mergers can neither deanonymize senders or receivers nor correlate a subsequent spending of an output by its precise amount and type. With such little trust required in mergers, this very basic functionality already allows creating local exchange markets, where users can send exchange offers (as transactions with a negative imbalance for the asked token and a positive imbalance for the offered), and community-selected participants can match them and merge them to then submit to the blockchain. The anonymity of the system is controlled by users: the system allows both private swaps between several parties, who agree on their exchange off-chain, and bigger exchange pools, as just mentioned. In both cases, the only information that the mergers and other users see is the one necessary to match the offers (total value for each unbalanced type), and it is erased as soon as the transaction is balanced and sent to the ledger. For partial merges, any type with an imbalance of zero is dropped. An open group of participants in a pool may provide sufficient liquidity and maintain a public order book, similar to classical exchanges.

In Section 5.1 we start by presenting an intuition for our construction together with a technical overview and comparison with Zcash and other concurrent work. The main contributions of this chapter are summarised as follows:

Formal Modelling via OTA Scheme and Update Oracles. In Section 5.3 we define a formal model called a One-Time Account (OTA) System that abstracts a private account-related mechanism used in private cryptocurrencies. In particular it captures the nullifier-like private UTXO mechanism, and we show how to instantiate an OTA system from Zcash in Section 5.3. The OTA model and our proof techniques could be of independent interest for proving systems such as Zcash and Monero secure.

Section 5.4 presents a formal security model for a multi-asset Zcash system with swaps that builds on top of the OTA system. The main security notions of the scheme are somewhat similar to Zcash, but our modelling approach is quite different, since we have to take into an account the update mechanics, which we model through adversarial access to state-modifying oracles.

ZSwap Construction. In Section 5.5 we present Zswap, being a minimal practical instantiation of a cryptocurrency mechanism supporting multiple assets and private non-interactive atomic swaps. It is based on a simplified version of Zcash that removes authorization and blinding signatures.

In Section 5.6 we prove our construction secure under commonly used assumptions similar to the ones used in Zerocash. This validates the removal of the Zcash signatures and shows that the perfect hiding and binding properties of spend and output commitments is sufficient for security. In addition, since Zswap uses a multi-asset mechanics that is being integrated in real-world solutions like Zcash, we believe to simultaneously provide the first security proofs of it which are of independent value.

Implementation and Evaluation. We implement and evaluate our protocol. In Section 5.7 we present the results showing that our merging mechanism is extremely efficient, and all the other performance overheads of our construction related to transaction creation and verification are small, as compared to the basic single-asset protocol without swaps.

Speaking of adoption and practical applications, we believe that a simple atomic swap mechanism we provide can be viewed as an easy and useful add-on to many Zcash-like private cryptocurrencies. An interesting open question is how to integrate Zswap with private smart contracts to support more elaborate private DeFi solutions operating on complex *intents*, similarly to the approach of the Anoma protocol[1]. A first step in that direction would be to extend a public smart contract system with minting policies for private assets, to support, e.g. the private trading of NFTs. Finally, our mechanism is compatible with existing consensus protocols, such as proof-of-work or stake, and can be viewed as a standalone cryptocurrency.

[1] `https://anoma.net/`

5.1 Technical Overview and Related Work

A major challenge of many existing solutions that investigate transaction malleability and swap mechanics is that transaction data is not explicitly separated from the transaction signature, which binds inputs from multiple users together. This, often results in the need for (slow) MPC protocols to construct such a signature jointly. An important insight of our work is that the Zcash ecosystem already implement signature separation. It was first introduces by Zcash Sapling [Hopwood et al., 2022] to reduce the size of SNARK circuits for large transactions, and was inherited by the corresponding Shielded Assets, or the similar MASP multi-asset protocols. Another inspiration of our work is SwapCT, that realizes atomic swaps on Monero [Engelmann et al., 2021]. We continue this direction by designing and proving secure an extension of a Zcash Sapling like system that satisfies the required multi-asset and atomic swap properties.

As explained in Section 2.7.1, Zcash Sapling, unlike in the Zerocash paper, separates the validation of inputs and outputs (each input and output requires a separate NIZK) from the transaction balancing, which is done using homomorphic commitments and a Schnorr-like binding signature. The binding signature "seals" the inputs and outputs in place, forbidding adding or removing any extra inputs or outputs. Instead, in our work we use a sparse multi-value Pedersen commitments and relax the signature, allowing transactions to be non-interactively merged together. While inspired by SwapCT, we deal with different challenges specific to the zk-SNARK setting and take advantage of the existing signature separation in Zcash Sapling. The explicit signature separation opens more space for potential transaction malleability, and should be taken with care. To our knowledge our work is the first to extend the security analysis of [Ben-Sasson et al., 2014a] in that direction.

Our modelling relies on a one-time account (OTA) scheme to anonymously and confidentially create and store value in notes. This is an abstraction from existing protocols in Zcash and Monero. The spending of input notes, stored in a Merkle tree, is authorized by a simulation extractable (SE) non-interactive zero-knowledge proof (NIZK). Double spends are prevented by proving the correctness of a deterministic nullifier, marking an input as spent. To connect inputs to newly created output notes, we use sparse homomorphic commitments to value-type pairs, which can be summed to check that the transaction is well-balanced. Importantly, the values and types remain hidden even when we publish their aggregated randomness, which is necessary to facilitate transaction merging.

Our protocol bears similarities to variants of Zcash and SwapCT which we compare in more detail:

Zcash Sapling: Compared to Sapling, we add multi-asset support and remove the authorization signature which is replaced by the SE NIZK. We also do not need a binding signature over all intermediate commitments — instead we directly publish the randomness. This opens room for controlled malleability, and enables non-interactive joint transaction generation (by merging) without MPC, which can be used to implement swaps mechanics. The multi-asset aspect of our scheme is very similar to Shielded Assets or MASP, both of which extend Sapling but do not provide a mechanism for atomic swaps. We also provide a rigorous theoretical formalization, which is lacking for both Sapling and its Shielded Asset and MASP variants, and can be of independent interest.

Conversely, the technical discussions and implementation effort that went into the Shielded Assets ZIP 220 and the MASP implementation are valuable starting points for the deployment of Zswap as part of a larger Zcash like blockchain. In particular, deployment of our new transactions into Zcash requires a hard fork and creates a new shielded pool holding typed notes. Transferring tokens from an existing shielded pool is possible by explicitly adding the `Zcash` type to the notes.

SwapCT: Our work is inspired by SwapCT which also provides atomic swaps and transaction merging, but on top of Monero (ring based anonymity). We simplified their scheme to unify their offers and transactions. With the use of SNARKs (requiring a trusted or transparent setup) our simpler construction no longer needs their anonymously aggregatable signatures. We achieve better confidentiality for unfinished transactions as we only reveal a total imbalance instead of per input and output values. Another important difference is that the use of SNARKS allows to efficiently support large rings. This motivates the use of a single global ring in our formalization.

Formalization: We formalize our protocol using game based definitions, which generalize Zerocash definitions, but are also compatible with Monero except for using a single global ring (as in Zcash and differing from SwapCT).

Zerocash [Ben-Sasson et al., 2014a] formalizes *Ledger indistinguishability*, *Transaction non-malleability* and *Balance*. Our balance definition is very similar with the addition of checking multiple assets. We replace indistinguishability with the privacy definition, as our transactions may be circulated off-chain. The non-malleability of merged transactions is inspired by Zerocash and SwapCT [Engelmann et al., 2021]. Instead of the strong non-malleability, it provides a relaxed property that controls malleability (transaction merging) and only requires theft prevention, namely that an adversary cannot steal from the intended recipients of a transaction, or split unbalanced transactions, rerouting honest funds. Modelling this malleability relies on a variant of hiding of the Pedersen commitment scheme, when the joint commitment randomness is revealed, which we call $\mathrm{HID\text{-}OR}$[2]. This property does not require any additional security assumptions.

5.2 Commitments and Open Randomness

In Section 2.5 we introduce the notion of sparse homomorphic commitments, and showed how to instantiate it in the random oracle model. In this chapter we will use SHC in what we called the "dynamic mode" — we assume the existence of a cryptographic hash function $H(\cdot)$, which will be computing the bases $H_i = H(\mathsf{ty}_i)$ on-the-fly.

A remark about value space of commitments must be made. To avoid value overflows in our protocol, we will upper-bound the committed values by $2^\alpha - 1$, and the total number of commitments that we homomorphically combine to β — in this way, the maximum sum in the exponent of a single base will be $\alpha + \beta$ bits, which must be below certain B which in turn must be reasonably smaller than the order of any base $H(\mathsf{ty})$ (thus the message space). We assume that this upper bound $B(\lambda)$ and the parameters $(\alpha(\lambda), \beta(\lambda))$, for simplicity, is given by the commitment scheme.

The Zswap construction that we will present later critically relies on the certain property of the sparse Pedersen commitment scheme, instantiating SHC, that we

[2]HIDing with Open Randomness

$\underline{\mathrm{HID\text{-}OR}^b_{\mathcal{A}}(1^\lambda):}$

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$

$\{(\mathsf{ty}_t, a_t), (\mathsf{ty}'_t, a'_t)\}_{t=0}^{1} \leftarrow \mathcal{A}(\mathsf{pp})$

$\Delta_{\mathsf{ty},t} := \{\mathbf{if}\ \mathsf{ty}_t = \mathsf{ty}\ \mathbf{then}\ a_t\ \mathbf{else}\ 0\} - \{\mathbf{if}\ \mathsf{ty}'_t = \mathsf{ty}\ \mathbf{then}\ a'_t\ \mathbf{else}\ 0\}$

assert $\forall \mathsf{ty} \in \bigcup \mathsf{ty}_t \cup \bigcup \mathsf{ty}'_t : \Delta_{\mathsf{ty},0} = \Delta_{\mathsf{ty},1}$

$r, r' \xleftarrow{\$} \mathbb{R}$

$\mathsf{com} \leftarrow \mathsf{Commit}(\mathsf{ty}_b, a_b, r);\ \mathsf{com}' \leftarrow \mathsf{Commit}(\mathsf{ty}'_b, a'_b, r')$

$b' \leftarrow \mathcal{A}(\mathsf{com}, \mathsf{com}', \mathsf{rc} = r - r')$

return b'

Figure 5.1: Hiding with Open Randomness Game

call *hiding with open randomness*. We observe that the adversary cannot distinguish between two pairs of commitments that sum to the same values (per type), even if we reveal their common randomness. This is the main property used in our swaps: we will use it to argue that transaction can be merged (to join two unbalanced swap offers), but not split apart (so swap offers cannot be adversarially modified).

Lemma 5.2.1 (Hiding with Open Randomness)**.** *The sparse Pedersen commitment scheme is* perfectly hiding with open randomness *in the RO model. By this we mean that if for all $\lambda \in \mathbb{N}$ and all PPT $\mathcal{A}$:*

$$\Pr\left[\mathrm{HID\text{-}OR}^1_{\mathcal{A}}(1^\lambda) = 1\right] - \Pr\left[\mathrm{HID\text{-}OR}^0_{\mathcal{A}}(1^\lambda) = 0\right] = 0$$

where the game is defined in Fig. 5.1.

The proof of this statement is based on perfect hiding together with the equivocation of the Pedersen commitment scheme.

Lemma 5.2.1. Let the hash function be modelled by the random oracle, and $H(\mathsf{ty}_i) = G^{t_i}$, and thus the challenger knows all t_i. The form of commitments that are given to the adversary is thus:

$$[t_b a_b + r], [t'_b a'_b + r'], \mathsf{rc} = r - r'$$

The initial game is $\mathrm{HID\text{-}OR}^0$, in which $\mathcal{A}$ sees the previous equation for $b = 0$. In $\mathcal{G}1$ we sample $\hat{r}$ and instead give $\mathcal{A}$ the following:

$$[\hat{r}], [t'_0 a'_0 + r'], \mathsf{rc} = (\hat{r} - t_0 a_0) - r'$$

Note that $t_0 a_0 + r$ and $\hat{r}$ are both uniform, so $\mathcal{A}$ does not see the difference, the transition is perfect. Similarly we equivocate the second commitment:

$$[\hat{r}], [\hat{r}'], \mathsf{rc} = (\hat{r} - \hat{r}') - (t_0 a_0 - t_0' a_0')$$

And now we "switch" the elements to the case of $b = 1$. Let $T = \{t_0, t_0'\} = \{t_1, t_1'\}$. Given $\Delta_{\mathsf{ty},0} = \Delta_{\mathsf{ty},1}$ for all $\mathsf{ty} \in T$ we have:

$$t_0 a_0 - t_0' a_0' = \sum_{t \in T} t \Delta_{\mathsf{ty},0} = \sum_{t \in T} t \Delta_{\mathsf{ty},1} = t_1 a_1 - t_1' a_1'$$

Therefore what $\mathcal{A}$ sees now is:

$$[\hat{r}], [\hat{r}'], \mathsf{rc} = (\hat{r} - \hat{r}') - (t_1 a_1 - t_1' a_1')$$

So we can proceed with replacing $\hat{r}$ and $\hat{r}'$ "back" into real commitments to the $b = 1$ values similarly to how we abstracted them in the first steps of the proof. The end result is a distribution that $\mathcal{A}$ sees for $b = 1$ ($\mathrm{HID\text{-}OR}^1$): $[t_1 a_1 + r], [t_1' a_1' + r'], \mathsf{rc} = r - r'$ so our hiding holds with probability 1 (is perfect). □

Corollary 5.2.1.1. *Assuming commitment hiding, Lemma 5.2.1 holds even if $\mathcal{A}$ provides two sets of size n (not just pairs) of input and output commitments, $\{(\mathsf{ty}_{i,t}, a_{i,t}), (\mathsf{ty}'_{i,t}, a'_{i,t})\}_{i,t=0,0}^{n,1}$, as long as they still jointly sum to the same values per type.*

Proof. The proof is exactly the same as the previous one, except that we first "idealise" $2n$ commitments (and not just two) from $b = 0$, and then de-idealise them all back. The transition logic in the middle holds similarly because $\Delta_{\mathsf{ty},0} = \Delta_{\mathsf{ty},1}$. □

5.3 One-Time-Account Scheme

To highlight our novel transaction mechanism, we first describe the *one-time account (OTA) scheme* that we use to model the mechanics of underlying accounts our transactions use. The OTA scheme may be seen as an anonymous version of an unspent transaction output (UTXO) system (e.g. the one used in Bitcoin); it generalises accounts of privacy-preserving transaction systems such as Zcash or Monero. Instead of creating transaction outputs including a long term identity as with plain UTXO, an OTA scheme generates a unique, anonymous, one-time account for each transaction output. To support the UTXO functionality to decide if a one-time-account is still valid, the OTA scheme allows generating a unique *nullifier* which anonymously marks it as spent. A duplicate nullifier indicates that the same one-time account is used.

The OTA scheme is used as follows. After a system Setup, each participant generates their credentials with KeyGen. Anyone with the public key can then non-interactively derive a one-time account, *also called "note"*[3], with Gen. Each note contains a set vector of attributes. Most importantly, one attribute is the amount the note represents. In our multi-type setting, a second attribute is the type of the amount. Other systems may make use of additional note attributes such as e.g. time locks — a timestamp identifier, allowing a note to be spendable only after a specified block number. To allow the intended recipient to recover the note, it is accompanied by an encryption C calculated by Enc. With the secret key, note and ciphertext, the original owner $\mathsf{Receive}$s the note and is then able to create the corresponding nullifier with $\mathsf{NulEval}$, a unique serial value characterizing the note that cannot be predicted by anyone else than the owner.

We emphasize that any party can create an OTA account for anyone knowing only their public key. The OTA itself does not allow any claims to the value stored inside. In our system, only the OTAs which are included as outputs of a valid, balanced and persisted transaction can be claimed by their owners in subsequent transactions. The transaction then enforces that sufficient inputs were consumed to cover for the output. To mint coins an OTA needs to be included on the ledger in an unbalanced transaction for the newly minted type. This transaction needs to be accepted by the ledger rules. This behaviour might as well be governed by a smart contract.

Definition 5.3.1. *A One-Time-Account (OTA) scheme consists of the PPT algorithms* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Gen}, \mathsf{Receive}, \mathsf{NulEval})$ *defined as follows:*

$\mathsf{Setup}(1^\lambda) \xrightarrow{\$} \mathsf{pp}$**:** *takes the security parameter* λ *and outputs the public parameters* pp *which are implicitly provided to the subsequent algorithms. This includes the note randomness space* $\mathbb{S}$*, the message space* $\mathbb{M}$*, and the encryption randomness space* Ξ*.*

$\mathsf{KeyGen}(1^\lambda) \xrightarrow{\$} (\mathsf{sk}, \mathsf{pk})$**:** *generates a key pair* $(\mathsf{sk}, \mathsf{pk})$*. We assume a function* $P : \mathsf{sk} \mapsto \mathsf{pk}$ *for generating the public key from a secret key.*

[3] Unlike in some other works, "note" here means the final, hidden account; and not the plaintext coin.

$\mathsf{Gen}(\mathsf{pk}, \vec{a}, r) \to \mathsf{note}$**:** *takes a public key* pk*, a vector of attributes* $\vec{a} \in \mathbb{M}^{|\vec{a}|}$*, where, by agreement, the first might be an amount, and randomness* $r \in \mathbb{S}$*. It outputs a one-time-account, known as* note*.*

$\mathsf{Enc}(\mathsf{pk}, (\vec{a}, r), \xi) \to C$**:** *encrypts the attributes* $\vec{a}$ *and the randomness* $r \in \mathbb{S}$ *to the public key* pk *with additional randomness* $\xi \in \Xi$*. It outputs a ciphertext* C*.*

$\mathsf{Receive}(\mathsf{note}, C, \mathsf{sk}) \to (\vec{a}, r)/\bot$**:** *if the* note *and ciphertext* C *belongs to the secret key* sk*, the algorithm outputs the vector of attributes* $\vec{a}$ *and the randomness* r *or fails otherwise.*

$\mathsf{NulEval}(\mathsf{sk}, r) \to \mathsf{nul}$**:** *Takes a secret key* sk *and a randomness* r *and outputs a nullifier* nul*.*

In addition, the algorithms Gen *and* NulEval *must be efficiently provable in zero-knowledge. More formally, the construction must provide NIZK-friendly circuits for the following languages:*

$$\mathcal{L}^{\mathsf{nul}} = \{(\mathsf{note}, \mathsf{nul}) \mid \exists (\mathsf{sk}, \vec{a}, r) : \mathsf{note} = \mathsf{Gen}(P(\mathsf{sk}), \vec{a}, r) \land \mathsf{nul} = \mathsf{NulEval}(\mathsf{sk}, r)\}$$
$$\mathcal{L}^{\mathsf{open}} = \{\mathsf{note} \mid \exists (\mathsf{pk}, \vec{a}, r) : \mathsf{note} = \mathsf{Gen}(\mathsf{pk}, \vec{a}, r)\}$$

The language $\mathcal{L}^{\mathsf{open}}$ *may optionally be extended, such that elements of* $\vec{a}$ *have relations to other commitments.*

We first present the basic correctness and soundness properties of the OTA scheme, which primarily dictate how Receive should be implemented.

Definition 5.3.2 (OTA Correctness)**.** *An OTA scheme is correct if for any* $\lambda \in \mathbb{N}$ *with* $\mathsf{pp} \in \mathsf{Setup}(1^\lambda)$ *it holds that any honestly generated note is receivable. Formally, for every* $(\mathsf{sk}, \mathsf{pk}) \in \mathsf{KeyGen}(\mathsf{pp})$*, every* $(\vec{a}, r) \in \mathbb{M}^{|\vec{a}|} \times \mathbb{S}$ *and* $\xi \in \Xi$ *it holds that*

$$\mathsf{Receive}\left(\mathsf{Gen}(\mathsf{pk}, \vec{a}, r), \mathsf{Enc}(\mathsf{pk}, (\vec{a}, r), \xi), \mathsf{sk}\right) = (\vec{a}, r)$$

Definition 5.3.3 (OTA Soundness)**.** *An OTA scheme is sound if any non-*$\bot$ *output of* Receive *reconstructs the note that was given to* Receive *as an input. Formally, for any* $\lambda \in \mathbb{N}$ *with* $\mathsf{pp} \in \mathsf{Setup}(1^\lambda)$*, every* $(\mathsf{sk}, \mathsf{pk}) \in \mathsf{KeyGen}(\mathsf{pp})$*, every* $(\vec{a}, r) \in \mathbb{M}^{|\vec{a}|} \times \mathbb{S}$*,* $\xi \in \Xi$ *it holds that*

$$\mathsf{Receive}(\mathsf{note}, C, \mathsf{sk}) = (\vec{a}, r) \implies \mathsf{OTA.Gen}(P(\mathsf{sk}), \vec{a}, r) = \mathsf{note}$$

If correctness is present, soundness can be easily achieved by appending a condition to the realisation of Receive that asserts that extracted $(\vec{a}, r)$ is valid.

Once a note is created, it must bind the attributes and prevent opening the note to a different vector, even for the owner with the correct secret key.

Definition 5.3.4 (OTA Binding). *An OTA scheme is binding with regard to the accounts created and the vector of attributes if for any* $\lambda \in \mathbb{N}$ *with* $\mathsf{pp} \in \mathsf{Setup}(1^\lambda)$ *and any PPT adversary* $\mathcal{A}$*, it holds that*

$$\Pr\left[\begin{array}{l}(\mathsf{pk}_0, r_0, \vec{a}_0, \mathsf{pk}_1, r_1, \vec{a}_1) \leftarrow \mathcal{A}(\mathsf{pp})\\ \mathsf{note}_0 \leftarrow \mathsf{Gen}(\mathsf{pk}_0, \vec{a}_0, r_0)\\ \mathsf{note}_1 \leftarrow \mathsf{Gen}(\mathsf{pk}_1, \vec{a}_1, r_1)\\ \textbf{return } \mathsf{note}_0 = \mathsf{note}_1 \wedge \vec{a}_0 \neq \vec{a}_1\end{array}\right] = \mathsf{negl}(\lambda)$$

The following privacy property assures that a note and its ciphertext do not leak who the note belongs to and what attributes it stores.

Definition 5.3.5 (OTA Privacy). *Consider the following oracle, modelling OTA note receiving:*

$$\boxed{\begin{array}{l}\underline{\mathcal{O}_{\mathsf{Rcv}}^{\mathsf{note}^*, C^*}(i, \mathsf{note}, C)}:\\ \textbf{assert } \mathsf{note} \neq \mathsf{note}^* \wedge C^* \neq C\\ \textbf{return } \mathsf{Receive}(\mathsf{note}, C, \mathsf{sk}_i)\end{array}}$$

An OTA scheme is private if for any $\lambda \in \mathbb{N}$ *with* $\mathsf{pp} \in \mathsf{Setup}(1^\lambda)$ *and any stateful PPT adversary* $\mathcal{A}$*, it holds that*

$$\Pr\left[\begin{array}{l}(\mathsf{sk}_0, \mathsf{pk}_0) \xleftarrow{\$} \mathsf{KeyGen}()\\ (\mathsf{sk}_1, \mathsf{pk}_1) \xleftarrow{\$} \mathsf{KeyGen}()\\ (i_0, \vec{a}_0, i_1, \vec{a}_1) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Rcv}}^{\perp,\perp}}(\mathsf{pp}, \mathsf{pk}_0, \mathsf{pk}_1)\\ b \xleftarrow{\$} \{0,1\}, r \xleftarrow{\$} \mathbb{S}, \xi \xleftarrow{\$} \Xi\\ \mathsf{note}^* \leftarrow \mathsf{Gen}(\mathsf{pk}_{i_b}, \vec{a}_b, r)\\ C^* \leftarrow \mathsf{Enc}(\mathsf{pk}_{i_b}, (\vec{a}_b, r), \xi)\\ b' \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Rcv}}^{\mathsf{note}^*, C^*}}(\mathsf{note}^*, C^*)\\ \textbf{return } b = b' \wedge |\vec{a}_0| = |\vec{a}_1|\end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

The privacy game implicitly subsumes (1) note and ciphertext hiding, and (2) note and encryption anonymity (key privacy). In the first case, $\mathcal{A}$ cannot decide the content of the note or the ciphertext; in the second, it cannot decide which key was used to create it. E.g. if $\mathtt{note}$ is not hiding, $\mathcal{A}$ efficiently wins the game by first returning two different vectors $\vec{a}_0 \neq \vec{a}_1$ and then distinguishing the note $\mathtt{note}_b$ according to the attribute vector.

We require notes to be *unique* when generated with honest randomness:

Definition 5.3.6 (Note Uniqueness). *A binding and private OTA scheme satisfies honestly generated note uniqueness: for all PPT $\mathcal{A}$,*

$$\Pr\left[\begin{array}{l} ((\mathsf{pk}_0, \vec{a}_0), (\mathsf{pk}_1, \vec{a}_1)) \leftarrow \mathcal{A}(1^\lambda) \\ r_0, r_1 \xleftarrow{\$} \mathbb{S} \\ \textbf{return } \mathsf{Gen}(\mathsf{pk}_0, \vec{a}_0, r_0) = \mathsf{Gen}(\mathsf{pk}_1, \vec{a}_1, r_1) \end{array}\right] = \mathsf{negl}(\lambda)$$

The next property, similar to the tagging scheme of Omniring [Lai et al., 2019], captures the requirement that nullifiers produced with sk must be only "predictable" for the party that holds sk. This is achieved by requiring $\mathsf{NulEval}$ to behave like a pseudorandom function.

Definition 5.3.7 (Nullifier Pseudorandomness). *Let $\lambda \in \mathbb{N}$ with $\mathsf{pp} \in \mathsf{Setup}(1^\lambda)$, $(\mathsf{sk}, \mathsf{pk}) \xleftarrow{\$} \mathsf{KeyGen}()$, and f be randomly sampled function on the range $\{0,1\}^{|\mathbb{S}|} \to \{0,1\}^{|\mathsf{NulEval}(\mathsf{sk},\cdot)|}$. An OTA scheme nullifier is* pseudorandom *if for any PPT adversary $\mathcal{A}$, it holds that*

$$\Pr\left[\mathcal{A}^{\mathsf{Receive}(\cdot,\cdot,\mathsf{sk}), f(\cdot)}(\mathsf{pk}) = 1\right] - \Pr\left[\mathcal{A}^{\mathsf{Receive}(\cdot,\cdot,\mathsf{sk}), \mathsf{NulEval}(\mathsf{sk},\cdot)}(\mathsf{pk}) = 1\right] = \mathsf{negl}(\lambda)$$

To detect duplicate use of the same note, each is assigned a unique nullifier. Even with knowledge of the secret key, it is not possible to create two different nullifiers for the same note. The separate secret keys are important for constructions based on algebraic nullifiers. E.g. Omniring creates tags as $g^{\frac{1}{\mathsf{sk}+r}}$ for a generator g, so randomness may be "traded" for secret key. Allowing only one note with a single secret key would not capture the realistic setting where an adversary controls multiple correlated accounts.

Definition 5.3.8 (Nullifier Uniqueness). *An OTA scheme satisfies nullifier uniqueness if for any $\lambda \in \mathbb{N}$ with $\mathsf{pp} \in \mathsf{Setup}(1^\lambda)$ and any PPT adversary $\mathcal{A}$, it holds that*

$$\Pr\left[\begin{array}{l} (\mathsf{sk}_0, r_0, \vec{a}_0, \mathsf{sk}_1, r_1, \vec{a}_1) \leftarrow \mathcal{A}(\mathsf{pp}); \\ \mathsf{note}_0 \leftarrow \mathsf{Gen}(P(\mathsf{sk}_0), \vec{a}_0, r_0); \\ \mathsf{note}_1 \leftarrow \mathsf{Gen}(P(\mathsf{sk}_1), \vec{a}_1, r_1) \\ \textbf{return } \mathsf{note}_0 = \mathsf{note}_1 \wedge \mathsf{NulEval}(\mathsf{sk}_0, r_0) \neq \mathsf{NulEval}(\mathsf{sk}_1, r_1) \end{array}\right] = \mathsf{negl}(\lambda)$$

Finally, the generated nullifiers must be also collision-resistant:

Definition 5.3.9 (Nullifier Collision-Resistance)**.** *An OTA scheme satisfies nullifier collision-resistance if for any* $\mathsf{pp} \in \mathsf{Setup}(1^\lambda)$ *and PPT* $\mathcal{A}$*, it holds that*

$$\Pr\left[\ (\mathsf{sk}_0, r_0, \mathsf{sk}_1, r_1) \leftarrow \mathcal{A}(\mathsf{pp}) \quad : \quad \mathsf{NulEval}(\mathsf{sk}_0, r_0) = \mathsf{NulEval}(\mathsf{sk}_1, r_1)\ \right] = \mathsf{negl}(\lambda)$$

5.3.1 Zerocash-Style OTA

To use Zerocash style one-time-accounts, we provide an instantiation which is compatible with Zerocash.

Given a key anonymous public key encryption scheme PKE, a labeled PRF and a commitment scheme (Commit). The PRF of type PRF^{sn} must in addition be collision-resistant. The randomness space is $\mathbb{S} := (\{0,1\}^\lambda)^3$, the key space is and the message space consists of two λ-bit integers $\mathbb{M} := \mathbb{Z}^2_{2^\lambda}$ the first representing an amount and the second identifying a type. The randomness space Ξ is specified by PKE.

$\mathsf{Setup}(1^\lambda) \xrightarrow{\$} \mathsf{pp}$**:** Initializes the SNARK parameters pp.

$\mathsf{KeyGen}() \xrightarrow{\$} (\mathsf{sk}, \mathsf{pk})$**:** Samples a_{sk} randomly from $\{0,1\}^\lambda$ and defines $a_{\mathsf{pk}} \leftarrow \mathsf{PRF}^{\mathsf{addr}}_{a_{\mathsf{sk}}}(0)$; samples an encryption key pair $\mathsf{sk}_{\mathsf{enc}}, \mathsf{pk}_{\mathsf{enc}}$ using the corresponding PKE algorithm. Return $\mathsf{sk} = (a_{\mathsf{sk}}, \mathsf{sk}_{\mathsf{enc}}), \mathsf{pk} = (a_{\mathsf{pk}}, \mathsf{pk}_{\mathsf{enc}})$.

$P(\mathsf{sk})$**:** is defined as $P(a_{\mathsf{sk}}, \mathsf{sk}_{\mathsf{enc}}) := (\mathsf{PRF}^{\mathsf{addr}}_{a_{\mathsf{sk}}}(0), P_{\mathsf{enc}}(\mathsf{sk}_{\mathsf{enc}}))$ where P_{enc} is assumed to be defined in the encryption scheme.

$\mathsf{Gen}(\mathsf{pk}, \vec{a}, r) \rightarrow \mathsf{note}$**:** Parse $(\mathsf{rk}, \mathsf{rc}, \mathsf{rn}) \leftarrow r$ and $(a_{\mathsf{pk}}, \mathsf{pk}_{\mathsf{enc}}) \leftarrow \mathsf{pk}$. Commit to $(a_{\mathsf{pk}}, \mathsf{rn})$ with randomness rk as commitment com and then commit to $\mathsf{com}, \vec{a}$ with randomness rc. So $\mathsf{note} = \mathsf{Commit}(\mathsf{Commit}(a_{\mathsf{pk}}, \mathsf{rn}; \mathsf{rk}), \vec{a}; \mathsf{rc})$.

$\mathsf{Enc}(\mathsf{pk}, (\vec{a}, r), \xi) \rightarrow C$**:** Parse $(a_{\mathsf{pk}}, \mathsf{pk}_{\mathsf{enc}}) \leftarrow \mathsf{pk}$. Encrypt $(\vec{a}, r)$ with $\mathsf{pk}_{\mathsf{enc}}$ to ciphertext C and return C.

$\mathsf{Receive}(\mathsf{note}, C, \mathsf{sk}) \rightarrow (\vec{a}', r')/\bot$**:** Parse $(a_{\mathsf{sk}}, \mathsf{sk}_{\mathsf{enc}}) \leftarrow \mathsf{sk}$. Decrypt C with $\mathsf{sk}_{\mathsf{enc}}$ to $(\vec{a}', r')$. Parse $(\mathsf{rk}', \mathsf{rc}', \mathsf{rn}') \leftarrow r'$ and verify that these values recreate the commitment

$$\mathsf{note} = \mathsf{Commit}(\mathsf{Commit}(P(a_{\mathsf{sk}}), \mathsf{rn}'; \mathsf{rk}'), \vec{a}'; \mathsf{rc}')$$

If decryption fails or the commitment does not match the note, return $\bot$.

$NulEval(\mathsf{sk}, r) \to \mathsf{nul}$: Parse $(a_{\mathsf{sk}}, \mathsf{sk}_{\mathsf{enc}}) := \mathsf{sk}$ and $(\mathsf{rk}, \mathsf{rc}, \mathsf{rn}) := r$. Evaluate $\mathsf{nul} = \mathsf{PRF}^{\mathsf{sn}}_{a_{\mathsf{sk}}}(\mathsf{rn})$ and return nul.

The completeness and soundness proofs are straightforward: completeness follows by completeness of the PKE, and soundness is ensured by the verification check in the end of $\mathsf{Receive}$.

Theorem 5.3.1 (OTA Binding)**.** *When using a binding commitment scheme* Commit*, the Zerocash commitment scheme is binding according to Definition 5.3.4*

Proof. Let $\mathcal{A}$ return $\mathsf{pk}_0, \vec{a}_0, r_0, \mathsf{pk}_1, \vec{a}_1, r_1$ such that $\mathsf{Gen}(\mathsf{pk}_0, \vec{a}_0, r_0) = \mathsf{Gen}(\mathsf{pk}_1, \vec{a}_1, r_1)$ and $\vec{a}_0 \neq \vec{a}_1$. Parsing $\forall i \in \{0, 1\} : (\mathsf{rk}_i, \mathsf{rc}_i, \mathsf{rn}_i) \leftarrow r_i$, the first condition implies the equality

$$\mathsf{Commit}\big(\mathsf{Commit}(\mathsf{pk}_0, \mathsf{rn}_0; \mathsf{rk}_0), \vec{a}_0; \mathsf{rc}_0\big) = \mathsf{Commit}\big(\mathsf{Commit}(\mathsf{pk}_1, \mathsf{rn}_1; \mathsf{rk}_1), \vec{a}_1; \mathsf{rc}_1\big)$$

but the commitments have different values for $\vec{a}$. This breaks the binding property of the commitment scheme. $\square$

Theorem 5.3.2 (Note Uniqueness)**.** *Zerocash OTA style notes are unique according to Definition 5.3.6*

Proof. Trivially follows from binding of the underlying commitment scheme – since both pk and $\vec{a}$ are commitment messages, finding a collision (even with adversarially chosen randomness) amounts to breaking commitment binding. $\square$

Theorem 5.3.3 (Nullifier Uniqueness)**.** *The Zerocash OTA scheme has unique nullifiers (Definition 5.3.8), if the PRF is secure and the commitment scheme is binding.*

Proof. Let $\mathcal{A}$ return $(\mathsf{sk}_0, r_0, \vec{a}_0, \mathsf{sk}_1, r_1, \vec{a}_1)$ which generates the same note but different nullifiers nul_0 and nul_1. The binding commitment scheme ensures that $(\mathsf{sk}_0, r_0, \vec{a}_0) = (\mathsf{sk}_1, r_1, \vec{a}_1)$. As $\mathsf{NulEval}$ is deterministic, the nullifiers are equal, contradicting our assumption. $\square$

Theorem 5.3.4 (Nullifier Pseudorandomness)**.** *The Zerocash OTA nullifiers are pseudorandom (Definition 5.3.7), if the PRF is secure and the commitment scheme is hiding.*

Proof. The pseudorandomness experiment is exactly the same as standard pseudorandomness, except $\mathcal{A}$ is given (1) a public key, (2) an oracle access to $\mathsf{Receive}(\cdot,\cdot,\mathsf{sk})$. The public key is computed as $\mathsf{PRF}^{\mathsf{addr}}_{a_{\mathsf{sk}}}(0)$, and so by pseudorandomness of this PRF (with a different domain) can be replaced by a random value ψ, so knowledge of an extra random value does not help $\mathcal{A}$ to distinguish. Regarding the $\mathsf{Receive}$ oracle, it performs decryption with an unrelated $\mathsf{sk}_{\mathsf{enc}}$, so it does not interfere with the main pseudorandomness reduction, since this oracle does not use a_{sk} (it uses ψ which we already argued to be random and thus irrelevant). □

Theorem 5.3.5 (Nullifier Collision-Resistance)**.** *Zerocash OTA nullifiers are collision-resistant.*

Proof. Follows directly from the collision-resistance of PRF^{sn} which in practice is instantiated using a collision-resistant hash function. □

Theorem 5.3.6 (OTA Privacy)**.** *Using a hiding commitment scheme and a IND-CCA, key anonymous*[4] *(IK-CCA) encryption scheme, the Zerocash-style OTA is private according to Definition 5.3.5.*

Proof. The proof proceeds in three hops:

1. By commitment hiding we replace $\mathsf{note}^* \leftarrow \mathsf{Commit}(\mathsf{Commit}(a_{\mathsf{pk}_{i_b}},\mathsf{rn};\mathsf{rk}),\vec{a};\mathsf{rc})$ by the commitment to zero: $\mathsf{note}^* \leftarrow \mathsf{Commit}(\mathsf{Commit}(0,0;\mathsf{rk}),0;\mathsf{rc})$. A reduction to commitment hiding is straightforward: we do not need anything else to simulate $\mathcal{O}_{\mathsf{Rcv}}$ to $\mathcal{A}$ when building $\mathcal{B}$ against hiding, because $\mathcal{O}_{\mathsf{Rcv}}$ does not reply to the challenge notes.

2. By IK-CCA we can replace the encryption under pk_{i_b} to encryption under pk_0 always and just ignore i_1, i_0. Note that in the first step we already remove note dependency on pk_{i_b}, so now public keys are only used in construction of C^*. In the reduction to IK-CCA we simulate $\mathcal{O}_{\mathsf{Rcv}}$ (which requires decrypting non-challenge C^*) to $\mathcal{A}$ using IK-CCA decryption oracles. By the end of this game $\mathcal{A}$ always receives $C^* = \mathsf{Enc}(\mathsf{pk}_0,(\vec{a_b},r),\xi)$.

3. Finally, by IND-CCA we replace the encryption of a_b by encryption of 0. To simulate the $\mathcal{O}_{\mathsf{Rcv}}$ oracle to $\mathcal{A}$ we again use the IND-CCA decryption oracle (and we just need one, since we always use the same encryption key pk_0).

[4] See [Bellare et al., 2001], Definition 1.

After the three games what $\mathcal{A}$ sees is $\mathsf{Commit}(\mathsf{Commit}(0, 0; \mathsf{rk}), 0; \mathsf{rc})$ as a note, and $\mathsf{Enc}_{\mathsf{pk}_0}(0; \xi)$ as a ciphertext. Both do not depend on b, and therefore $\mathcal{A}$ wins the final game with probability exactly $1/2$. $\square$

5.4 Zswap Scheme

A Zswap scheme is an extension of an OTA scheme which allows creating (SignTx), merging ($\mathsf{MergeSig}$), and verifying (Verify) transactions that transfer coins between OTA accounts. SignTx takes a pre-transaction ptx as input and produces a transaction signature σ, $\mathsf{MergeSig}$ combines transaction signatures $\sigma_1, \ldots, \sigma_n$, and Verify verifies a signature σ for a transaction tx. A signature σ is viewed separately from its transaction tx (created by $\mathsf{tx} \leftarrow \mathsf{CompleteTx}(\mathsf{ptx})$ defined below), and not contained in it.

Many algorithms will make use of st – the current state of valid previously issued notes. It consists of two parts: $\mathsf{st.MT}$ is the Merkle tree containing notes as leaves, and $\mathsf{st.NF}$ is the set of used nullifiers. Intuitively, when the note is spendable, its commitment should be in $\mathsf{st.MT}$; when the note is spent, its (unique, unlinkable) nullifier goes into $\mathsf{st.NF}$.

5.4.1 Protocol Definition

We first present auxiliary algorithms for Zswap in Fig. 5.2 for creating pre-transactions and transactions. These use the OTA scheme algorithms only in a black-box manner and simplify the exposition when defining the security properties. The first set of functions glues the Zswap and OTA schemes together.

- $\mathsf{BuildPTx}$ constructs inputs for SignTx from I, O instructions and a set of secret keys SK. Inputs I consist of a list of existing (note, C) notes, outputs O consist of a list of public key, value, and type triplets $(\mathsf{pk}^{\mathcal{T}}, a^{\mathcal{T}}, \mathsf{ty}^{\mathcal{T}})$ for creating the output notes. By receiving input notes and generating output notes it produces a pre-transaction information ptx.

- $\mathsf{CompleteTx}$ constructs a transaction tx for Verify from a pre-transaction ptx similar to SignTx.

- $\mathsf{MergeTx}$ creates a new transaction tx by combining a set of existing transactions $\mathsf{tx}_1, \ldots, \mathsf{tx}_n$. This is the non-cryptographic analogue of $\mathsf{MergeSig}$.

- $\mathsf{CheckPTx}$ checks whether pre-transaction is valid w.r.t. st, which is used in the correctness property. It is easy to see that for $\mathsf{ptx} = \mathsf{BuildPTx}(\mathsf{st}, \cdot, \cdot, \cdot)$ we have $\mathsf{CheckPTx}(\mathsf{st}, \mathsf{ptx}) = 1$.
- $\mathsf{TryReceive}$ attempts to "receive" a note note by decrypting its ciphertext C using any of the set SK of available secret keys: if an input can be received with one of the keys it also computes the note's nullifier.
- $\mathsf{CheckBalance}$ is merely an alias that checks that for each type the pre-transaction is balanced, and all the values are within bounds.

Definition 5.4.1. *A Zswap transaction scheme, built on top of an OTA scheme, consists of a tuple of PPT algorithms* $(\mathsf{Setup}, \mathsf{SignTx}, \mathsf{MergeSig}, \mathsf{Verify})$ *defined as follows:*

$\mathsf{Setup}(1^\lambda) \xrightarrow{\$} \mathsf{pp}$ *takes the security parameter* λ *and outputs public parameters* pp *which are implicitly given to all the following algorithms. Setup is called once when a Zswap system is initialized.*

$\mathsf{SignTx}(\mathsf{st}, \mathsf{ptx}) \xrightarrow{\$} \sigma$ *takes a pre-transaction* $\mathsf{ptx} = (\mathcal{S}, \mathcal{T})$ *where*

- $\mathcal{S} = \{(\mathsf{sk}_i^{\mathcal{S}}, \mathsf{note}_i^{\mathcal{S}}, \mathsf{nul}_i, \mathsf{path}_i, (a_i^{\mathcal{S}}, \mathsf{ty}_i^{\mathcal{S}}), r_i^{\mathcal{S}})\}_{i=1}^{|\mathcal{T}|}$ *is a set of inputs with a nullifier* nul_i *corresponding to the* $\mathsf{note}_i^{\mathcal{S}}$*, and stored in the current state at the given path* $\mathsf{st}.\mathsf{MT}[\mathsf{path}_i]$*, secret key* $\mathsf{sk}_i^{\mathcal{S}}$*, amount* $a_i^{\mathcal{S}}$ *and type* $\mathsf{ty}_i^{\mathcal{S}}$ *with input OTA notes' randomness* $r_i^{\mathcal{S}}$*.*
- $\mathcal{T} = \{(\mathsf{pk}_i^{\mathcal{T}}, \mathsf{note}_i^{\mathcal{T}}, (a_i^{\mathcal{T}}, \mathsf{ty}_i^{\mathcal{T}}), r_i^{\mathcal{T}})\}_{i=1}^{|\mathcal{T}|}$ *is a set of (output) notes* $\mathsf{note}_i^{\mathcal{T}}$ *with amount* $a_i^{\mathcal{T}}$*, type* $\mathsf{ty}_i^{\mathcal{T}}$ *and output OTA notes' randomness* $r_i^{\mathcal{T}}$*.*

It outputs a signature σ *as authorization to spend the inputs* $\mathcal{S}$ *on the given outputs* $\mathcal{T}$*.*

$\mathsf{Verify}(\vec{\mathsf{st}}, \mathsf{tx}, \sigma) \to b$ *takes a transaction* tx*, a signature* σ *and returns a bit* b *representing the validity of the transaction w.r.t. the valid history of states* $\vec{\mathsf{st}}$ *(this history may be partial, or contain just one last element[5]).*

$\mathsf{MergeSig}(\{\sigma_i\}_{i=1}^n) \to \sigma$ *takes* n *transaction signatures and generates a combined signature* σ *valid for the union of the transactions. To merge the corresponding transactions* $\mathsf{tx}_1, \ldots, \mathsf{tx}_n$ *together, we use the function* $\mathsf{MergeTx}$ *defined*

[5]See the correctness definition in the next sections.

$\underline{\mathsf{BuildPTx}(\mathsf{st}, I, O, \mathsf{SK})}$:

$\mathcal{S}, \mathcal{T} \leftarrow \emptyset$

for $(\mathsf{note}^{\mathcal{S}}, C^{\mathcal{S}}) \in I$ **do**

  $\mathsf{path} \leftarrow \mathsf{st.MT.getPath}(\mathsf{note}^{\mathcal{S}})$

  assert $\mathsf{path} \neq \bot$

  $(\mathsf{sk}^{\mathcal{S}}, \mathsf{nul}, (a^{\mathcal{S}}, \mathsf{ty}^{\mathcal{S}}), r^{\mathcal{S}}) \leftarrow \mathsf{TryReceive}(\mathsf{note}, C, \mathsf{SK})$

  $\mathcal{S} := \mathcal{S} \cup (\mathsf{sk}^{\mathcal{S}}, \mathsf{note}^{\mathcal{S}}, \mathsf{nul}, \mathsf{path}, (a^{\mathcal{S}}, \mathsf{ty}^{\mathcal{S}}), r^{\mathcal{S}})$

assert $\{\mathsf{nul}_i\}$ are distinct and $\forall i.\ \mathsf{nul}_i \notin \mathsf{st.NF}$

for $(\mathsf{pk}^{\mathcal{T}}, a^{\mathcal{T}}, \mathsf{ty}^{\mathcal{T}}) \in O$ **do** $r^{\mathcal{T}} \xleftarrow{\$} \mathbb{S}, \xi \xleftarrow{\$} \Xi$

  $\mathsf{note}^{\mathcal{T}} \leftarrow \mathsf{OTA.Gen}(\mathsf{pk}^{\mathcal{T}}, (a^{\mathcal{T}}, \mathsf{ty}^{\mathcal{T}}), r^{\mathcal{T}})$

  $C^{\mathcal{T}} \leftarrow \mathsf{OTA.Enc}(\mathsf{pk}^{\mathcal{T}}, ((a^{\mathcal{T}}, \mathsf{ty}^{\mathcal{T}}), r^{\mathcal{T}}), \xi)$

  $\mathcal{T} := \mathcal{T} \cup (\mathsf{pk}^{\mathcal{T}}, \mathsf{note}^{\mathcal{T}}, C^{\mathcal{T}}, (a^{\mathcal{T}}, \mathsf{ty}^{\mathcal{T}}), r^{\mathcal{T}})$

$\mathsf{ptx} \leftarrow (\mathcal{S}, \mathcal{T})$; **assert** $\mathsf{CheckBalance}(\mathsf{ptx}) = 1$

return ptx

$\underline{\mathsf{CheckPTx}(\mathsf{st}, \mathsf{ptx@}(\mathcal{S}, \mathcal{T}))}$:

Parse $\mathcal{S}$ as $\{(\mathsf{sk}_i^{\mathcal{S}}, \mathsf{note}_i^{\mathcal{S}}, \mathsf{nul}_i, \mathsf{path}_i, (a_i^{\mathcal{S}}, \mathsf{ty}_i^{\mathcal{S}}), r_i^{\mathcal{S}})\}_{i=1}^{|\mathcal{S}|}$

Parse $\mathcal{T}$ as $\{(\mathsf{pk}_i^{\mathcal{T}}, \mathsf{note}_i^{\mathcal{T}}, C_i^{\mathcal{T}}, (a_i^{\mathcal{T}}, \mathsf{ty}_i^{\mathcal{T}}), r_i^{\mathcal{T}})\}_{i=1}^{|\mathcal{T}|}$

% $C_i^{\mathcal{T}}$ is left to the receiver to verify

assert $\{\mathsf{nul}_i\}$ are distinct and $\forall i.\ \mathsf{nul}_i \notin \mathsf{st.NF}$

for $i \in [|\mathcal{S}|]$ **do**

  assert $\mathsf{st.MT}[\mathsf{path}_i] = \mathsf{note}_i^{\mathcal{S}}$

  assert $\mathsf{nul}_i = \mathsf{OTA.NulEval}(\mathsf{sk}_i^{\mathcal{S}}, r_i^{\mathcal{S}})$

  assert $\mathsf{note}_i^{\mathcal{S}} = \mathsf{OTA.Gen}(\mathsf{OTA}.P(\mathsf{sk}_i), (a_i^{\mathcal{S}}, \mathsf{ty}_i^{\mathcal{S}}), r_i^{\mathcal{S}})$

for $i \in [|\mathcal{T}|]$ **do assert** $\mathsf{note}_i^{\mathcal{T}} = \mathsf{OTA.Gen}(\mathsf{pk}_i^{\mathcal{T}}, (a_i^{\mathcal{T}}, \mathsf{ty}_i^{\mathcal{T}}), r_i^{\mathcal{T}})$

assert $\mathsf{CheckBalance}(\mathsf{ptx}) = 1$

return 1

$\underline{\mathsf{CompleteTx}(\mathsf{ptx@}(\mathcal{S}, \mathcal{T}))}$:

Parse $\mathcal{S}$ as $\{(\cdot, \cdot, \mathsf{nul}_i, \cdot, (a_i^{\mathcal{S}}, \mathsf{ty}_i^{\mathcal{S}}), \cdot)\}_{i=1}^{|\mathcal{S}|}$

Parse $\mathcal{T}$ as $\{(\cdot, \mathsf{note}_i^{\mathcal{T}}, C_i^{\mathcal{T}}, (a_i^{\mathcal{T}}, \mathsf{ty}_i^{\mathcal{T}}), \cdot)\}_{i=1}^{|\mathcal{T}|}$

for $\mathsf{ty} \in \mathsf{Ty@}(\{\mathsf{ty}_i^{\mathcal{S}}\} \cup \{\mathsf{ty}_i^{\mathcal{T}}\})$ **do**

  $\Delta_{\mathsf{ty}} \leftarrow \sum_{(a_i^{\mathcal{S}}, \mathsf{ty}) \in \mathcal{S}} a_i^{\mathcal{S}} - \sum_{(a_i^{\mathcal{T}}, \mathsf{ty}) \in \mathcal{T}} a_i^{\mathcal{T}}$

return $\left(\{\mathsf{nul}_i\}_{i=1}^{|\mathcal{S}|}, \{\mathsf{note}_i^{\mathcal{T}}, C_i^{\mathcal{T}}\}_{i=1}^{|\mathcal{T}|}, \{\Delta_{\mathsf{ty}}\}_{\mathsf{ty} \in \mathsf{Ty}}\right)$

Figure 5.2: Auxiliary algorithms for Zswap. These only depend on the OTA scheme and do not use Zswap methods such as Verify or SignTx.

$\underline{\mathsf{MergeTx}(\{\mathsf{tx}_j\}_{j=1}^{n})}$:

Parse tx_j as $\left(\{\mathsf{nul}_{j,i}\}_{i=1}^{|\mathcal{S}|_j}, \{(\mathsf{note}^{\mathcal{T}}_{j,i}, C^{\mathcal{T}}_{j,i})\}_{i=1}^{|\mathcal{T}|}, \{\Delta_{j,\mathsf{ty}}\}_{\mathsf{ty}\in\mathsf{Ty}_j}\right)$

assert $\sum|\mathcal{S}_i| + \sum|\mathcal{T}_i| \leq \beta$

return $\left(\mathsf{ToSet}\left(\bigcup_{j,i}\{\mathsf{nul}_{j,i}\}\right), \mathsf{ToSet}\left(\bigcup_{j,i}\{(\mathsf{note}^{\mathcal{T}}_{j,i}, C^{\mathcal{T}}_{j,i})\}\right),\right.$

$\left.\quad\quad \mathsf{ToSet}\left(\left\{\sum_j \Delta_{j,\mathsf{ty}}\right\}_{\mathsf{ty}\in\bigcup_j \mathsf{Ty}_j}\right)\right)$

$\underline{\mathsf{TryReceive}(\mathsf{note}, C, \mathsf{SK})}$:

for $\mathsf{sk} \in \mathsf{SK}$ **do**

  $\mathsf{res} \leftarrow \mathsf{OTA.Receive}(\mathsf{note}, C, \mathsf{sk})$

  if $\mathsf{res} = ((a, \mathsf{ty}), r) \neq \perp$ **then**

    $\mathsf{nul} \leftarrow \mathsf{OTA.NulEval}(\mathsf{sk}, r)$

    return $(\mathsf{sk}, \mathsf{nul}, (a, \mathsf{ty}), r)$

return $\perp$

$\underline{\mathsf{CheckBalance}(\mathsf{ptx@}(\mathcal{S}, \mathcal{T}))}$:

assert $\sum|\mathcal{S}| + \sum|\mathcal{T}| \leq \beta$

for $\mathsf{ty} \in \{\mathsf{ty}^{\mathcal{S}}_i\} \cup \{\mathsf{ty}^{\mathcal{T}}_i\}$ **do**

  assert $\sum_{(a^{\mathcal{S}}_i, \mathsf{ty})\in\mathcal{S}} a^{\mathcal{S}}_i - \sum_{(a^{\mathcal{T}}_i, \mathsf{ty})\in\mathcal{T}} a^{\mathcal{T}}_i \geq 0$

  assert $(\forall i.\, a^{\mathcal{S}}_i < 2^{\alpha}) \wedge (\forall i.\, a^{\mathcal{T}}_i < 2^{\alpha})$

return 1

Figure 5.2: Auxiliary algorithms for Zswap (cont.)

in Fig. 5.2. It just concatenates their input nullifiers, output notes, and sums their Δ_{ty} *for each* ty*. As of the* st*, they must be equal in the transactions that need to be merged, but we can assume that* st *is only updated once an epoch, which is set to be a time interval long enough for transactions to be merged.*

5.4.2 Atomic Swap: A Workflow Example

After presenting all necessary algorithms, we show a small example of how they interact to create an atomic swap transaction between two parties, Alice and Bob. First, the system is created by Setup. Each participant joining, generates their key pair $(\mathsf{sk}, \mathsf{pk})$ with KeyGen. The creation of new assets is delegated to an external consensus mechanism which updates the global state of the system according to an agreed policy. At one point, Alice and Bob have to be the beneficiary of a transaction. They notice this by calling $\mathsf{TryReceive}$ with their secret key sk on every published transaction output (note, C). If they successfully receive a (note, C) to an amount a and type ty, they keep it for when they want to spend it. Let's assume Alice received a note of 10$ and Bob has a note of 10€. Alice now wants euros and Bob dollars. They assume an exchange rate of 7:5 and proceed to generate their pre-transactions ptx_i. Each party calls $\mathsf{BuildPTx}$ with their note as input instruction I. As output instructions O, Alice sends 5€ to her key pk. To make sure that someone fulfills the offer, she creates a change output to herself with only 2$, leaving 1$ as incentive. Bob performs the same. He inputs the 10€ note and generates outputs of 7$ and 5€ to himself.

The resulting pre-transactions are signed by both parties respectively with SignTx to get signatures σ_i. The final transactions tx_i are generated from the pre-transactions ptx_i by $\mathsf{CompleteTx}$ and can be verified against their signature σ_i with Verify. Note that both transactions by themselves cannot be included in the public ledger. Both have a type with a negative balance. Alice's transaction has $\Delta_{€} = -5$ and Bob's transaction has an imbalance of $\Delta_{\$} = -7$. So far Alice and Bob have not communicated. The non-malleability of the transactions allow them to publish their transactions into an exchange pool. Exchange pools may be run globally or with limited access. The first party seeing both transactions recognizes that they are complementary and is able to merge both tx_i together with $\mathsf{MergeTx}$ and their signatures σ_i with $\mathsf{MergeSig}$. As a prize, the merger is allowed to claim the surplus of

1$ paid by Alice. Technically there are now 3 transactions merged. The resulting merged transaction has no imbalance and can be included in the public ledger. Then all parties get their specified outputs and store them for future transactions.

5.4.3 Security Modelling with Support Oracles

The security of our scheme is based on several top-level games, plus the games we define for the OTA scheme. To model potential blockchain executions we need to introduce the oracles which will model note spending that the adversary $\mathcal{A}$ sees, interactively. The relevant oracles, $\mathcal{O}_{\mathsf{KeyGen}}$, $\mathcal{O}_{\mathsf{Spend}}$ and $\mathcal{O}_{\mathsf{Insert}}$ are presented in Fig. 5.3. They are responsible for generating keys, honest transactions and inserting them into the ledger.

$\mathcal{O}_{\mathsf{KeyGen}}$ allows generating honest public and secret keys for further use in $\mathcal{O}_{\mathsf{Spend}}$.

$\mathcal{O}_{\mathsf{Spend}}$ models leakage of honestly generated transactions, including unbalanced "offer" transactions. Each $\mathcal{O}_{\mathsf{Spend}}$ query allows spending some notes, and successful spend logs are recorded in Spent. An adversary can specify any possible state it likes as long as the request is valid with respect to this state. The requirement that O has at least one honest output is a modelling artefact, and is explained further in the "anti-theft" section.

$\mathcal{O}_{\mathsf{Insert}}$ models "recording" balanced transaction in the ledger. Branching is possible inside $\mathcal{O}_{\mathsf{Insert}}$ — $\mathcal{A}$ can append an honest transaction to any of the recorded states, but still one can only spend notes through $\mathcal{O}_{\mathsf{Insert}}$ if their nullifier has not been used in the same branch. Moreover, the st_0 variable is shared between the oracle records as the first "root" state that $\mathcal{A}$ can initialize the oracles with. $\mathcal{O}_{\mathsf{Insert}}$ only accepts states that are eventually linked with this root st_0.

GetLog function is a state maintenance helper. When called on the state st it makes sure that st is a valid progression of the initial state st_0, and returns two values: Ins', which is a log of this state progression (containing transaction history), and $\vec{\mathsf{st}}$ – subset of Ins' containing states only (only state progression history). When GetLog is called in security experiments on st, it implicitly asserts that st is a valid state that was created through the oracles.

$\mathcal{O}_{\mathsf{KeyGen}}()$:

if SK or PK is $\perp$ **then**
    $\mathsf{SK}, \mathsf{PK} \leftarrow \emptyset$
$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}()$
$\mathsf{SK} := \mathsf{SK} \parallel \mathsf{sk}$
$\mathsf{PK} := \mathsf{PK} \parallel \mathsf{pk}$
return pk

$\mathsf{GetLog}(\mathsf{st})$:

$\mathsf{Ins}' \leftarrow [\,];\ \vec{\mathsf{st}} \leftarrow (\mathsf{st})$
while $x@(\mathsf{st}', \mathsf{st}, \cdot, \cdot) \in \mathsf{Ins}$ **do**
    $\mathsf{Ins}' := x \parallel \mathsf{Ins}'; \mathsf{st} := \mathsf{st}'$
    $\vec{\mathsf{st}} = \vec{\mathsf{st}} \parallel \mathsf{st}'$
assert $\mathsf{st} = \mathsf{st}_0$
return $(\mathsf{Ins}', \vec{\mathsf{st}})$

$\mathcal{O}_{\mathsf{Spend}}(\mathsf{st}, I, O)$:

if Spent is $\perp$ **then** $\mathsf{Spent} \leftarrow \emptyset$
assert $|O| > 0 \wedge \exists(\mathsf{pk}, \cdot, \cdot) \in O : \mathsf{pk} \in \mathsf{PK}$
$\mathsf{ptx}@(\mathcal{S}, \mathcal{T}) \leftarrow \mathsf{BuildPTx}(\mathsf{st}, I, O, \mathsf{SK})$
$\sigma \leftarrow \mathsf{SignTx}(\mathsf{st}, \mathsf{ptx})$
Parse $\mathcal{S}$ as $\{(\mathsf{nul}_i, \cdot, \cdot, \cdot)\}_{i=1}^{|\mathcal{S}|}$; $\mathcal{T}$ as $\{(\cdot, \mathsf{note}_j^{\mathcal{T}}, C_j, \cdot, \cdot)\}_{j=1}^{|\mathcal{T}|}$;
$\mathsf{Spent} := \mathsf{Spent} \cup (\mathsf{st}, I, \{\mathsf{nul}_i\}_{i=1}^{|I|}, \{(\mathsf{note}_j^{\mathcal{T}}, C_j)\}_{j=1}^{|\mathcal{T}|}))$
return σ

$\mathcal{O}_{\mathsf{Insert}}(\mathsf{st}, \mathsf{tx}, \sigma)$:

if Ins is $\perp$ **then** $\mathsf{Ins} \leftarrow \emptyset$
if $\mathsf{st}_0 = \perp$ **then**
    assert $\mathsf{st.NF} = \perp$
    $\vec{\mathsf{st}} \leftarrow \{\mathsf{st}\}$
else
    assert $(\mathsf{st}, \cdot, \cdot, \cdot) \in \mathsf{Ins}$
    $(\cdot, \vec{\mathsf{st}}) \leftarrow \mathsf{GetLog}(\mathsf{st})$
$(\mathsf{Nf}@\{\mathsf{nul}_i\}_{i=1}^{|\mathcal{S}|}, \{\mathsf{note}_j^{\mathcal{T}}, \cdot\}_{j=1}^{|\mathcal{T}|}, \{\Delta_{\mathsf{ty}}\}_{\mathsf{ty} \in \mathsf{Ty}}) \leftarrow \mathsf{tx}$
assert $\mathsf{Verify}(\vec{\mathsf{st}}, \mathsf{tx}, \sigma) = 1$ `% Transaction verifies`
assert $\forall \mathsf{ty} \in \mathsf{Ty} : \Delta_{\mathsf{ty}} \geq 0$ `% And is not an offer`
Construct st' s.t.
    $\mathsf{st}'.\mathsf{MT} \leftarrow \mathsf{st.MT.insert}(\{\mathsf{note}_j^{\mathcal{T}}\}_{j=1}^{|\mathcal{T}|})$
    $\mathsf{st}'.\mathsf{NF} \leftarrow \mathsf{st.NF} \cup \mathsf{Nf}$
$\mathsf{Ins} := \mathsf{Ins} \cup (\mathsf{st}, \mathsf{st}', \mathsf{tx}, \sigma)$
if $\mathsf{st}_0 = \perp$ **then** $\mathsf{st}_0 \leftarrow \mathsf{st}$

Figure 5.3: Oracles for the security experiments.

5.4.3.1 Correctness

We start from the basic correctness definition, covering interactions between honest users.

Definition 5.4.2 (Correctness)**.** *A Zswap scheme is correct if OTA is correct and if for all* $\lambda \in \mathbb{N}$, $\mathsf{pp} \in \mathsf{Setup}(1^\lambda)$*, and all PPT* $\mathcal{B}$ *(setup algorithm) it holds that:*

(1) Honestly generated transactions are immediately valid: *For any* $\mathcal{S}, \mathcal{T}, \mathsf{st}$ *such that* $\mathsf{CheckPTx}(\mathsf{st}, (\mathcal{S}, \mathcal{T})) = 1$*, and* $\mathsf{tx} = \mathsf{CompleteTx}(\mathcal{S}, \mathcal{T})$*, and for any signature* $\sigma \leftarrow \mathsf{SignTx}(\mathsf{st}, (\mathcal{S}, \mathcal{T}))$*, it holds that* $\mathsf{Verify}([\mathsf{st}], \mathsf{tx}, \sigma) = 1$*.*

(2) Honestly generated transactions are valid in any future state: *Let* $\mathsf{st}_1, \mathsf{st}_2$ *be any pair of states returned by* $\mathcal{B}^{\mathcal{O}_{\mathsf{KeyGen}}, \mathcal{O}_{\mathsf{Spend}}, \mathcal{O}_{\mathsf{Insert}}}(\mathsf{pp})$*, such that* $(\cdot, \vec{\mathsf{st}}) \leftarrow \mathsf{GetLog}(\mathsf{st}_2)$ *and* $\mathsf{st}_1 \in \vec{\mathsf{st}}$ *(this implies* GetLog *does not fail, and* st_2 *is a valid progression of* st_1*;* st_1 *may be equal to* st_2*). Let* $\mathcal{S}, \mathcal{T}$ *be such that* $\mathsf{CheckPTx}(\mathsf{st}_1, (\mathcal{S}, \mathcal{T})) = 1$*; let* $\mathsf{tx} = \mathsf{CompleteTx}(\mathcal{S}, \mathcal{T})$ *and* $\sigma \leftarrow \mathsf{SignTx}(\mathsf{st}_1, (\mathcal{S}, \mathcal{T}))$*. Then* $\mathsf{Verify}(\vec{\mathsf{st}}, \mathsf{tx}, \sigma) = 1$*.*

(3) Honestly merged valid transactions are again valid: *Let* $n \in \mathbb{N}$*. Let* $\{\mathsf{st}_i\}_{i=1}^n, \mathsf{st} \leftarrow \mathcal{B}^{\mathcal{O}_{\mathsf{KeyGen}}, \mathcal{O}_{\mathsf{Spend}}, \mathcal{O}_{\mathsf{Insert}}}(\mathsf{pp}, n)$ *be a set of states such that* $(\mathsf{st}_i, \mathsf{st})$ *are valid progressions in the same history:* $\vec{\mathsf{st}} \leftarrow \mathsf{GetLog}(\mathsf{st})$ *and* $\forall i.\, \mathsf{st}_i \in \vec{\mathsf{st}}$*. Let there be a set of* $\{(\mathcal{S}_i, \mathcal{T}_i)\}_{i \in [n]}$ *s.t. for all* i*,* $\mathsf{CheckPTx}(\mathsf{st}_i, (\mathcal{S}_i, \mathcal{T}_i)) = 1$*, for each* $\sigma_i \in \mathsf{SignTx}(\mathsf{st}_i, (\mathcal{S}_i, \mathcal{T}_i))$ *and* $\sigma = \mathsf{MergeSig}(\{\sigma_i\}_{i=1}^n)$*, and* $\mathsf{tx} \leftarrow \mathsf{MergeTx}(\{\mathsf{CompleteTx}\,(\mathcal{S}_i, \mathcal{T}_i)\}_{i=1}^n)$*, it holds that* $\mathsf{Verify}(\vec{\mathsf{st}}, \mathsf{tx}, \sigma) = 1$*.*

Regarding the ability to "receive" coins that were sent to the party, the formal completeness definition is part of the OTA scheme, and the fact we receive properly send coins is guaranteed by $\mathsf{BuildPTx}$, which is not part of zswap scheme. At the same time, correctness of ciphertexts is not guaranteed, so it is possible to create an output with an invalid ciphertext, undecryptable by the receiver. In this case, receiver of the coin must consider this transaction factually invalid, as burning the coin that was designated to them.

5.4.4 Security Definitions

Definition 5.4.3 (Zswap Security)**.** *The Zswap scheme is secure if the underlying OTA scheme is secure, and the Zswap scheme satisfies (1) anti-theft, (2) balance, and (3) privacy; properties as defined in this section.*

$\underline{\mathsf{Anti\text{-}Theft}_{\mathcal{A}}(1^\lambda)}$:

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$

$\mathsf{st}^* \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{KeyGen}}, \mathcal{O}_{\mathsf{Spend}}, \mathcal{O}_{\mathsf{Insert}}}(\mathsf{pp})$

$(\mathsf{Ins}', \cdot) \leftarrow \mathsf{GetLog}(\mathsf{st}^*)$

$\textbf{for } (\mathsf{st}, \cdot, \mathsf{tx}, \cdot) \in \mathsf{Ins}' \textbf{ do}$

$\quad (\cdot, (\mathsf{Nf}^{\mathcal{A}}, M^{\mathcal{A}})) \leftarrow \mathsf{SplitTx}(\mathsf{st}, \mathsf{tx})$

$\quad M^{\mathcal{A}\prime} \leftarrow \{(\mathsf{note}, C) \in M^{\mathcal{A}} \mid \mathsf{TryReceive}(\mathsf{note}, C, \mathsf{SK}) \neq \bot\}$

$\quad \textbf{if } \exists(\cdot, \cdot, \mathsf{Nf}, M) \in \mathsf{Spent} : M^{\mathcal{A}\prime} \cap M \neq \emptyset \vee \mathsf{Nf}^{\mathcal{A}} \cap \mathsf{Nf} \neq \emptyset \textbf{ then}$

$\quad\quad \textbf{return } 1$

$\textbf{return } 0$

$\underline{\mathsf{SplitTx}(\mathsf{st}, \mathsf{tx})}$:

$(\mathsf{Nf}@\{\mathsf{nul}_i\}_{i=1}^{|\mathcal{S}|}, M@\{\mathsf{note}_j^{\mathcal{T}}, \cdot\}_{j=1}^{|\mathcal{T}|}, C) \leftarrow \mathsf{tx}$

$\mathsf{tx}_{\mathcal{H}} \leftarrow []$

$\mathsf{Nf}_0 \leftarrow \mathsf{Nf}$

$\textbf{for } (\mathsf{st}', \cdot, \mathsf{Nf}'@\{\mathsf{nul}_i'\}_{i=1}^{|\mathcal{S}'|}, M'@\{(\mathsf{note}_j^{\mathcal{T}\prime}, C_j')\}_{j=1}^{|\mathcal{T}'|}) \in \mathsf{Spent} \textbf{ do}$

$\quad \textbf{if } \mathsf{st}'.\mathsf{MT} = \mathsf{st}.\mathsf{MT} \wedge M' \subset M \wedge \mathsf{Nf}' \subset \mathsf{Nf}_0 \textbf{ then}$

$\quad\quad \mathsf{tx}_{\mathcal{H}} = \mathsf{tx}_{\mathcal{H}} \parallel (\mathsf{Nf}', M')$

$\quad\quad \mathsf{Nf}_0 := \mathsf{Nf}_0 \setminus \mathsf{Nf}'$

$\mathsf{tx}_{\mathcal{A}} = (\mathsf{Nf} \setminus \{\mathsf{Nf}_{\mathcal{H}} \mid (\mathsf{Nf}_{\mathcal{H}}, \cdot) \in \mathsf{tx}_{\mathcal{H}}\},\ M \setminus \{M_{\mathcal{H}} \mid (\cdot, M_{\mathcal{H}}) \in \mathsf{tx}_{\mathcal{H}}\})$

$\textbf{return } (\mathsf{tx}_{\mathcal{H}}, \mathsf{tx}_{\mathcal{A}})$

Figure 5.4: The anti-theft experiment.

We now elaborate on the mentioned top-level security properties.

Anti-Theft and Non-Malleability. The notion of anti-theft we describe here is our main non-malleability notion. Intuitively it says that an adversary $\mathcal{A}$ can only merge transactions, but cannot split honest transactions apart, or modify them in any other way. More concretely, for any honest tx that $\mathcal{A}$ sees, $\mathcal{A}$ cannot submit tx^* which contains any of the (1) input nullifiers of tx; or (2) honest output notes of tx; without merging the whole tx into tx^*. Anti-theft subsumes the even more basic property that $\mathcal{A}$ cannot spend honest notes without doing it honestly (asking honest parties through $\mathcal{O}_{\mathsf{Spend}}$).

The anti-theft game is modelled by allowing $\mathcal{A}$ to interact with the oracles described before: $\mathcal{A}$ can generate new keys, produce unbalanced honest transactions (spend), and submit (insert) transactions (which we want to prove to be only honest, dishonest, or merges of the two types). The adversary then returns a state st^* which is examined. The challenger searches through the log of inserted transactions. The adversary wins if a transaction in the log contains an incomplete part of (an honest) transaction earlier returned by $\mathcal{O}_{\mathsf{Spend}}$ to $\mathcal{A}$, but *used* only partially. To do that, it calls a sub-function $\mathsf{SplitTx}$ which locates complete "honest subtransactions" returned by $\mathcal{O}_{\mathsf{Spend}}$ in tx, and returns these honest sub-transactions as first argument, and the remaining parts of tx^*. If the challenger recognizes nullifiers or output notes in this remaining part, that were created in $\mathcal{O}_{\mathsf{Spend}}$, it means that $\mathcal{A}$ managed to deconstruct it, changing the output of $\mathcal{O}_{\mathsf{Spend}}$, or stealing an honest nullifier.

Definition 5.4.4 (Anti-Theft)**.** *A Zswap scheme is protecting against theft, if for any* $\lambda \in \mathbb{N}$ *and any PPT adversary* $\mathcal{A}$*,* $\Pr[\text{Anti-Theft}_{\mathcal{A}}(1^\lambda) = 1] = \mathsf{negl}(\lambda)$ *where the* $\text{Anti-Theft}_{\mathcal{A}}(1^\lambda)$ *game is defined in Fig. 5.4.*

Essentially, anti-theft captures non-malleability of output notes and their ciphertexts, which is in practice guaranteed by NIZK SE (that implies instance binding). If $\mathcal{A}$ can maul an output (e.g. change its value, or destination) and still submit the tx to $\mathcal{O}_{\mathsf{Insert}}$ successfully, it wins the game, since $\mathsf{SplitTx}$ will not locate tx as being in Spent, and thus the game challenger $\mathcal{C}$ will catch $\mathcal{A}$ spending a nullifier that was in Spent but not located by $\mathsf{SplitTx}$.

On the more basic non-malleability side, $\mathcal{A}$ cannot construct transactions spending honest notes. Assume that $\mathcal{A}$ constructs tx_0 sending a note to an honest party ($\mathsf{pk} \in \mathsf{PK}$), and then succeeds to spend it using tx^*, without using $\mathcal{O}_{\mathsf{Spend}}$. If this is possible, it means $\mathcal{A}$ presented a nullifier inside tx^* corresponding to the note — then $\mathcal{A}$ could also make a single query to $\mathcal{O}_{\mathsf{Spend}}$ instructing to spend the same note to elsewhere, and ignore the result of $\mathcal{O}_{\mathsf{Spend}}$, submitting tx^* as planned. This would "mark" the nullifier, and trigger winning condition of the anti-theft game.

Modelling details. The requirement that O in $\mathcal{O}_{\mathsf{Spend}}$ contains at least one honest output is needed so that $\mathsf{SplitTx}$ can uniquely identify honest sub-transactions[6]. In other words, note uniqueness is necessary for the anti-theft game to make sense.

[6]Without this requirement, and without adding extra marker information to notes, identifying honest sub-transactions within $\mathsf{SplitTx}$ would take exponential time.

Without it the game would produce false positives: $\mathcal{A}$ could trick $\mathsf{SplitTx}$ into not recognising some honest sub-transactions, even though the tx^* submitted by $\mathcal{A}$ is perfectly normal. This modelling artefact does not limit $\mathcal{A}$ from creating unbalanced input-only transactions, since $\mathcal{A}$ can still request to include a single zero-valued output.

No similar restrictions are put on the inputs, and they can be null; in other words, $\mathcal{A}$ can instruct to generate an offer with a single honest output. However, it is easy to see that $\mathcal{A}$ cannot trigger the winning condition with this input — since the output notes are unique, it is guaranteed that this output note will be detected in $\mathsf{SplitTx}$. This does not rule out the possibility of $\mathcal{A}$ using this output in other way to break the property.

The set Nf_0 in $\mathsf{SplitTx}$ is needed since without it honest sub-transactions can be counted twice. E.g. let $\mathsf{tx}_0, \mathsf{tx}_1$ be two honest transactions with the same input nullifiers Nf, but completely different outputs M_0, M_1 (each containing at least one honest output note). Without Nf_0, for tx containing Nf and $M_0 \cup M_1$, $\mathsf{SplitTx}$ will detect both $\mathsf{tx}_0, \mathsf{tx}_1$, and the winning condition will not be triggered. We, on the other hand, want anti-theft to prevent this: with Nf_0 one of tx_i will be considered honest, and the honest output note in tx_{1-i} will trigger the anti-theft winning condition. We do not need to similarly count M since output notes are unique — adding $M = M \setminus M'$ into $\mathsf{SplitTx}$ simply does not change the behaviour of the game.

Finally, it is critical to filter $M^{\mathcal{A}\prime}$ from $M^{\mathcal{A}}$. Otherwise, $\mathcal{A}$ would be able to trivially win the game by triggering $M^{\mathcal{A}} \cap M \neq 0$ in the following way:

1. $\mathcal{A}$, through $\mathcal{O}_{\mathsf{Spend}}$, requests an honest spend to an adversarial public key $\mathsf{pk}^{\mathcal{A}}$;
2. $\mathcal{A}$ obtains the proof and corresponding note^*, receives it, and creates a completely different, adversarial transaction tx^* on its own, where note^* is an output;
3. the game will not find any honest subtransaction in tx^* and thus $\mathsf{note}^* \in M \cap M^{\mathcal{A}}$.

Balance. The balance property says that transactions distribute underlying coins "properly" — that is, the adversarial balance per type only changes *predictably*, by the adversary receiving or sending these coins. In particular, the balance game forbids malicious conversion between asset types, coin forging, and double spending — both within a single transaction, and for any history of adversarial interaction with the ledger.

Formally, the property is modelled as a game (Fig. 5.5) of $\mathcal{A}$ "against" the honest parties, in which $\mathcal{A}$ has to prove that it has more coins that it could have obtained honestly. The challenger lets $\mathcal{A}$ interact with the oracles, and asks $\mathcal{A}$ to present two sets of values: I_0 and I containing notes together with their secrets. Unlike in other games, st_0 in balance must contain only adversarial coins, and I_0 must be all the unspent corresponding notes. The other set I is the set of unspent adversarial coins from the new state st^* that (supposedly, as $\mathcal{A}$ claims) contain illicitly produced coins, breaking the global balance condition. This balance condition is computed next in the following manner. Initially, v_0 is set to the sum of all values in I_0, and $v_{\mathcal{A}}$ as a sum of all values in I (how much per type $\mathcal{A}$ owned in the beginning of the game, and how much it owns in the end). Then, by traversing the history of transactions from st_0 to st^*, the game computes the following two values:

1. $v_{\mathcal{H}-}$ is the sum of all honest inputs in transactions; and
2. $v_{\mathcal{H}+}$ is the total coins received by honest parties (located in transaction outputs).

The final condition checks that $\mathcal{A}$ cannot show in st^* more coins than: (1) it had in st_0, plus (2) what was sent by honest parties, minus what was received by honest parties. This last difference is non-positive (because I_0 is all the system coins), and importantly the balance of $\mathcal{A}$ per type is "tied" to the balance of honest parties, so no extra coins can be produced.

Note that balance is defined in conjunction with anti-theft, since it uses $\mathsf{SplitTx}$ which it assumes to work correctly (according to the anti-theft definition). In practice it also makes sense to see them together: balance and anti-theft jointly guarantee that adversary cannot receive more than honestly, *given* that it can combine its transactions with dishonest ones. Theft guarantees that $\mathcal{A}$ can only merge transactions; balance guarantees that these transactions never break the total balance of coins in the system.

In particular, the balance property guarantees the following:

1. Total spendable amount of coins per type is constant in the system: if there

$\underline{\mathsf{Balance}_{\mathcal{A},\mathsf{Ext}_\mathcal{A}}(1^\lambda)}$:

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
$(\mathsf{st}^*, I_0, I) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{KeyGen}},\mathcal{O}_{\mathsf{Spend}},\mathcal{O}_{\mathsf{Insert}}}(\mathsf{pp})$
assert $\{\mathsf{note}_i\} \in I_0$ are distinct, and $\mathsf{st}_0.\mathsf{MT}$ contains *only* these notes.
assert $\{\mathsf{note}_i\} \in I$ are distinct, and are in $\mathsf{st}^*.\mathsf{MT}$
for $(\mathsf{note}_i, \mathsf{nul}, (a, \mathsf{ty}), \mathsf{sk}, r) \in I \cup I_0$ **do**
  assert $\mathsf{note}_i = \mathsf{Gen}(P(\mathsf{sk}), (a, \mathsf{ty}), r) \wedge \mathsf{nul} = \mathsf{NulEval}(\mathsf{sk}, r)$
for $(\cdot, \mathsf{nul}, (a, \mathsf{ty}), \cdot, \cdot) \in I_0$ **do**
  assert $\mathsf{nul} \notin \mathsf{st}_0.\mathsf{NF}$
  $v_0[\mathsf{ty}] := v_0[\mathsf{ty}] + a$
for $(\cdot, \mathsf{nul}, (a, \mathsf{ty}), \cdot, \cdot) \in I$ **do**
  assert $\mathsf{nul} \notin \mathsf{st}^*.\mathsf{NF}$
  $v_\mathcal{A}[\mathsf{ty}] := v_\mathcal{A}[\mathsf{ty}] + a$
$v_{\mathcal{H}-}, v_{\mathcal{H}+} \leftarrow (\mathsf{ty} \mapsto 0)$ `% map with default value` 0
$(\mathsf{Ins}', \cdot) \leftarrow \mathsf{GetLog}(\mathsf{st}^*)$
for all $(\mathsf{st}, \cdot, \mathsf{tx}, \sigma) \in \mathsf{Ins}'$ **do**
  $(\{\mathsf{nul}_i\}_{i=1}^{|\mathcal{S}|}, \{\mathsf{note}_j^{\mathcal{T}}, \cdot\}_{j=1}^{|\mathcal{T}|}, \emptyset) \leftarrow \mathsf{tx}$
  $(\mathsf{tx}_\mathcal{H}, \cdot) \leftarrow \mathsf{SplitTx}(\mathsf{st}, \mathsf{tx})$
  for $(\mathsf{Nf}, M) \in \mathsf{tx}_\mathcal{H}$ **do**
    Find $(\cdot, I, \mathsf{Nf}, M) \in \mathsf{Spent}$
    for $(\mathsf{note}', C') \in I$ **do**
      $(\cdot, (a, \mathsf{ty}), \cdot) \leftarrow \mathsf{TryReceive}(\mathsf{note}', C', \mathsf{SK})$
      $v_{\mathcal{H}-}[\mathsf{ty}] := v_{\mathcal{H}-}[\mathsf{ty}] + a$ `% Honestly spent inputs`
  for $(\mathsf{note}', C') \in M$ **where** $(\cdot, M) \in \mathsf{tx}$ **do**
    $\mathsf{res} \leftarrow \mathsf{TryReceive}(\mathsf{note}', C', \mathsf{SK})$
    if $\mathsf{res} = (\cdot, (a, \mathsf{ty}), \cdot) \neq \bot$ **then**
      $v_{\mathcal{H}+}[\mathsf{ty}] := v_{\mathcal{H}+}[\mathsf{ty}] + a$ `% Outputs to honest parties`
if $\exists \mathsf{ty} : v_\mathcal{A}[\mathsf{ty}] > v_0[\mathsf{ty}] + v_{\mathcal{H}-}[\mathsf{ty}] - v_{\mathcal{H}+}[\mathsf{ty}]$ **then**
  return 1
else return 0

Figure 5.5: The balance experiment

are more coins in the system than v_0, and they are spendable, $\mathcal{A}$ can transfer them to itself, and present them in I, thus breaking $v_{\mathcal{A}} > v_0$. Even if these coins belong to the honest users, and thus $v_{\mathcal{H}-} - v_{\mathcal{H}+} > 0$ (which should not happen), $\mathcal{A}$ can always create a transaction transferring these coins to itself using $\mathcal{O}_{\mathsf{Spend}}$.

2. Every transaction is balanced per type, that is it does not produce more coins than it consumes. This means no conversion between types, and no coin forging. It follows from (1), since an unbalanced transaction would create more funds in the state st' that follows this transaction. So $\mathcal{A}$ could just present state st' and win the balance game.

3. Double spending is forbidden. Because we have balancing, it is not possible to spend a note in such a way so that its funds disappear from the counter of the total coins. This means that double spending would produce total coin imbalance, which is forbidden.

Definition 5.4.5 (Balance)**.** *A Zswap scheme is balanced if for all PPT adversaries $\mathcal{A}$ there exists a PPT extractor $\mathrm{Ext}_{\mathcal{A}}$ such that $\Pr[\mathsf{Balance}_{\mathcal{A},\mathsf{Ext}_{\mathcal{A}}}(1^\lambda) = 1] = \mathsf{negl}(\lambda)$ with the game defined in Fig. 5.5.*

Privacy. The privacy game captures secrecy of coin transfers by means of an indistinguishability experiment. To model the game we will first introduce the notion of a transaction instruction tree T. Such a variable width tree has as its leaf i either

1. the instructions to construct an honest transaction $(\{\mathsf{note}_{i,j,C_{i,j}}\}_{j=1}^{|\mathcal{S}|_i}, \{(\mathsf{pk}_{i,j}, a^{\mathcal{T}}_{i,j}, \mathsf{ty}^{\mathcal{T}}_{i,j})\}_{j=1}^{|\mathcal{T}|_i})$, similarly to the input to $\mathcal{O}_{\mathsf{Spend}}$, or

2. a fully adversarial transaction $(\mathsf{tx}_i, \sigma_i)$.

Its intermediate leaves are empty, and merely represent how children transactions must be merged.

We will also need to decide what a transaction resulting from a merge according to T leaks. We formalize this notion by defining the tree equivalency relation, formally in Fig. 5.6 as follows: $\mathsf{EquivTree}(T_0, T_1) = 1$ if the following conditions hold:

1. The imbalance of amounts in each type is equal (B_1).

2. The number of input notes is equal (published nullifiers), and the number of output notes is equal. This applies to honest ($B_2^{\mathcal{S}} \wedge B_2^{\mathcal{T}}$) leaves, as the same

$\underline{\mathsf{Privacy}^b_{\mathcal{A}}(1^\lambda)}$:

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
$(\mathsf{st}, T_0, T_1) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{KeyGen}}, \mathcal{O}_{\mathsf{Spend}}, \mathcal{O}_{\mathsf{Insert}}}(\mathsf{pp})$
assert $(\cdot, \vec{\mathsf{st}}) \leftarrow \mathsf{GetLog}(\mathsf{st})$ $\quad\%\ \vec{\mathsf{st}}[1] = \mathsf{st}$
$\mathsf{tx}_0 \leftarrow \mathsf{EvalTree}(\vec{\mathsf{st}}, T_0); \mathsf{tx}_1 \leftarrow \mathsf{EvalTree}(\vec{\mathsf{st}}, T_1)$
assert $\mathsf{tx}_0 \neq \bot \wedge \mathsf{tx}_1 \neq \bot \wedge \mathsf{EquivTree}(\mathsf{st}, T_0, T_1) = 1$
$b' \leftarrow \mathcal{A}(\mathsf{tx}_b)$
return b'

Figure 5.6: Transaction privacy experiment. The helper functions EvalTree and EquivTree are defined in Fig. 5.7.

property implicitly holds for adversarial leaves due to the next check (B_3).

3. Malicious offers need to be the same in both trees, but maybe not at same positions (B_3).

4. For all the honest nullifiers Nf that $\mathcal{A}$ receives via $\mathcal{O}_{\mathsf{Spend}}$, if any of the related notes are included in the honest leaves of a tree, these notes must be included in both trees, (B_4).

5. The adversarial output instructions of honest leaves are the same in both trees (B_5).

The privacy notion itself asks $\mathcal{A}$ to present two equivalent trees, and then builds a single transaction for each tree which both must not fail. It returns one merged transaction. The adversary wins the game if it can decide which tree was used.

To illustrate the notion with a single example, imagine trees that contains a single leaf node, in the first case spending a single note with X coins, sending 1 to Alice and $X - 1$ to Bob, and in the second case spending a single note of Y coins, sending $Y - 2$ to Alice and 2 to Bob. Such trees are equivalent according to our definition, and thus $\mathcal{A}$ should not be able to decide which transaction is produced by which tree.

Definition 5.4.6 (Transaction Privacy). *A Zswap scheme has private transactions, if for all PPT adversaries $\mathcal{A}$ it holds that:*

$$\left|\Pr[\mathsf{Privacy}^0_{\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathsf{Privacy}^1_{\mathcal{A}}(1^\lambda) = 1]\right| = \mathsf{negl}(\lambda)$$

with $\mathsf{Privacy}^b_{\mathcal{A}}(1^\lambda)$ *defined in Fig. 5.6.*

$\underline{\mathsf{EvalTree}(\vec{\mathsf{st}}, T)}$:

for all $\mathsf{leaf}_i \in T$ **do**
  if $\mathsf{leaf}_i = (\mathsf{tx}, \sigma)$ **then** % Validate adversarial txs
    assert $\mathsf{Verify}(\vec{\mathsf{st}}, \mathsf{tx}, \sigma) = 1$
  if $\mathsf{leaf}_i = (I, O)$ **then** % Replace instruction leaves
    $\mathsf{ptx} \leftarrow \mathsf{BuildPTx}(\vec{\mathsf{st}}[1], I, O, \mathsf{SK})$ % with real txs
    $\mathsf{tx}_i \leftarrow \mathsf{CompleteTx}(\mathsf{ptx})$
    $\sigma_i \leftarrow \mathsf{SignTx}(\vec{\mathsf{st}}[1], \mathsf{ptx})$
    Replace leaf_i by $(\mathsf{tx}_i, \sigma_i)$ in T
% Fold T into a single root node by merging
while $\exists$ node N in T with only $\{(\mathsf{tx}_i, \sigma_i)\}_{i=1}^{c}$ as children **do**
  Replace node with merged transactions of its children:
  $N \leftarrow (\mathsf{MergeTx}(\{\mathsf{tx}_i\}_{i=1}^{c}, \mathsf{MergeSig}(\{\sigma_i\}_{i=1}^{c}))$
  Remove its children
return T

$\underline{\mathsf{EquivTree}(\mathsf{st}, T_0, T_1)}$:

Parse each $\mathsf{leaf}_{b,i}$ of each T_b as either an adversarial leaf $\mathsf{leaf}^{\mathcal{A}}$:
  $(\mathsf{tx}_{b,i}@(\{\mathsf{nul}_{b,i,j}\}_{j=1}^{|\mathcal{S}|_{b,i}}, \{(\mathsf{note}^{\mathcal{T}}_{b,i,j}, C^{\mathcal{T}}_{b,i,j})\}_{j=1}^{|\mathcal{T}|_{b,i}}, \{\Delta_{b,i,\mathsf{ty}}\}_{\mathsf{ty}\in\mathsf{Ty}_{b,i}}), \sigma_{b,i}) \leftarrow \mathsf{leaf}^{\mathcal{A}}_{b,i}$
 or an honest leaf $\mathsf{leaf}^{\mathcal{H}}$:
  $(\{(\mathsf{note}^{\mathcal{S}}_{b,i,j}, C^{\mathcal{S}}_{b,i,j})\}_{j=1}^{|\mathcal{S}|_{b,i}}, \{(\mathsf{pk}_{b,i,j}, a^{\mathcal{T}}_{b,i,j}, \mathsf{ty}^{\mathcal{T}}_{b,i,j})\}_{j=1}^{|\mathcal{T}|_{b,i}}) \leftarrow \mathsf{leaf}^{\mathcal{H}}_{b,i}$
for $b \in \{0,1\}, \mathsf{leaf}^{\mathcal{H}}_i \in T_b$ **do**
  for $j \in [|\mathcal{T}_{b,i}|]$ **do**
    $(\mathsf{nul}_{b,i,j}, (a^{\mathcal{S}}_{b,i,j}, \mathsf{ty}^{\mathcal{S}}_{b,i,j}), r_{b,i,j}) \leftarrow \mathsf{TryReceive}(\mathsf{note}^{\mathcal{S}}_{b,i,j}, C^{\mathcal{S}}_{b,i,j}, \mathsf{SK})$
  for $\mathsf{ty} \in \{\mathsf{ty}^{\mathcal{S}}_{b,i,j}\}_{j=1}^{|\mathcal{T}|_{b,i}}$ **do**
    $\Delta_{b,i,\mathsf{ty}} \leftarrow \sum_{j:\mathsf{ty}_{b,i,j}=\mathsf{ty}} a^{\mathcal{S}}_{b,i,j} - \sum_{j:\mathsf{ty}_{0,i,j}=\mathsf{ty}} a^{\mathcal{T}}_{0,i,j}$
$B_1 \leftarrow \forall \mathsf{ty} : \sum_{\mathsf{leaf}^{\mathcal{H}}_i \in T_0} \Delta_{0,i,\mathsf{ty}} = \sum_{\mathsf{leaf}^{\mathcal{H}}_i \in T_1} \Delta_{1,i,\mathsf{ty}}$
$B_2^{\mathcal{S}} \leftarrow \sum_{\mathsf{leaf}^{\mathcal{H}}_i \in T_0} |\mathcal{S}|_{0,i} = \sum_{\mathsf{leaf}^{\mathcal{H}}_i \in T_1} |\mathcal{S}|_{1,i}$
$B_2^{\mathcal{T}} \leftarrow \sum_{\mathsf{leaf}^{\mathcal{H}}_i \in T_0} |\mathcal{T}|_{0,i} = \sum_{\mathsf{leaf}^{\mathcal{H}}_i \in T_1} |\mathcal{T}|_{1,i}$
$B_3 \leftarrow \bigcup_{\mathsf{leaf}^{\mathcal{A}}_i \in T_0} \{(\mathsf{tx}_{0,i}, \sigma_{0,i})\} = \bigcup_{\mathsf{leaf}^{\mathcal{A}}_i \in T_1} \{(\mathsf{tx}_{1,i}, \sigma_{1,i})\}$
$\mathsf{Nf} \leftarrow \bigcup_{(\cdot,\cdot,\mathsf{Nf}_i,\cdot)\in\mathsf{Spent}} \mathsf{Nf}_i$
$B_4 \leftarrow \mathsf{Nf} \cap (\bigcup_{\mathsf{leaf}^{\mathcal{H}}_i \in T_0, j} \mathsf{nul}_{0,i,j}) = \mathsf{Nf} \cap (\bigcup_{\mathsf{leaf}^{\mathcal{H}}_i \in T_1, j} \mathsf{nul}_{1,i,j})$
for $b \in \{0,1\}, \mathsf{leaf}^{\mathcal{H}}_i \in T_b$ **do** $O^{\mathcal{A}}_b \leftarrow \bigcup_{\mathsf{pk}_{b,i,j} \notin \mathsf{PK}} (\mathsf{pk}_{b,i,j}, a^{\mathcal{S}}_{b,i,j}, \mathsf{ty}^{\mathcal{S}}_{b,i,j})$
$B_5 \leftarrow O^{\mathcal{A}}_0 = O^{\mathcal{A}}_1$
return $B_1 \wedge B_2^{\mathcal{S}} \wedge B_2^{\mathcal{T}} \wedge B_3 \wedge B_4 \wedge B_5$

Figure 5.7: Helper functions for the transaction privacy experiment (Fig. 5.6).

5.5 The Zswap Protocol

We present the Zswap protocol in Fig. 5.8. It extends the OTA scheme (constructed in Section 5.3) and additionally utilizes a sparse homomorphic commitment scheme (sparse Pedersen): $\mathsf{SHC} = (\mathsf{ComSetup}, \mathsf{Commit})$. and NIZKs: $\mathsf{NIZK} = (\mathsf{NIZK.Setup}, \mathsf{NIZK.Prove}, \mathsf{NIZK.Verify}, \mathsf{NIZK.Sim})$.

The high-level idea is to create a transaction which has separate inputs and outputs handled by the OTA scheme. A transaction links them together through SHC commitments. Each input and output has a corresponding SHC commitment with an equal amount and type. This equivalency is assured by an NIZK for each input and each output. The output NIZK additionally assure non-malleability for the note ciphertext. The input NIZKs enforce that the transaction creator possessed the secret key to authorize the spending and prove that the published nullifier is correct. This is captured by two NIZK languages.

The first one authenticates a valid spend — it says that the (rerandomized) SHC commitment $\mathsf{com}^{\mathcal{S}}$ is well-formed and contains the same value and type as a note in the Merkle tree of state st with the given nullifier nul. To prevent overflows in the homomorphic commitments, we include a range proof where α is chosen small enough in relation to the group order (maximum inputs and outputs times $2^\alpha < |\mathbb{G}|$). Thereby we assume integer amounts in subsequent arguments.

$$
\begin{aligned}
\mathcal{L}^{\mathsf{spend}} = \Big\{ \big(\mathsf{st}, \mathsf{nul}, \mathsf{com}^{\mathcal{S}}\big) \mid \exists (\mathsf{path}, \mathsf{note}, \mathsf{sk}^{\mathcal{S}}, a^{\mathcal{S}}, \mathsf{ty}^{\mathcal{S}}, r^{\mathcal{S}}, \mathsf{rc}^{\mathcal{S}}) :& \\
\mathsf{st.MT}[\mathsf{path}] = \mathsf{note} \wedge& \\
(\mathsf{note}, \mathsf{nul}; \mathsf{sk}^{\mathcal{S}}, (a^{\mathcal{S}}, \mathsf{ty}^{\mathcal{S}}), r^{\mathcal{S}}) \in \mathcal{L}^{\mathsf{nul}} \wedge& \\
(\mathsf{note}; \mathsf{OTA}.P(\mathsf{sk}^{\mathcal{S}}), (a^{\mathcal{S}}, \mathsf{ty}^{\mathcal{S}}), r^{\mathcal{S}}) \in \mathcal{L}^{\mathsf{open}} \wedge& \\
\mathsf{com}^{\mathcal{S}} = \mathsf{Commit}(\mathsf{ty}^{\mathcal{S}}, a^{\mathcal{S}}; \mathsf{rc}^{\mathcal{S}}) \wedge a^{\mathcal{S}} \in [2^\alpha] \Big\}&
\end{aligned}
$$

The second language $\mathcal{L}^{\mathsf{output}}$ is even simpler. It claims that the two output commitments, the real (which is contained inside the output note) and the randomized one, contain the same value of the same type.

$$
\begin{aligned}
\mathcal{L}^{\mathsf{output}} = \Big\{ (\mathsf{note}^{\mathcal{T}}, C^{\mathcal{T}}, \mathsf{com}^{\mathcal{T}}) \mid \exists (\mathsf{note}, \mathsf{pk}^{\mathcal{T}}, a^{\mathcal{T}}, \mathsf{ty}^{\mathcal{T}}, r^{\mathcal{T}}, \mathsf{rc}^{\mathcal{T}}) :& \\
(\mathsf{note}; \mathsf{pk}, (a^{\mathcal{T}}, \mathsf{ty}^{\mathcal{T}}), r^{\mathcal{T}}) \in \mathcal{L}^{\mathsf{open}} \wedge& \\
\mathsf{com}^{\mathcal{T}} = \mathsf{Commit}(\mathsf{ty}^{\mathcal{T}}, a^{\mathcal{T}}; \mathsf{rc}^{\mathcal{T}}) \wedge a^{\mathcal{T}} \in [2^\alpha] \Big\}&
\end{aligned}
$$

$\underline{\mathsf{Setup}(1^\lambda)}$:

$\mathsf{pp}_{\mathsf{SHC}} \leftarrow \mathsf{ComSetup}(1^\lambda)$; $\mathsf{pp}_{\mathsf{OTA}} \leftarrow \mathsf{OTA.Setup}(1^\lambda)$

$\mathsf{pp}_{\mathsf{spend}} \leftarrow \mathsf{NIZK}[\mathcal{L}^{\mathsf{spend}}].\mathsf{Setup}(1^\lambda)$; $\mathsf{pp}_{\mathsf{output}} \leftarrow \mathsf{NIZK}[\mathcal{L}^{\mathsf{output}}].\mathsf{Setup}(1^\lambda)$

return $\mathsf{pp} := (\mathsf{pp}_{\mathsf{SHC}}, \mathsf{pp}_{\mathsf{OTA}}, \mathsf{pp}_{\mathsf{spend}}, \mathsf{pp}_{\mathsf{output}})$

$\underline{\mathsf{SignTx}(\mathsf{st}, \mathsf{ptx@}(\mathcal{S}, \mathcal{T}))}$:

Parse $\mathcal{S}$ as $\{(\mathsf{sk}_i^{\mathcal{S}}, \mathsf{note}_i^{\mathcal{S}}, \mathsf{nul}_i, \mathsf{path}_i, (a_i^{\mathcal{S}}, \mathsf{ty}_i^{\mathcal{S}}), r_i^{\mathcal{S}})\}_{i=1}^{|\mathcal{S}|}$

Parse $\mathcal{T}$ as $\{(\mathsf{pk}_i, \mathsf{note}_i^{\mathcal{T}}, C_i^{\mathcal{T}}, (a_i^{\mathcal{T}}, \mathsf{ty}_i^{\mathcal{T}}), r_i^{\mathcal{T}})\}_{i=1}^{|\mathcal{T}|}$

$\{\mathsf{rc}_i^{\mathcal{S}} \xleftarrow{\$} \mathbb{R}, \mathsf{com}_i^{\mathcal{S}} \leftarrow \mathsf{Commit}(\mathsf{ty}_i^{\mathcal{S}}, a_i^{\mathcal{S}}; \mathsf{rc}_i^{\mathcal{S}})\}_{i=1}^{|\mathcal{S}|}$ `% Rerandomized input &`

$\{\mathsf{rc}_i^{\mathcal{T}} \xleftarrow{\$} \mathbb{R}, \mathsf{com}_i^{\mathcal{T}} \leftarrow \mathsf{Commit}(\mathsf{ty}_i^{\mathcal{T}}, a_i^{\mathcal{T}}; \mathsf{rc}_i^{\mathcal{T}})\}_{i=1}^{|\mathcal{T}|}$ `% output commitments`

$\{\mathsf{x}_i^{\mathcal{S}} := (\mathsf{st}, \mathsf{nul}_i, \mathsf{com}_i^{\mathcal{S}})\}_{i=1}^{|\mathcal{S}|}; \mathsf{w}_i^{\mathcal{S}} := (\mathsf{path}_i, \mathsf{sk}_i^{\mathcal{S}}, a_i^{\mathcal{S}}, \mathsf{ty}_i^{\mathcal{S}}, r_i^{\mathcal{S}}, \mathsf{rc}_i^{\mathcal{S}})\}_{i=1}^{|\mathcal{S}|}$

$\{\pi_i^{\mathcal{S}} \leftarrow \mathsf{NIZK}[\mathcal{L}^{\mathsf{spend}}].\mathsf{Prove}(\mathsf{x}_i^{\mathcal{S}}, \mathsf{w}_i^{\mathcal{S}})\}_{i=1}^{|\mathcal{S}|}$

$\{\mathsf{x}_i^{\mathcal{T}} := (\mathsf{note}_i^{\mathcal{T}}, C_i^{\mathcal{T}}, \mathsf{com}_i^{\mathcal{T}}); \mathsf{w}_i^{\mathcal{T}} := (\mathsf{pk}_i, a_i^{\mathcal{T}}, \mathsf{ty}_i^{\mathcal{T}}, r_i^{\mathcal{T}}, \mathsf{rc}_i^{\mathcal{T}})\}_{i=1}^{|\mathcal{T}|}$

$\{\pi_i^{\mathcal{T}} \leftarrow \mathsf{NIZK}[\mathcal{L}^{\mathsf{output}}].\mathsf{Prove}(\mathsf{x}_i^{\mathcal{T}}, \mathsf{w}_i^{\mathcal{T}})\}_{i=1}^{|\mathcal{T}|}$

return $(\mathsf{ToSet}(\{(\pi_i^{\mathcal{S}}, \mathsf{com}_i^{\mathcal{S}})\}_1^{|\mathcal{S}|}), \mathsf{ToSet}(\{(\pi_i^{\mathcal{T}}, \mathsf{com}_i^{\mathcal{T}})\}_1^{|\mathcal{T}|}), \sum_{i=1}^{|\mathcal{S}|} \mathsf{rc}_i^{\mathcal{S}} - \sum_{i=1}^{|\mathcal{T}|} \mathsf{rc}_i^{\mathcal{T}})$

$\underline{\mathsf{Verify}(\vec{\mathsf{st}}, \mathsf{tx}, \sigma)}$:

Parse tx as $\left(\{\mathsf{nul}_i\}_{i=1}^{|\mathcal{S}|}, \{(\mathsf{note}_i^{\mathcal{T}}, C_i^{\mathcal{T}})\}_{i=1}^{|\mathcal{T}|}, \{\Delta_{\mathsf{ty}}\}_{\mathsf{ty} \in \mathsf{Ty}}\right)$

assert $|\mathcal{S}| + |\mathcal{T}| \leq \beta$

assert $\{\mathsf{nul}_i\}$ are distinct and $\forall i : \mathsf{nul}_i \notin \vec{\mathsf{st}}[1].\mathsf{NF}$

Parse σ as $(\{(\pi_i^{\mathcal{S}}, \mathsf{com}_i^{\mathcal{S}})\}_{i=1}^{|\mathcal{S}|}, \{(\pi_i^{\mathcal{T}}, \mathsf{com}_i^{\mathcal{T}})\}_{i=1}^{|\mathcal{T}|}, \mathsf{rc})$

for $i \in [|\mathcal{S}|]$ **do**

  Find $\hat{i}$ such that tx was created w.r.t. $\vec{\mathsf{st}}[\hat{i}]$.

  $\mathsf{x}_i := (\vec{\mathsf{st}}[\hat{i}], \mathsf{nul}_i, \mathsf{com}_i^{\mathcal{S}})$

  assert $\mathsf{NIZK}[\mathcal{L}^{\mathsf{spend}}].\mathsf{Verify}(\pi_i^{\mathcal{S}}, \mathsf{x}_i)$

for $j \in [|\mathcal{T}|]$ **do assert** $\mathsf{NIZK}[\mathcal{L}^{\mathsf{output}}].\mathsf{Verify}(\pi_j^{\mathcal{T}}, (\mathsf{note}_j^{\mathcal{T}}, C_j^{\mathcal{T}}, \mathsf{com}_j^{\mathcal{T}}))$

% $\mathsf{Commit}(0, 0; \mathsf{rc}) \oplus \bigoplus_{\mathsf{ty} \in \mathsf{Ty}} \mathsf{Commit}(\mathsf{ty}, \Delta_{\mathsf{ty}}, 0) =$

return $\bigoplus \mathsf{com}_i^{\mathcal{S}} \ominus \bigoplus \mathsf{com}_i^{\mathcal{T}}$

$\underline{\mathsf{MergeSig}(\{\sigma_j\}_{j=1}^n)}$:

Parse σ_j as $(\{(\pi_i^{\mathcal{S}_j}, \mathsf{com}_i^{\mathcal{S}_j})\}_{i=1}^{|\mathcal{S}_j|}, \{(\pi_i^{\mathcal{T}_j}, \mathsf{com}_i^{\mathcal{T}_j})\}_{i=1}^{|\mathcal{T}_j|}, \mathsf{rc}_j)$

assert $\sum |\mathcal{S}_i| + \sum |\mathcal{T}_i| \leq \beta$

$\mathsf{rc} \leftarrow \sum_{j=1}^n \mathsf{rc}_j$

return $\left(\mathsf{ToSet}\left(\bigcup_{j=1}^n \{(\pi_i^{\mathcal{S}_j}, \mathsf{com}_i^{\mathcal{S}_j})\}_{i=1}^{|\mathcal{S}_j|}\right), \mathsf{ToSet}\left(\bigcup_{j=1}^n \{(\pi_i^{\mathcal{T}_j}, \mathsf{com}_i^{\mathcal{T}_j})\}_{i=1}^{|\mathcal{T}_j|}\right), \mathsf{rc}\right)$

Figure 5.8: The Zswap Construction

Note that the ciphertext $C^{\mathcal{T}}$ is not referred to in the relation, but when used with a simulation extractable (SE) NIZK, realizes a Signature of Knowledge. I.e. every proof is bound to a specific ciphertext and is invalid for any other ciphertext. The transaction signature σ then contains a proof $\pi_i^{\mathcal{S}}$ for each input together with the SHC $\mathsf{com}_i^{\mathcal{S}}$ and for each output a signature $\pi_i^{\mathcal{T}}$ and the SHC $\mathsf{com}_i^{\mathcal{T}}$. The last component of the signature is the aggregated randomness of the commitments which, together with the Δ_{ty} imbalance, allows verification.

For a full transaction verification, all proofs and signatures contained in σ must be valid and the published nullifiers must be unique regarding the set of nullifiers in state st.

With the transaction signature σ having a separate proof for each input and output, it is possible to merge transactions by calculating the union of their proofs. To maintain the verifiability of the commitments, the randomness of the merged transactions is added. The irreversible addition operation then prevents future parties to unmerge a transaction if they have not seen the separate parts beforehand. To maintain the anonymity, we order inputs and outputs canonically after each merge.

As a remark, the aggregated randomness in a transaction may be replaced by a proof of knowledge. Like the binding signature of Sapling, this finalizes a transaction such that it can no longer be merged with others.

5.6 Security Proof

In this section we prove the main three security properties of Zswap construction in Fig. 5.8 we introduced in Section 5.4.

Theorem 5.6.1 (Anti-Theft)**.** *The Zswap protocol prevents theft (Definition 5.4.4), assuming OTA security, NIZK zero-knowledge, NIZK simulation-extractability, and SHC binding and* $\mathrm{HID\text{-}OR}$.

We first discuss the general idea of the proof. When $\mathcal{A}$ triggers the winning condition of the anti-theft game with an adversarial transaction tx^*, there exists a note or a nullifier taken from some honest $\mathcal{O}_{\mathsf{Spend}}$ query $\hat{E}$, producing tx, such that not all nullifiers and notes from tx were included in tx^*. This is the query that triggers the winning condition. By NIZK simulation-extractability, the proofs that were produced in $\hat{E}$ "bind" together notes and nullifiers with the corresponding input and output commitments. This means $\mathcal{A}$ uses some commitments from tx, but drops some other. Assuming commitment binding, the values we extract for commitments of tx^* (we can extract them from NIZKs) are the same as the values committed in tx.

From this point we can build a reduction $\mathcal{B}$ to $\mathrm{HID\text{-}OR}$. $\mathcal{B}$ guesses a commitment C_X that is present in both $\mathsf{tx}, \mathsf{tx}^*$, and C_Y that is only present in tx. It asks the $\mathrm{HID\text{-}OR}$ challenger $\mathcal{C}$ for either

1. two commitments corresponding to the values $(a_1, \mathsf{ty}_1), (a_2, \mathsf{ty}_2)$ as they should be honestly; or
2. the values swapped as $(a_2, \mathsf{ty}_1), (a_1, \mathsf{ty}_2)$ if both indices correspond to inputs or both to outputs, and
3. swapped with negation $(-a_2, \mathsf{ty}_2), (-a_1, \mathsf{ty}_1)$ otherwise.

This way of embedding guarantees the $\mathrm{HID\text{-}OR}$ requirement that $\Delta_{\mathsf{ty},1} = \Delta_{\mathsf{ty},2}$. Then $\mathcal{B}$ simulates the two NIZKs corresponding to C_X, C_Y. When $\mathcal{A}$ presents tx^*, $\mathcal{B}$ will extract the randomness rc_i for all commitments except for C_X, and thus because tx^* also includes the joint randomness rc^*, $\mathcal{B}$ can compute the randomness rc_X for C_X (as $\mathsf{rc}^* - \sum \mathsf{rc}_i$). Given rc_X, $\mathcal{B}$ can check whether C_X contains (a_1, ty_1) or the "swapped" value, and thus wins $\mathrm{HID\text{-}OR}$.

Theorem 5.6.1. The proof starts by assuming $\mathcal{A}$ wins the game, and finishes with breaking the $\mathrm{HID\text{-}OR}$ assumption, while using other assumptions in the process.

Assume $\mathcal{A}$ wins the anti-theft game. The challenger finds valid $\mathsf{st}, \mathsf{tx}^*, \sigma^*$ in the log of $\mathcal{O}_{\mathsf{Insert}}$ such that for $(\mathsf{Nf}^{\mathcal{A}}, M^{\mathcal{A}'})$ (obtained through $\mathsf{SplitTx}$ and filtering $M^{\mathcal{A}}$) there exists an entry $E = (\cdot, \cdot, \mathsf{Nf}, M)$ in Spent such that $M^{\mathcal{A}\prime} \cap M \neq \emptyset \vee \mathsf{Nf}^{\mathcal{A}} \cap \mathsf{Nf} \neq \emptyset$.

We will now argue that whenever $\mathcal{A}$ uses (in tx^*) an output note note or input nullifier nul from E, it also uses the corresponding $\mathsf{com}^{\mathcal{T}}$ or $\mathsf{com}^{\mathcal{S}}$. And vice versa — including a commitment in tx^* from E forces $\mathcal{A}$ to also include the same note or nul as in E.

Let us first assume that $M^{\mathcal{A}\prime} \cap M \neq \emptyset$, let $\mathsf{note} \in M^{\mathcal{A}\prime} \cap M$. note is honestly owned (by SK), by construction of $M^{\mathcal{A}\prime}$. Since honestly produced notes are unique (see Section 5.3), all *output* notes produced in $\mathcal{O}_{\mathsf{Spend}}$ are unique too, and thus we can determine in which "critical" Spent call $\hat{E}$ this note was created; thus $\hat{E}$ is uniquely defined. Locate the corresponding $\mathcal{O}_{\mathsf{Spend}}$ call, and the related "output" proof $\pi^{\mathcal{T}}$ together with $\mathsf{com}^{\mathcal{T}}$. Say that in tx^* the proof $\pi^{\mathcal{T}\prime}$ corresponding to note is for the statement $\mathsf{x}^{\mathcal{T}\prime} = (\mathsf{note}, C^{\mathcal{T}\prime}, \mathsf{com}^{\mathcal{T}\prime})$. We claim that under NIZK SE and OTA anonymity, $\mathsf{com}^{\mathcal{T}\prime} = \mathsf{com}^{\mathcal{T}}$ (and, more generally, $\mathsf{x}^{\mathcal{T}\prime} = \mathsf{x}^{\mathcal{T}}$, where $\mathsf{x}^{\mathcal{T}} = (\mathsf{note}, C^{\mathcal{T}}, \mathsf{com}^{\mathcal{T}})$ as produced in $\mathcal{O}_{\mathsf{Spend}}$). In other words, $\pi^{\mathcal{T}}$ binds together a unique note and $\mathsf{com}^{\mathcal{T}}$, so $\mathcal{A}$ cannot produce a proof for note with a different $\mathsf{com}^{\mathcal{T}\prime} \neq \mathsf{com}^{\mathcal{T}}$.

Anti-Theft Claim 1. The gist of the reduction is that it will embed OTA anonymity challenge (note^c, C^c) in $\hat{E}$ into tx instead of (note, C), and simulate $\pi^{\mathcal{T}}$; later, on seeing $\pi^{\mathcal{T}\prime}$ from tx^*, if $\mathsf{com}^{\mathcal{T}\prime} \neq \mathsf{com}^{\mathcal{T}}$, the reduction extracts note^c message from $\pi^{\mathcal{T}\prime}$ by NIZK SE.

The reduction $\mathcal{B}$ starts by guessing the $\mathcal{O}_{\mathsf{KeyGen}}$ query to use for embedding the public key pk^c provided by the anonymity game. (Formally the game will give $\mathcal{B}$ two public keys — $\mathsf{pk}_0, \mathsf{pk}_1$, but $\mathcal{B}$ will only use one of them, say pk_0, and also for the challenge later, sending $i_0 = i_1 = 0$). Since the number of queries to $\mathcal{O}_{\mathsf{KeyGen}}$ is poly-limited, the reduction can decide from the beginning which $\mathcal{O}_{\mathsf{KeyGen}}$ query it will use for this. In that query $\mathcal{B}$ will return pk^c to $\mathcal{A}$.

Immediately $\mathcal{B}$ has the following issue with simulating $\mathcal{O}_{\mathsf{Spend}}$: when asked to spend coins from pk^c it cannot produce a proper $\pi^{\mathcal{S}}$ since for that it needs to know the corresponding sk^c. Luckily, this can be overcome. When the challenger gets (note, C) as part of $\mathcal{A}$'s input in I, it will first, as before, try to see whether this note can be received using SK (in which case it is an honest spend request). If not, it might be that this is a request to spend from pk^c: this $\mathcal{B}$ will verify by sending (note, C) to $\mathcal{O}_{\mathsf{Rcv}}$. In this case it will obtain $((a, \mathsf{ty}), r)$ and derive nul from this information. Then, $\mathcal{B}$ will simulate the corresponding $\pi^{\mathcal{S}}$.

$\mathcal{B}$ also pre-guesses a "critical" $\mathcal{O}_{\mathsf{Spend}}$ query, in which it embeds the anonymity challenge — again $\mathcal{B}$ can do it assuming poly-limited number of queries $\mathcal{A}$ can do. This critical query must have an adversarial instruction to create an output note for pk^c with some (ty, a). But instead of doing that, $\mathcal{B}$ will give $\mathcal{C}$ two different sets of attributes (e.g. $(\mathsf{ty}, 0), (\mathsf{ty}, 1)$) for the same anonymity game key ($i_0 = i_1 = 0$), receive the challenge note and ciphertext note^c, C^c, and embed them into the $\mathcal{O}_{\mathsf{Spend}}$ reply. The corresponding $\pi^{\mathcal{T}}$ must be simulated[7] Note that this embedding

[7] Formally, to apply SE, we must simulate all the NIZKs in the game (and we can do that). However, by ZK $\mathcal{A}$ cannot distinguish between simulated and non-simulated proofs. Therefore, to simplify the proof, we do not simulate all the proofs produced by $\mathcal{O}_{\mathsf{Spend}}$, but only this critical one,

strategy only makes sense when note is honest.

After embedding, $\mathcal{B}$ will continue to simulate $\mathcal{O}_{\mathsf{Spend}}$ as before, with the only difference. In all the following $\mathcal{O}_{\mathsf{Spend}}$ queries (if they happen at all) where $\mathcal{A}$ attempts to spend from note^c, $\mathcal{B}$ will use the original (ty, a) instead of properly receiving the note (which it cannot do since sk^c is owned by $\mathcal{C}$ only).

In the end of the anti-theft game, on detecting tx^* triggering a winning condition, assuming $\mathsf{com}^{\mathcal{T}\prime} \neq \mathsf{com}^{\mathcal{T}}$ we know that $\mathsf{x}^{\mathcal{T}\prime} \neq \mathsf{x}^{\mathcal{T}}$. Hence, by NIZK SE $\mathcal{B}$ can extract from $\pi^{\mathcal{T}}$ included with the challenge note of tx^* (extraction is possible whenever proof verifies, and statement is different from statements of all (simulated) proofs $\mathcal{A}$ sees), and obtain the randomness r for this note. Using r, $\mathcal{B}$ can decide which note, note_0 or note_1 it was given by anonymity challenger $\mathcal{C}$, and thus break OTA anonymity. □

Second, assume that $\mathsf{Nf}^{\mathcal{A}} \cap \mathsf{Nf} \neq \emptyset$, and take any nullifier nul from this intersection. This case is a bit more tricky: since nullifiers are not unique, and nul can appear in many Spent entries (call their set $\mathsf{Spent}_{\mathsf{nul}}$), it is not immediately clear which (critical) query $\hat{E} \in \mathsf{Spent}_{\mathsf{nul}}$ the nullifier was "taken from"[8].

In tx^* locate the corresponding $\mathcal{L}^{\mathsf{spend}}$ proof $\pi^{\mathcal{S}\prime}$ and the commitment $\mathsf{com}^{\mathcal{S}\prime}$, such that the proof verifies for $\mathsf{x}^{\mathcal{S}\prime} = (\mathsf{st}^*, \mathsf{nul}, \mathsf{com}^{\mathcal{S}\prime})$. In the queries from $E \in \mathsf{Spent}_{\mathsf{nul}}$, $\mathcal{O}_{\mathsf{Spend}}$ produces proofs π_i for statements $\mathsf{x}_i^{\mathcal{S}} = (\mathsf{st}_i, \mathsf{nul}, \mathsf{com}_i^{\mathcal{S}})$. We claim (by nullifier pseudorandomness, collision-resistance, and weak SE of the NIZK) that for some $E_j \in \mathsf{Spent}_{\mathsf{nul}}$ it holds that $\mathsf{x}_j^{\mathcal{S}} = \mathsf{x}^{\mathcal{S}\prime}$, and as a result, $\mathsf{com}^{\mathcal{S}\prime} = \mathsf{com}_j^{\mathcal{S}}$. As a side result of this claim, $\mathcal{B}$ can now identify the critical query $E_j = \hat{E}$ by looking for $\mathsf{com}^{\mathcal{S}'}$ among $\mathsf{com}_j^{\mathcal{S}}$.

Anti-Theft Claim 2. The proof is similar to the first claim, but by NIZK SE extraction we obtain the preimage of the nullifier (which is supposed to be pseudorandom).

Recall that the pseudorandomness game gives us a challenge public key pk^c, a $\mathsf{Receive}(\cdot, \cdot, \mathsf{sk}^c)$ oracle $\mathcal{O}_{\mathsf{Rcv}}$, and the challenge oracle $\mathcal{O}_{\mathsf{PRF}}$ that returns evaluations of either a real $\mathsf{NulEval}(\mathsf{sk}^c, \cdot)$ or a randomly chosen $f(\cdot)$.

---

which is equivalent.

[8]The fact that each honest query has at least one honest output does not help with query identification: $\mathcal{A}$ does not *have to* include any honest output notes into tx^*, and still must be able to trigger the winning condition of the game.

The reduction $\mathcal{B}$ first selects a target $\mathcal{O}_{\mathsf{KeyGen}}$ query as in Claim 1, and in this query, simulating $\mathcal{O}_{\mathsf{KeyGen}}$ to $\mathcal{A}$, it will return pk^c (without knowing sk^c). Whenever $\mathcal{B}$ needs to receive a note sent to pk^c (e.g. sent by $\mathcal{A}$ through $\mathcal{O}_{\mathsf{Insert}}$), $\mathcal{B}$ will use the $\mathcal{O}_{\mathsf{Rcv}}$.

As the game proceeds, $\mathcal{B}$ uses $\mathcal{O}_{\mathsf{PRF}}$ to generate a value for *each* $\mathsf{NulEval}(\mathsf{sk}^c, \cdot)$ (in $\mathcal{O}_{\mathsf{Spend}}$ queries if such appear). It simulates all the corresponding $\mathcal{L}^{\mathsf{spend}}$ NIZKs. Call the logs of all these $\mathcal{O}_{\mathsf{Spend}}$ queries $\mathsf{Spent}_{\mathsf{nul}}$. In other words, $\mathcal{B}$ embeds $\mathcal{O}_{\mathsf{PRF}}$ responses into all relevant queries $\mathsf{Spent}_{\mathsf{nul}}$ simultaneously.

After simulating the anti-theft game to $\mathcal{A}$, some tx^* will trigger the winning condition. By weak SE, and since $\pi^{\mathcal{S}\prime} \in \mathsf{tx}^*$ verifies on $\mathsf{x}^{\mathcal{S}\prime}$ that includes nul (if $\mathcal{B}$ guessed correctly), unless $\mathsf{x}^{\mathcal{S}\prime} = \mathsf{x}^{\mathcal{S}}_i$ for some $\mathsf{x}^{\mathcal{S}}_i \in \mathsf{Spent}_{\mathsf{nul}}$, we can extract the witness from $\pi^{\mathcal{S}\prime}$. If extraction is possible, $\mathcal{B}$ obtains (sk, r) such that $\mathsf{NulEval}(\mathsf{sk}, r) = \mathsf{nul}$ for $\mathsf{nul} \in \mathsf{Spent}_{\mathsf{nul}}$. By nullifier collision-resistance it is not possible that the extracted (sk, r) is a different input from (sk^c, r') (where r' is from the critical query). This means that either sk is an actual secret key used in $\mathcal{O}_{\mathsf{PRF}}$ if it is instantiated with a nullifier evaluation; or the oracle is random. So $\mathcal{B}$ uses sk to compare $\mathcal{O}_{\mathsf{PRF}}$ outputs with $\mathsf{NulEval}(\mathsf{sk}, r)$ and thus break the PRF game.

Therefore, it has to be that $\exists j$ such that $\mathsf{x}^{\mathcal{S}}_j = \mathsf{x}^{\mathcal{S}\prime}$ and thus $\mathsf{com}^{\mathcal{S}\prime} = \mathsf{com}^{\mathcal{S}}_j$. □

This last reduction can be applied to all the nullifiers in $\mathsf{Nf}^{\mathcal{A}} \cap \mathsf{Nf}$: if $\mathcal{A}$ uses an honest nullifier, we are able to locate the critical query it was taken from, and the related $\mathsf{com}^{\mathcal{S}}$ in tx^* is the same as in that critical query.

Now, in tx^*, the adversary might try to combine several nullifiers or notes from different Spent queries, and for the final reduction we only need to focus on a single such critical query. In other words, winning condition "$\exists \mathsf{Spent}$ entry $\mid \ldots$ " can be triggered by several such entries. Fix any query $\hat{E}$ that has an honest nullifier or a note triggering winning condition[9].

Let tx be the transaction from $\hat{E}$, which defines M, Nf. Call I_M the indices in tx in which we observe $M^{\mathcal{A}\prime} \cap M \neq \emptyset$, similarly I_{Nf} for $\mathsf{Nf}^{\mathcal{A}} \cap \mathsf{NF}$. Similarly, call I^*_M, I^*_{Nf} the corresponding indices in tx^*. (E.g. this notation implies $\forall i : \mathsf{nul}_{I_{\mathsf{Nf},i}} = \mathsf{nul}^*_{I^*_{\mathsf{Nf},i}}$)

There exists a set $C_0 = \{\mathsf{com}^{\mathcal{S}}_i\}_{I_{\mathsf{Nf}}} \cup \{\mathsf{com}^{\mathcal{T}}_i\}_{I_M}$ as part of tx. And after the previous

[9] Not all $\mathcal{O}_{\mathsf{Spend}}$ queries *just triggering* winning condition here work: when fixing $\hat{E}$ that shares a *nullifier* with tx^*, recall that some other $\mathcal{O}_{\mathsf{Spend}}$ query can also have nul, but by the second claim we can locate the "true" critical query for that nul by comparing $\mathsf{com}^{\mathcal{S}}$. Fix only such a "truly" critical query.

two claims on NIZK binding commitments we now know that there is a subset $C_0^* = \{\mathsf{com}_i^{\mathcal{S}*}\}_{I_{\mathsf{Nf}}^*} \cup \{\mathsf{com}_i^{\mathcal{T}*}\}_{I_M^*}$ of commitments C^* in σ^* which is exactly the same as C_0. But there are also other two disjoint sets of commitments, $C_1 = C \setminus C_0$ and $C_1^* = C \setminus C_0^*$. E.g. C_1^* corresponds to the input nullifiers and output notes of tx^* that were not taken from $\hat{E}$, but from elsewhere (other honest queries, or adversarially generated).

Recall that C_0 is defined w.r.t. $M \cap M^{\mathcal{A}\prime}$ and $\mathsf{Nf} \cap \mathsf{Nf}^{\mathcal{A}}$, and so nullifiers and notes included with C_1^* in tx^* are, by definition, disjoint with M and Nf in tx. We now present the last claim: if $\mathcal{A}$ triggers the winning condition of the anti-theft game, then it can break security (binding or $\mathrm{HID\text{-}OR}$) of the commitment scheme.

Anti-Theft Claim 3. We first describe the reduction $\mathcal{B}$ to $\mathrm{HID\text{-}OR}$. First, it pre-guesses a query $\hat{E}$ in which it will embed. When $\mathcal{A}$ asks query $\hat{E}$, $\mathcal{B}$ takes any two distinct indices corresponding to inputs and outputs, which define two commitments C_X and C_Y it will embed into. The reduction will create all commitments except these two honestly, and regarding the two it will put there $\mathrm{HID\text{-}OR}$ output challenge for either: (1) original two original values $(a_1, \mathsf{ty}_1), (a_2, \mathsf{ty}_2)$; or (2) them swapped as $(a_2, \mathsf{ty}_1), (a_1, \mathsf{ty}_2)$ if both indices correspond to inputs or both to outputs, and swapped with negation $(-a_2, \mathsf{ty}_2), (-a_1, \mathsf{ty}_1)$ otherwise. This way to embed guarantees the $\mathrm{HID\text{-}OR}$ requirement that $\Delta_{\mathsf{ty},0} = \Delta_{\mathsf{ty},1}$. The joint randomness rc_0 for C_X, C_Y the reduction will sum together with the randomness for other honestly produced inputs or outputs to output the final total randomness rc for tx. The only two NIZKs that need to be simulated are the ones that correspond to C_X, C_Y. Note that we always have at least one input and output in the transaction — this is because there is always by $\mathcal{O}_{\mathsf{Spend}}$ mandates $|O| > 0$, and $|I| > 0$ since offer without inputs cannot trigger the winning condition, since it is guaranteed to be classified as honest by $\mathsf{SplitTx}$ because of note uniqueness.

Continue simulating the anti-theft game to $\mathcal{A}$. This does not require any changes — further queries to $\mathcal{O}_{\mathsf{Spend}}$ do not depend on our embedding, and can be performed as before.

In the end of the game, $\mathcal{A}$ provides a tx^* which triggers the winning condition with some notes or nullifiers corresponding to commitments C_0. The reduction will find C_x in C_0 (abort if it is not present), it will assert that $C_Y \notin C$ (abort otherwise). Then it will call NIZK extractor to obtain the randomness rc_i^* for all commitments in C^* except for C_X, which gives joint $\mathsf{rc}^{*\prime}$ for the whole C_1^* when summed up, and thus $\mathcal{B}$ computes $\mathsf{rc}_0^* = \mathsf{rc}^* - \mathsf{rc}^{*\prime}$ (where rc^* is total joint randomness of $\mathtt{tx}^*$). The reduction will return $b = 1$ iff $C_X = \mathsf{Com}((a_1, \mathsf{ty}_1), \mathsf{rc}_0^*)$.

Now we argue that the reduction breaks binding or $\mathrm{HID\text{-}OR}$of the commitment scheme if $\mathcal{A}$ wins the anti-theft game. First, the total probability of all guesses to be correct is $1/|Q|m^2$, which is polynomial, where Q is a number of queries to $\mathcal{O}_{\mathsf{Spend}}$, and $m = \mathsf{poly}(\lambda)$ is a maximum number of inputs and outputs in a $\hat{E}$ transaction). The guess inside $\hat{E}$ is correct if $C_X \in C_0$ and $C_Y \in C_1$ — therefore, the probability of the guess being correct is at least $1/m^2 = \mathsf{poly}(\lambda)$.

Now, assume all guesses are correct, and that $\mathcal{A}$ wins the anti-theft game — then $\hat{E}$ was not included completely in $\mathtt{tx}^*$, so $C_X \in C_0^*, C_Y \notin C^*$. We know by the previous two claims that C_X in $\mathtt{tx}^*$ is the same as in $\mathtt{tx}$ (since the nullifier or the note is the same, because it triggered the winning condition). We cannot extract from the corresponding (simulated) NIZK π_X — however, by NIZK SE we can still extract[10] from all the other NIZKs in $\mathtt{tx}^*$ (or they were honestly produced in $\mathtt{tx}$, which is equivalent), which is what reduction does. So the extractor in $\mathcal{B}$ will succeed, and $\mathcal{B}$ will obtain rc_0^*. This rc_0^* should give C_X when message is either (a_1, ty_2) or (a_2, ty_2); and thus $\mathcal{B}$ wins $\mathrm{HID\text{-}OR}$. And if C_X does not match $\mathsf{Com}((a_b, \mathsf{ty}_b), \mathsf{rc}_0^*)$ for both $b \in \{0, 1\}$, it means that binding of C_X is broken, that is $\mathcal{A}$ found a different value and randomness giving rise to the same C_X. Thus $\mathcal{B}$ can either determine whether C_X commits to the original value or the "swapped" one, which is enough to break $\mathrm{HID\text{-}OR}$; or $\mathcal{B}$ breaks commitment binding. □

This concludes the anti-theft proof. □

Theorem 5.6.2 (Balance)**.** *The Zswap protocol satisfies the Balance property (Definition 5.4.5) by anti-theft, NIZK SE, OTA Security, commitment binding, and Merkle tree binding.*

Intuitively, the property reduces to:

[10]Again, there is a catch in how we use SE here. In the original SE, all proofs are simulated, and one can extract from those which do not have the same instances x as simulated ones $\mathsf{x}_i^{\mathsf{sim}}$. In our case, only some (two) proofs are simulated, and other proofs are honest. But from honest proofs we can straightforwardly extract (since we produced them), so still it is true that extraction is possible when $\forall i : \mathsf{x} \neq \mathsf{x}_i^{\mathsf{sim}}$.

1. NIZK SE (implying knowledge soundness (KS)) and commitment homomorphism (every valid transaction is balanced),

2. anti-theft, since $\mathcal{A}$ can use only its own notes; and binding, since $\mathcal{A}$ can only open output commitments in a single way when spending them.

The first guarantees that transactions do not break per-type balance and do not introduce any coins out of thin air etc. The second proves that in the long-term $\mathcal{A}$ can only move funds through such honest transactions.

Theorem 5.6.2. We progress by the following sequence of games:

$\underline{\mathcal{G}_1}$**:** We start by introducing transaction extractor into the balance game. The game $\mathsf{Balance}^1$, presented below, is different from the standard $\mathsf{Balance}$ in two aspects:

1. the setup algorithm now generates the NIZK in the dishonest way, not disposing of the trapdoor td from $\mathsf{NIZK.Setup}$;

2. when the game processes transactions in the loop, the extractor $\mathsf{Ext}_{\mathcal{A}}$ is introduced, which uses td to obtain pre-transaction ptx from every tx. This extracted ptx is then asserted to be an input to tx.

Lines marked with a star $*$ indicate new or changed lines.

$\underline{\mathsf{Balance}^1_{\mathcal{A},\mathsf{Ext}_{\mathcal{A}}}(1^\lambda)}$

% Use simulated setup for the NIZK.
$(\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{Setup}'(1^\lambda)$ *
$(\mathsf{st}^*, I_0, I, \mathsf{SK}^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{KeyGen}}, \mathcal{O}_{\mathsf{Spend}}, \mathcal{O}_{\mathsf{Insert}}}(\mathsf{pp})$
$\ldots$
$(\mathsf{Ins}', \cdot) \leftarrow \mathsf{GetLog}(\mathsf{st}^*)$
for all $(\mathsf{st}, \cdot, \mathsf{tx}, \sigma) \in \mathsf{Ins}'$ **do**
    $\ldots$
    Parse tx as $\left(\{\mathsf{nul}_i^*\}_{i=1}^{|\mathcal{S}|}\{(\mathsf{note}_i^{\mathcal{T}*}, C_i)\}_{i=1}^{|\mathcal{T}|}, \{\Delta_{a:\mathsf{ty}}\}_{\mathsf{ty} \in \mathsf{Ty}}\right)$ *
    % Extract pre-transaction used to create tx *
    $\mathsf{ptx}_{\mathsf{Ext}}@(\mathcal{S}_{\mathsf{Ext}}, \mathcal{T}_{\mathsf{Ext}}) \leftarrow \mathsf{Ext}_{\mathcal{A}}(\mathsf{pp}, \mathsf{td}, \mathsf{tx}, \sigma)$ *
    Parse $\mathcal{S}_{\mathsf{Ext}}$ as $\{(\mathsf{sk}_i^{\mathcal{S}}, \mathsf{note}_i^{\mathcal{S}}, \mathsf{nul}_i, \mathsf{path}_i, (a_i^{\mathcal{S}}, \mathsf{ty}_i^{\mathcal{S}}), r_i^{\mathcal{S}})\}_{i=1}^{|\mathcal{S}|}$ *
    Parse $\mathcal{T}_{\mathsf{Ext}}$ as $\{(\mathsf{pk}_i^{\mathcal{T}}, \mathsf{note}_i^{\mathcal{T}}, C_i^{\mathcal{T}}, (a_i^{\mathcal{T}}, \mathsf{ty}_i^{\mathcal{T}}), r_i^{\mathcal{T}})\}_{i=1}^{|\mathcal{T}|}$ *
    $(\cdot, \vec{\mathsf{st}}) \leftarrow \mathsf{GetLog}(\mathsf{st})$ *

$\quad$ **assert** $\mathsf{CheckPTx}(\mathsf{st}, \mathsf{ptx}_{\mathsf{Ext}}) = 1 \wedge \mathsf{CompleteTx}(\mathsf{ptx}_{\mathsf{Ext}}) = \mathsf{tx}$ *
$\quad$ ...
if $\exists \mathsf{ty} : v_{\mathcal{A}}[\mathsf{ty}] > v_0[\mathsf{ty}] + v_{\mathcal{H}-}[\mathsf{ty}] - v_{\mathcal{H}+}[\mathsf{ty}]$ **then return** 1
else return 0

Formally, we claim that for every adversary $\mathcal{A}$ there exists $\mathsf{Ext}_{\mathcal{A}}$ s.t.

$$\Pr[\mathsf{Balance}^1_{\mathcal{A},\mathsf{Ext}_{\mathcal{A}}}(1^\lambda) = 1] \geq \Pr[\mathsf{Balance}_{\mathcal{A}}(1^\lambda) = 1] - \mathsf{negl}(\lambda)$$

assuming KS of the NIZKs holds, and commitments are binding:

Theorem 5.6.2, Transition 1. The extractor $\mathsf{Ext}_{\mathcal{A}}$ is essentially a wrapper around two NIZK extractors (for spend and output proofs): $\mathsf{Ext}_{\mathcal{A}}$ internally calls $\mathsf{NIZK}[\mathcal{L}^{\mathsf{spend}}].\mathsf{Ext}_{\mathcal{A}}$ and $\mathsf{NIZK}[\mathcal{L}^{\mathsf{output}}].\mathsf{Ext}_{\mathcal{A}}$, both of which are guaranteed to exist by KS of the NIZKs, and concatenates their results into $\mathcal{S}$ and $\mathcal{T}$.

Note that the ptx assertion is the only way the games are different in terms of possibility of different outcome. The only other line that can fail is the internal $\mathsf{GetLog}(\mathsf{st})$; but since it queries a state which was returned by the previous $\mathsf{GetLog}(\mathsf{st}^*)$, it will not abort. Therefore we must only argue why the assertion containing $\mathsf{CheckPTx}$ and $\mathsf{CompleteTx}$ will hold every time – in other words the probability for adversary to win will not be hindered by the assertion failing.

We first argue why $\mathsf{CheckPTx}(\mathsf{st}, \mathsf{ptx}_{\mathsf{Ext}}) = 1$. The check that $\mathsf{nul}_i \notin \mathsf{st.MT}$ is directly checked in Verify. All checks except for the $\mathsf{CheckBalance}$ reduce to the NIZK knowledge-soundness.

Looking at every $\mathcal{L}^{\mathsf{spend}}$ NIZK first, the fact it verifies implies that the extractor $\mathsf{NIZK}[\mathcal{L}^{\mathsf{spend}}].\mathsf{Ext}_{\mathcal{A}}$ returns $\mathsf{w} = (\mathsf{path}_i, \mathsf{sk}^{\mathcal{S}}_i, a^{\mathcal{S}}_i, \mathsf{ty}^{\mathcal{S}}_i, r^{\mathcal{S}}_i, \mathsf{rc}^{\mathcal{S}}_i)$, which by NIZK KS guarantees for all $i \in [|\mathcal{S}|]$:

$$\begin{aligned}
&\mathsf{st.MT}[\mathsf{path}_i] = \mathsf{note}^{\mathcal{S}}_i \wedge \\
&\quad \mathsf{nul}_i = \mathsf{OTA.NulEval}(\mathsf{sk}^{\mathcal{S}}_i, r^{\mathcal{S}}_i) \wedge \\
&\quad \mathsf{note}^{\mathcal{S}}_i = \mathsf{OTA.Gen}(\mathsf{OTA}.P(\mathsf{sk}_i), (a^{\mathcal{S}}_i, \mathsf{ty}^{\mathcal{S}}_i), r^{\mathcal{S}}_i)
\end{aligned}$$

Conditions (1) and (3) are satisfied by $\mathcal{L}^{\mathsf{open}}$, and (2) is by $\mathcal{L}^{\mathsf{nul}}$ (see $\mathcal{L}^{\mathsf{spend}}$ structure).

Similarly, the output NIZK for the $\mathcal{L}^{\text{output}}$ language guarantees that the extractor NIZK[$\mathcal{L}^{\text{output}}$].Ext$_{\mathcal{A}}$ will return $(\mathsf{pk}_i^{\mathcal{T}}, a_i^{\mathcal{T}}, \mathsf{ty}_i^{\mathcal{T}}, r_i^{\mathcal{T}}, \mathsf{rc}_i^{\mathcal{T}})$ such that:

$$\forall i \in [|\mathcal{T}|] : \mathsf{note}_i^{\mathcal{T}} = \mathsf{OTA.Gen}(\mathsf{pk}_i^{\mathcal{T}}, (a_i^{\mathcal{T}}, \mathsf{ty}_i^{\mathcal{T}}), r_i^{\mathcal{T}})$$

Which again holds by the NIZK KS and the structure of $\mathcal{L}^{\text{open}}$, being part of $\mathcal{L}^{\text{output}}$.

The last balance check that $\mathsf{CheckBalance}(\mathcal{S}, \mathcal{T}) = 1$, which is implemented as $\forall \mathsf{ty} \in \{\mathsf{ty}_i^{\mathcal{S}}\}_{i=1}^{|\mathcal{S}|} \cup \{\mathsf{ty}_i^{\mathcal{T}}\}_{i=1}^{|\mathcal{T}|}$: $\sum_{a \in \{a_i^{\mathcal{S}} | \mathsf{ty}_i^{\mathcal{S}} = \mathsf{ty}\}_{i=1}^{|\mathcal{S}|}} a - \sum_{a \in \{a_i^{\mathcal{T}} | \mathsf{ty}_i^{\mathcal{T}} = \mathsf{ty}\}_{i=1}^{|\mathcal{T}|}} a = \Delta_{\mathsf{ty}}$ succeeds by the commitment homomorphic property in $\mathsf{Verify}(\cdot, \cdot, \cdot)$. We know that tx verifies, therefore the homomorphic sum in Verify holds. By the previous steps we also know that the extracted type-value pairs $(a_i^{\mathcal{S}}, \mathsf{ty}^{\mathcal{S}})$ and $(a_i^{\mathcal{T}}, \mathsf{ty}^{\mathcal{T}})$ are inputs to the commitments $\mathsf{com}^{\mathcal{S}}$ and $\mathsf{com}^{\mathcal{T}}$ correspondingly. By the $a \in [2^{\alpha}]$ check in NIZK, and by the β check in Verify (and by the choice of α and β), we know that homomorphic sums of $\mathsf{com}^{\mathcal{S}}$,$\mathsf{com}^{\mathcal{T}}$ will not overflow. Since the commitments in Verify (including commitment to Δ_{ty}) sum to the identity, the committed values a_i sum to exactly Δ_{ty} per type.

Therefore $\mathsf{CheckPTx} = 1$ with overwhelming probability by existence of NIZK extractors.

Finally, we claim that $\mathsf{CompleteTx} = 1$. CompleteTx does two things: first, it removes secret information from ptx, and, second, it computes the imbalances Δ_{ty}. First, the public information in $\mathsf{tx} \cap \mathsf{ptx}$ is input nullifiers, and note-cipertext output pairs – because these are in the statements of corresponding NIZKs, they are bound to be exactly the same (formally, these values are the input of Ext as x, and not its output). Therefore all parts of tx except for the deltas are exactly like in the ptx extracted. Second, that the deltas in tx are also the same as computed from a_i follows from the commitment binding. And the fact that they sum up to Δ we have just shown when arguing $\mathsf{CheckPTx} = 1$.

Therefore the gap $\mathsf{negl}(\lambda)$ in this game consists of probability of failing, or binding being broken. $\square$

Now that we have ptx values extracted in clear, the main balance property will follow inductively by following the $(\mathsf{st}, \mathsf{st}', \mathsf{tx}, \sigma) \in \mathsf{Ins}'$ loop step by step.

But before we present the main proof reasoning, we must introduce several auxiliary constructions into our game. Our intention within next several games is to trace the notes that $\mathcal{A}$ can spend, and will argue that both during the game, and in the end of it, the sum of these is not enough to break the balance predicate.

$\underline{\mathcal{G}_2}$**:** We add an additional condition on the extracted data by asserting that it is equal to the type-value pairs we use when modifying $v_{\mathcal{H}+}, v_{\mathcal{H}-}$.

$\underline{\mathsf{Balance}^2_{\mathcal{A},\mathsf{Ext}_\mathcal{A}}(1^\lambda)}$

$\ldots$
for all $(\mathsf{st}, \cdot, \mathsf{tx}, \sigma) \in \mathsf{Ins}'$ **do**
  $\ldots$
  $\mathsf{ptx}_{\mathsf{Ext}}@(\mathcal{S}_{\mathsf{Ext}}, \mathcal{T}_{\mathsf{Ext}}) \leftarrow \mathsf{Ext}_\mathcal{A}(\mathsf{pp}, \mathsf{td}, \mathsf{tx}, \sigma)$
  $\ldots$
  for $(\mathsf{Nf}, M) \in \mathsf{tx}_\mathcal{H}$ **do**
    Find $(\cdot, I, \mathsf{Nf}, M) \in \mathsf{Spent}$
    for $(\mathsf{note}', C') \in I$ **do**
      $(\cdot, (a, \mathsf{ty}), \cdot) \leftarrow \mathsf{TryReceive}(\mathsf{note}', C', \mathsf{SK})$
      $v_{\mathcal{H}-}[\mathsf{ty}] := v_{\mathcal{H}-}[\mathsf{ty}] + a$
      assert $(a, \mathsf{ty}) = (a_i^\mathcal{S}, \mathsf{ty}_i^\mathcal{S})$ *
  for $(\mathsf{note}', C') \in M$ **where** $(\cdot, M) \in \mathsf{tx}$ **do**
    $\mathsf{res} \leftarrow \mathsf{TryReceive}(\mathsf{note}', C', \mathsf{SK})$
    if $\mathsf{res} = (\cdot, (a, \mathsf{ty}), \cdot) \neq \perp$ **then**
      $v_{\mathcal{H}+}[\mathsf{ty}] := v_{\mathcal{H}+}[\mathsf{ty}] + a$
      assert $(a, \mathsf{ty}) = (a_i^\mathcal{T}, \mathsf{ty}_i^\mathcal{T})$ *
$\ldots$

This game transition is by OTA binding and nullifier collision-resistance.

Theorem 5.6.2, Transition 2. The first assertion is reached because $(\cdot, I, \mathsf{Nf}, M) \in \mathsf{Spent}$ for (Nf, M) located by SplitTx to be honest on the basis of $(\mathsf{Nf}, M) \in \mathsf{tx}$, where tx is a currently processed transaction. Since notes that are added to I pass checks in BuildPTx (inside $\mathcal{O}_{\mathsf{Spend}}$), we know that (a, ty) are valid inputs producing $\mathsf{nul} \in \mathsf{Nf}$ and note. The argument is exactly the same as second claim of proof of Theorem 5.6.1: this requires nullifier pseudorandomness, collision-resistance, and weak SE of the NIZK. By this argument we know that the NIZK statements are the same, thus value commitments are the same, and thus the values inside the commitments (by value commitment binding) are the same.

The second assertion will hold by OTA binding: $\mathsf{TryReceive}$ guarantees that (a, ty) are valid input to note', and so are $(a_i^{\mathcal{S}}, \mathsf{ty}_i^{\mathcal{S}})$ by $\mathsf{CheckPTx}$ and the previous game. Therefore these must be equal. □

$\underline{\mathcal{G}_3}$**:** In the next game $\mathsf{Balance}^3$ we add more meaning to the extracted data by linking input notes to the notes mentioned in previous transactions. We assert that all the extracted input notes are the same as some output notes generated before, as the notes originally present in I_0; and that they are not an input to any previous transaction (thus not double spent).

$\underline{\mathsf{Balance}^3_{\mathcal{A},\mathsf{Ext}_\mathcal{A}}(1^\lambda)}$

...
for all $(\mathsf{st}, \cdot, \mathsf{tx}, \sigma) \in \mathsf{Ins}'$ **do**
    ...
    $\mathsf{ptx}_{\mathsf{Ext}}@(\mathcal{S}_{\mathsf{Ext}}, \mathcal{T}_{\mathsf{Ext}}) \leftarrow \mathsf{Ext}_\mathcal{A}(\mathsf{pp}, \mathsf{td}, \mathsf{tx}, \sigma)$
    ...
    for $\mathsf{nul} \in \mathsf{tx}$ **do**
        assert The corresponding extracted $\mathsf{note}^{\mathcal{S}}$: *
            (1) Is present at least once *
                (1.1) in some $\mathcal{T}_{\mathsf{Ext}}$ of some previous transaction; or *
                (1.2) in I_0 directly. *
            (2) Is not part of $\mathcal{S}_{\mathsf{Ext}}$ in any previous txs. *
    ...
if $\exists \mathsf{ty} : v_\mathcal{A}[\mathsf{ty}] > v_0[\mathsf{ty}] + v_{\mathcal{H}-}[\mathsf{ty}] - v_{\mathcal{H}+}[\mathsf{ty}]$ **then return** 1
else return 0

The transition is by MT binding and nullifier uniqueness.

Theorem 5.6.2, Transition 3. By NIZK KS $\mathsf{note}^{\mathcal{S}}$ must be in the Merkle tree. Notes are put into the MT at previous steps, or they are present in $\mathsf{st}_0.\mathsf{MT}$ — this is how MT is populated — so we can always find the previous step where $\mathsf{note}^{\mathcal{S}}$ was introduced. In other words, we can always find the previous state with $\mathsf{Com}(\mathsf{note}^{\mathcal{S}})$ present. The note inserted into MT at that step is equal to $\mathsf{note}^{\mathcal{S}}$ because MT is a binding commitment scheme. This proves the first part of the condition: there always exists the previous step with $\mathsf{note}^{\mathcal{S}}$ extracted, or this $\mathsf{note}^{\mathcal{S}}$ was present in $\mathsf{st}_0.\mathsf{MT}$.

Regarding the second condition of the assertion, namely that $\mathsf{note}^{\mathcal{S}}$ is not an input to any previous tx. Let nul be the nullifier of $\mathsf{note}^{\mathcal{S}}$; by $\mathcal{L}^{\mathsf{nul}}$ and previous game we know that $\mathsf{note}^{\mathcal{S}} = \mathsf{Gen}(\mathsf{pk}, (a, \mathsf{ty}), r)$ and $\mathsf{nul} = \mathsf{NulEval}(\mathsf{sk}, r)$. Now assume the contrary, that there is a previous spend which extracts $\mathsf{note}^{\mathcal{S}}$ too. Backtrace to this spend, and let nul_0 be its nullifier, together with r, sk_0 all jointly satisfying the same $\mathcal{L}^{\mathsf{nul}}$ equation. It must be that $\mathsf{nul}_0 \neq \mathsf{nul}$ where nul belongs to $\mathsf{note}^{\mathcal{S}}$ — if $\mathsf{nul}_0 = \mathsf{nul}$, tx cannot be verified at the current state (nullifier's set is append-only by construction). This is enough to break nullifier uniqueness, since we just observed two nullifiers for the same note:

$$\mathsf{note}^{\mathcal{S}} = \mathsf{Gen}(\mathsf{OTA}.P(\mathsf{sk}), (a, \mathsf{ty}), r) = \mathsf{Gen}(\mathsf{OTA}.P(\mathsf{sk}_0), (a_0, \mathsf{ty}_0), r_0)$$
$$\mathsf{NulEval}(\mathsf{sk}, r) \neq \mathsf{NulEval}(\mathsf{sk}_0, r_0)$$

Thus double-spending is not allowed. □

Note that the first condition of the game is more nuanced. In fact, because (malicious) transaction creator has control over the output note randomness, it is possible to create several output notes with the same value and randomness. However, since all these notes have the same nullifier, only one of them can be spent[11]. Hence we only require finding "at least one output".

$\underline{\mathcal{G}_4}$**:** In the next game $\mathsf{Balance}^4$ we will add a (multi-)set $N_{\overline{\mathcal{H}}}$ initially populated with notes in I_0. This set will track all non-honest notes, where "not honest" here means not just adversarial notes (that $\mathcal{A}$ can claim similarly to how $v_{\mathcal{A}}$ is computed) but also notes that are "burned" — that cannot be accessed by anyone.

Each loop iteration will

1. remove adversarial input notes from $N_{\overline{\mathcal{H}}}$, and
2. add to $N_{\overline{\mathcal{H}}}$ all the output notes that cannot be received by SK.

[11] Similar to the "Faerie Gold" attack in zcash

For both actions we need the data extracted in $\mathsf{Balance}^1$, that "reveals" the notes behind the nullifiers, including adversarial ones. The new variable $v_{\overline{\mathcal{H}}}$ will track balances inside $N_{\overline{\mathcal{H}}}$, and is updated whenever $N_{\overline{\mathcal{H}}}$ is changed. $N_{\overline{\mathcal{H}}}$ and $v_{\overline{\mathcal{H}}}$ track *all* the transient adversarial transactions, not counted in the final $v_{\mathcal{A}}$ (which only sums up the "resulting"[12] assets of $\mathcal{A}$).

$\underline{\mathsf{Balance}^4_{\mathcal{A},\mathsf{Ext}_\mathcal{A}}(1^\lambda)}$

$\ldots$
$N_{\overline{\mathcal{H}}} \leftarrow \emptyset$ *
for $(\mathsf{note}_i, C_i) \in I_0$ **do**
  $N_{\overline{\mathcal{H}}} = N_{\overline{\mathcal{H}}} \cup \{\mathsf{note}_i\}$ *
  $(\cdot, \mathsf{nul}, (a, \mathsf{ty}), \cdot) \leftarrow \mathsf{TryReceive}(\mathsf{note}_i, C_i, \mathsf{SK}^*)$
  assert $\mathsf{nul} \notin \mathsf{st}_0.\mathsf{NF}$
  $v_0[\mathsf{ty}] := v_0[\mathsf{ty}] + a$
$v_{\mathcal{H}-}, v_{\mathcal{H}+} \leftarrow (\mathsf{ty} \mapsto 0)$
$(\mathsf{Ins}', \cdot) \leftarrow \mathsf{GetLog}(\mathsf{st}^*)$
for all $(\mathsf{st}, \cdot, \mathsf{tx}, \sigma) \in \mathsf{Ins}'$ **do**
  $\ldots$
  $\mathsf{ptx}_{\mathsf{Ext}}@(\mathcal{S}_{\mathsf{Ext}}, \mathcal{T}_{\mathsf{Ext}}) \leftarrow \mathsf{Ext}_\mathcal{A}(\mathsf{pp}, \mathsf{td}, \mathsf{tx}, \sigma)$
  $\ldots$
  for $\mathsf{note}' \in \mathcal{S}_{\mathsf{Ext}} \mid$ corresponding $\mathsf{sk}^{\mathcal{S}} \notin \mathsf{SK}$ **do** *
    $N_{\overline{\mathcal{H}}} := N_{\overline{\mathcal{H}}} \setminus \{\mathsf{note}'\}$ *
    % $\mathsf{ty}^{\mathcal{S}}, a^{\mathcal{S}}$ as extracted by $\mathsf{Ext}_\mathcal{A}$ *
    $v_{\overline{\mathcal{H}}}[\mathsf{ty}^{\mathcal{S}}] := v_{\overline{\mathcal{H}}}[\mathsf{ty}^{\mathcal{S}}] - a^{\mathcal{S}}$ *
  for $(\mathsf{note}', C') \in M$ **where** $(\cdot, M) \in \mathsf{tx}$ **do**
    $\ldots$
    $\mathsf{res} \leftarrow \mathsf{TryReceive}(\mathsf{note}', C', \mathsf{SK})$
    if $\mathsf{res} = (\cdot, (a, \mathsf{ty}), \cdot) \neq \bot$ **then**
      $v_{\mathcal{H}+}[\mathsf{ty}] := v_{\mathcal{H}+}[\mathsf{ty}] + a$
    else *
      $N_{\overline{\mathcal{H}}} := N_{\overline{\mathcal{H}}} \cup \{\mathsf{note}'\}$ *
      % $\mathsf{ty}^{\mathcal{T}}, a^{\mathcal{T}}$ as extracted by $\mathsf{Ext}_\mathcal{A}$ *

[12] E.g. $\mathcal{A}$ can "refresh" the note $\mathsf{note} \in I_0$ by moving its funds fully to another adversarial $\mathsf{note}' \notin I$, and without $N_{\overline{\mathcal{H}}}$ this will not be reflected anywhere.

$v_{\overline{\mathcal{H}}}[\mathsf{ty}^{\mathcal{T}}] := v_{\overline{\mathcal{H}}}[\mathsf{ty}^{\mathcal{T}}] + a^{\mathcal{T}}$ *

if $\exists \mathsf{ty} : v_{\mathcal{A}}[\mathsf{ty}] > v_0[\mathsf{ty}] + v_{\mathcal{H}-}[\mathsf{ty}] - v_{\mathcal{H}+}[\mathsf{ty}]$ **then return** 1

else return 0

As can be seen in the part where $N_{\overline{\mathcal{H}}}$ is populated, it contains all the notes that cannot be received by honest parties, which includes: (1) adversarial notes, with or without correct ciphertext (this does not matter), (2) notes to keys that are not controlled by both adversary and honest parties (burned), (3) notes to honest parties with malformed ciphertext (also effectively burned).

We now argue that $\Pr[\mathsf{Balance}^4_{\mathcal{A},\mathsf{Ext}_{\mathcal{A}}}(1^\lambda) = 1] \geq \Pr[\mathsf{Balance}^2_{\mathcal{A},\mathsf{Ext}_{\mathcal{A}}}(1^\lambda) = 1] - \mathsf{negl}(\lambda)$ by OTA binding.

Theorem 5.6.2, Transition 4. This change only *adds* parallel logic into our computation that almost does not interact with any previous logic. It does not abort in all cases — when we append to the set and add or subtract from the corresponding value variable — but one. The only exception is the set subtraction, which we argue does not fail because of $\mathsf{Balance}^3$. There we prove that each input note is present at least once in some outputs, and is not present in the inputs. The latter guarantees that the note will not be removed twice. The former guarantees that the note will be present once to be removed: when attempting to remove a note, given that we know that it was in some outputs, the only exception would be that this note was not put into $N_{\overline{\mathcal{H}}}$ at that point, which means that it is receivable using SK, but to trigger removal $\mathsf{sk}^{\mathcal{S}}$ at the current step must be $\notin$ SK. Finding two different secret keys, one in SK, another not in SK, for the same note, is not possible by OTA binding. □

$\underline{\mathcal{G}_5}$**:** Next we add an "intermediate balance assertion", similarly to the final one, but with $v_{\overline{\mathcal{H}}}[\mathsf{ty}]$ in place of $v_{\mathcal{A}}[\mathsf{ty}]$, to the end of each iteration (and before the loop).

$\underline{\mathsf{Balance}^5_{\mathcal{A},\mathsf{Ext}_{\mathcal{A}}}(1^\lambda)}$

...

$(\mathsf{Ins}', \cdot) \leftarrow \mathsf{GetLog}(\mathsf{st}^*)$

assert $\forall \mathsf{ty} : v_{\overline{\mathcal{H}}}[\mathsf{ty}] \leq v_0[\mathsf{ty}] + v_{\mathcal{H}-}[\mathsf{ty}] - v_{\mathcal{H}+}[\mathsf{ty}]$ *

for all $(\mathsf{st}, \cdot, \mathsf{tx}, \sigma) \in \mathsf{Ins}'$ **do**
  $\ldots$
  assert $\forall \mathsf{ty} : v_{\overline{\mathcal{H}}}[\mathsf{ty}] \leq v_0[\mathsf{ty}] + v_{\mathcal{H}-}[\mathsf{ty}] - v_{\mathcal{H}+}[\mathsf{ty}]$ *
if $\exists \mathsf{ty} : v_{\mathcal{A}}[\mathsf{ty}] > v_0[\mathsf{ty}] + v_{\mathcal{H}-}[\mathsf{ty}] - v_{\mathcal{H}+}[\mathsf{ty}]$ **then return** 1
else return 0

We argue that it is not harder to win $\mathsf{Balance}^5$ than to win $\mathsf{Balance}^4$ by OTA soundness and binding.

Theorem 5.6.2, Transition 5. It could be that an adversarial strategy that worked with $\mathsf{Balance}^4$ will fail because the assertion fails. So we must argue that the assertion never fails. We proceed inductively.

In the base case, before the first loop iteration (before any transaction is processed), the assertion holds since $v_{\overline{\mathcal{H}}}[\mathsf{ty}] = v_0[\mathsf{ty}]$ and $v_{\mathcal{H}+} = v_{\mathcal{H}-} = 0$ – so trivially $\forall \mathsf{ty}.\, v_{\overline{\mathcal{H}}}[\mathsf{ty}] \leq v_0[\mathsf{ty}]$.

Assume now that the assertion holds in the beginning of the loop, we will show it persists through the loop execution. This essentially reduces to the observation that the end-of-the-loop equation is updated by differences that satisfy this equation; which reduces to the balancing property of a single transaction.

Fix a type ty, and apply the following reasoning to each type in the transaction (all of which are available in the extracted data). Compute the local *difference* values $v^{\Delta}_{\mathcal{H}-}, v^{\Delta}_{\mathcal{H}+}$ as prescribed by the game, but without updating the old $v_{\mathcal{H}-}, v_{\mathcal{H}+}$ immediately. Similarly compute $v^{\Delta}_{\overline{\mathcal{H}}}$ from the extracted (input and output) NIZKs by adding all the non-honest output values and subtracting all the non-honest adversarial input values. Note that v_0 is at no point updated after it is initialised before the main loop.

Now we need to prove that $v^{\Delta}_{\overline{\mathcal{H}}} \leq v^{\Delta}_{\mathcal{H}-} - v^{\Delta}_{\mathcal{H}+}$, then the updated end-of-the-loop equation will still hold. This reduces to the commitment balancing condition. First, observe that $v^{\Delta}_{\overline{\mathcal{H}}} = \sum a_i^{\mathcal{T}\overline{\mathcal{H}}} - \sum a_i^{\mathcal{S}\overline{\mathcal{H}}}$: we compute $v^{\Delta}_{\overline{\mathcal{H}}}$ using these extracted values a_i directly, summing all outputs and subtracting all inputs. Similarly, $v^{\Delta}_{\mathcal{H}-} = \sum a_i^{\mathcal{S}\mathcal{H}}$ and $v^{\Delta}_{\mathcal{H}+} = \sum a_i^{\mathcal{T}\mathcal{H}}$, the only difference being that values a_i here are obtained not from the extracted data, but by directly receiving the corresponding notes via $\mathsf{TryReceive}(\mathsf{note}, C, \mathsf{SK})$. But these "received" values $((a, \mathsf{ty}), r)$ are equal to the extracted ones as we asserted in the second game.

By the last (balancing) check in $\mathsf{CheckPTx}$ introduced in $\mathsf{Balance}^1$, substituting the extracted a_i values we just discussed, we obtain

$$\sum a_i^{\mathcal{S}\overline{\mathcal{H}}} + \sum a_i^{\mathcal{S}\mathcal{H}} - \sum a_i^{\mathcal{T}\overline{\mathcal{H}}} - \sum a_i^{\mathcal{T}\mathcal{H}} = 0$$

Which translates into $-v_{\overline{\mathcal{H}}}^{\Delta} + v_{\mathcal{H}-}^{\Delta} - v_{\mathcal{H}+}^{\Delta} = 0$, equivalent to $v_{\overline{\mathcal{H}}}^{\Delta} = v_{\mathcal{H}-}^{\Delta} - v_{\mathcal{H}+}^{\Delta}$ as we need.

So the predicate persists through the loop iteration. □

$\underline{\mathcal{G}_6}$**:** Our next and final step is $\mathsf{Balance}^6$ where we show that after the loop $v_{\mathcal{A}} \le v_{\overline{\mathcal{H}}}$. This is because the former counts transactions that can be received with SK^*, and the latter all that cannot be received by SK. Formally this can be shown by comparing all the notes in $N_{\overline{\mathcal{H}}}$ and I: we claim that all notes in I are present in $N_{\overline{\mathcal{H}}}$.

We show it by doing two things. First, instead of computing $v_{\mathcal{A}}$ in a separate loop before the main loop, we will move it *into* the main loop. Now, $v_{\mathcal{A}}$ is updated whenever the balance game locates a $\mathsf{note} \in I$ in the outputs of tx (this note is still attempted to be received by SK^*). To track what we have already counted into $v_{\mathcal{A}}$, we will create a variable $N_{\mathcal{A}}$: by design after the game $N_{\mathcal{A}}$ is exactly all the notes in I. This allows us to track the successive stake accumulation of $\mathcal{A}$, but only on those coins that it claims in I. And second, every time we locate a coin that goes into $v_{\mathcal{A}}$, we will assert that all notes in I are also in $N_{\overline{\mathcal{H}}}$.

$\underline{\mathsf{Balance}^6_{\mathcal{A},\mathsf{Ext}_{\mathcal{A}}}(1^\lambda)}$

...
for $(\cdot, \mathsf{nul}, (a, \mathsf{ty}), \cdot, \cdot) \in I$ **do**
  assert $\mathsf{nul} \notin \mathsf{st}^*.\mathsf{NF}$
  % Removed the $v_{\mathcal{A}}$ population line *
...
$N_{\mathcal{A}} \leftarrow \emptyset$ % This tracks notes in $v_{\mathcal{A}}$ on the fly *
for all $(\mathsf{st}, \cdot, \mathsf{tx}, \sigma) \in \mathsf{Ins}'$ **do**
  ...

    for $(\mathsf{note}', C') \in M$ **where** $(\cdot, M) \in \mathsf{tx}$ **do**
        % Compute $v_{\mathcal{A}}$ on the fly *
        if $(\mathsf{note}', C') \in I$ **then** *
            $\ldots$ % After $N_{\overline{\mathcal{H}}}$ is updated
            if $(\mathsf{note}', (a, \mathsf{ty}), \mathsf{nul}, \ldots) \in I$ **then**
                assert $\mathsf{nul} \notin \mathsf{st}^*.\mathsf{NF}$ *
                $v_{\mathcal{A}}[\mathsf{ty}] := v_{\mathcal{A}}[\mathsf{ty}] + a$ *
                $N_{\mathcal{A}} = N_{\mathcal{A}} \cup \mathsf{note}'$ *
                assert $\forall \mathsf{note} \in N_{\mathcal{A}}.\, \mathsf{note} \in N_{\overline{\mathcal{H}}}$ *
    assert $\forall \mathsf{ty} : v_{\overline{\mathcal{H}}}[\mathsf{ty}] \leq v_0[\mathsf{ty}] + v_{\mathcal{H}-}[\mathsf{ty}] - v_{\mathcal{H}+}[\mathsf{ty}]$ *
assert $v_{\mathcal{A}} \leq v_{\overline{\mathcal{H}}}$ *
if $\exists \mathsf{ty} : v_{\mathcal{A}}[\mathsf{ty}] > v_0[\mathsf{ty}] + v_{\mathcal{H}-}[\mathsf{ty}] - v_{\mathcal{H}+}[\mathsf{ty}]$ **then return** 1
else return 0

The transition is by nullifier uniqueness, pseudorandomness, and OTA binding.

Theorem 5.6.2, Transition 6. Moving $v_{\mathcal{A}}$ computation into the loop on its own does not affect the control flow: this is because all the notes in I are present as outputs of transactions in Ins', since we assert (line 4 of the original Balance game) that all the notes from I must be present in the Merkle tree $\mathsf{st}.\mathsf{MT}$ of the final state, and state maintenance oracles guarantee that they were introduced in one of the previous states $\mathsf{st}' \in \vec{\mathsf{st}}$.

The only thing that is important are the two assertions.

Let us focus on the first assertion $\forall \mathsf{note} \in N_{\mathcal{A}}.\, \mathsf{note} \in N_{\overline{\mathcal{H}}}$. It holds by induction: assuming on the previous iteration (of the inner loop) this condition holds, it can only fail if (1) some old notes in $N_{\mathcal{A}}$ have been removed from $N_{\overline{\mathcal{H}}}$, or (2) the currently added to $N_{\mathcal{A}}$ note has not been added to $N_{\overline{\mathcal{H}}}$ just a few steps before.

The first condition violates the assumption that notes in I are unspent: if note was removed from $N_{\mathcal{A}}$ it means its nullifier has been revealed. But in the final state st^* this nullifier is not present (this is checked in the beginning of the game). So nullifier uniqueness must be broken.

Second condition can fail if the note was not added to $N_{\overline{\mathcal{H}}}$, which happens only if it can be received honestly. But this means that a note in I, in the beginning of the game, can be received using SK, and also $\mathcal{A}$ showed a correct nullifier for it. This is impossible by nullifier pseudorandomness: $\mathcal{A}$ cannot generate nullifiers for notes generated for SK.

The last assertion $v_{\mathcal{A}} \leq v_{\overline{\mathcal{H}}}$ holds because we just showed notes inclusion at each step of the iteration; and because of OTA binding the values used to compute $v_{\overline{\mathcal{H}}}$ (extracted in $\mathsf{Balance}^1$) are the same as values provided by $\mathcal{A}$ in I. □

Because the predicate $v_{\overline{\mathcal{H}}} \leq \ldots$ holds in the end of each iteration for all types, and thus in the end of the last iteration, and because $v_{\mathcal{A}} \leq v_{\overline{\mathcal{H}}}$ as we showed in the last transition, the predicate in the end of the game with $v_{\mathcal{A}} \leq (v_{\overline{\mathcal{H}}} \leq) \ldots$ will also hold, and thus $\mathcal{A}$ cannot win the last game unless with negligible probability. Therefore, it cannot win the original balance game. □

Theorem 5.6.3 (Privacy)**.** *The Zswap protocol is private (Definition 5.4.6), if NIZK is zero-knowledge, Pedersen commitments are hiding, and OTA is private and satisfies nullifier pseudorandomness.*

Proof. First, we note that changing the order of merge in the tree T does not affect the resulting merge transaction. Let T' be a variant of T where any two transactions are swapped — then $\mathsf{EquivTree}(T, T') = 1$ and $\mathsf{EvalTree}(\mathsf{st}, T) = \mathsf{EvalTree}(\mathsf{st}, T')$. The first statement can be verified by manually checking all the predicates and making sure they are indifferent to the order of the leaves. As for the evaluation equality statement, first note that all the leaves will be processed in the same way and with respect to the same st irrespectively of their order. So we only need to argue that $\mathsf{MergeTx}$ and $\mathsf{MergeSig}$ are commutative — which is trivial since they are only uniting sets and taking sums (which are commutative operations on their own).

Hence we can represent T as a merge of two transactions, coming from subtrees $T_{\mathcal{A}}$ and $T_{\mathcal{H}}$. $T_{\mathcal{A}}$ contains exactly the same leaves for both T_0 and T_1 if they are equivalent. Therefore, we only need to show the indistinguishability of transactions $\mathsf{tx}_{\mathcal{H},0}$ and $\mathsf{tx}_{\mathcal{H},1}$, created from $T_{\mathcal{H},0}, T_{\mathcal{H},1}$ correspondingly. If $T_{\mathcal{H}}$ is empty in one case (contains no leaves), it must be empty in the other case, due to the restriction on input and output size, B_2, in $\mathsf{EquivTree}$. And thus if $\mathsf{tx}_{\mathcal{H},b}$ is empty, $\mathsf{tx}_{\mathcal{H},1-b}$ should be empty too for each $b \in \{0, 1\}$, in which case the privacy proof is trivial, since adversarial transactions are constructed in exactly the same manner in both worlds. Therefore, assume that there is at least one input or output in an honest transaction in both cases. Next, observe that:

$$\begin{aligned}\mathsf{MergeSig}(\{\mathsf{SignTx}(\mathsf{st}, \mathsf{BuildPTx}(\mathsf{st}, I_i, O_i, \mathsf{SK}))\}_i) = \\ \mathsf{SignTx}(\mathsf{st}, \mathsf{BuildPTx}(\mathsf{st}, I' = \bigcup I_i, O' = \bigcup O_i, \mathsf{SK}))\end{aligned}$$

Because of this homomorphic property (and a similar one for $\mathsf{MergeTx}$), we can assume that both $\mathsf{tx}_{\mathcal{H},b}$ and their signatures are just a direct output of $\mathsf{CompleteTx}$ and SignTx (w.r.t. (I', O')), and no merging is involved. The form of $\mathsf{tx}_{\mathcal{H},b}$ that $\mathcal{A}$ receives is

$$\big(\{\mathsf{nul}_i\}^{|\mathcal{S}|},\ \{(\mathsf{note}_i, C_i)\}^{|\mathcal{T}|},\ \{\Delta_{\mathsf{ty}}\},\ \sigma_i @ \big(\{(\pi_i^{\mathcal{S}}, \mathsf{com}_i^{\mathcal{S}})\}^{|\mathcal{S}|}, \{(\pi_i^{\mathcal{T}}, \mathsf{com}_i^{\mathcal{T}})\}^{|\mathcal{T}|}, \mathsf{rc}\big)\big)$$

We first argue informally why this transaction looks the same for both trees.

1. set sizes: number of inputs and outputs $|\mathcal{S}|$ and $|\mathcal{T}|$ are the same, guaranteed by B_2, and by the fact these dimensions are just summed when transactions are merged,
2. size and content of the $\{\Delta_{\mathsf{ty}}\}$ set is the same by B_1,
3. the set of (honest) input nullifiers $\{\mathsf{nul}_i\}^{|\mathcal{S}|}$ contains: nullifiers that $\mathcal{A}$ received from $\mathcal{O}_{\mathsf{Spend}}$ — these are the same by B_4; and nullifiers unknown previously to $\mathcal{A}$ — deterministic but indistinguishable by nullifier pseudorandomness,
4. $\{(\mathsf{note}_i, C_i)\}^{|\mathcal{T}|}$ are either: belonging to adversarial keys in which case both the notes and ciphertexts have the same distribution, as guaranteed by B_5 — these notes contain the same values, but created in both trees with uniformly sampled randomness, so are indistinguishable; the same applies to the adversarial ciphertexts. Or the notes belong to the honest keys, in which case they are (together with the ciphertexts) indistinguishable by the OTA privacy,
5. in the signatures σ_i:
   (5.1) by zero-knowledge, proofs $\pi_i^{\mathcal{S}}$ and $\pi_i^{\mathcal{T}}$ can be simulated in both worlds, so they are indistinguishable,
   (5.2) intermediary commitments $\mathsf{com}_i^{\mathcal{S}}, \mathsf{com}_i^{\mathcal{T}}$ can contain just zero (except for one chosen commitment for each type which must contain Δ_{ty} to balance out the public imbalance), and this is indistinguishable by $\mathrm{HID\text{-}OR}$— commitment hiding with open randomness (by a variation with n elements involved simultaneously, as described in Lemma 5.2.1).

(5.3) joint randomness rc — the total randomness is always the sum of the individual rc_i of internal nodes. If at least one node in the tree is honest, the final rc is uniform in both worlds, so perfectly indistinguishable. If there are no honest nodes, then the final rc is exactly equal in both worlds (by B_3 malicious leaves must be the same).

The formal reduction is as follows. We start from $\mathsf{Privacy}^0_{\mathcal{A}}(1^\lambda)$.

$\underline{\mathcal{G}_1}$**:** Replace all honest NIZKs (created inside $\mathcal{O}_{\mathsf{Spend}}$ and SignTx) to simulated NIZKs.

$\underline{\mathcal{G}_2}$**:** By nullifier pseudorandomness, replace all the honest nullifiers $\mathcal{A}$ has not seen through $\mathcal{O}_{\mathsf{Spend}}$ by the nullifiers over a different set of secret keys.

To switch an individual $\mathsf{NulEval}(\mathsf{sk}_0, r_0)$ to $\mathsf{NulEval}(\mathsf{sk}_1, r_1)$, we first pick a random function f and switch *all* evaluations of $\mathsf{NulEval}(\mathsf{sk}_0, r)$ to $f(r)$ (for all r). Since the nullifiers in $\mathsf{tx}^{\mathcal{H}}$ have not been queried previously in $\mathcal{O}_{\mathsf{Spend}}$, the evaluation of $f(r_0)$ is the first one in the game. Hence, it is equivalent to returning a random value ψ instead of $f(r_0)$. Then, again by pseudorandomness, we return all the other (not related to $\mathsf{tx}^{\mathcal{H}}$) evaluations of $f(\cdot)$ back to $\mathsf{NulEval}(\mathsf{sk}_0, \cdot)$. We now perform exactly the same steps then for the second key, de-idealizing ψ into $\mathsf{NulEval}(\mathsf{sk}_1, r_1)$. First we replace all evaluations of $\mathsf{NulEval}(\mathsf{sk}_1, r)$ to $f(r)$ (for all r), then we observe that r_1 has not yet been queried to $f(\cdot)$, so it is equivalent to return $f(r_1)$ instead of ψ. Then we replace all $f(r)$ back to $\mathsf{NulEval}(\mathsf{sk}_1, r)$, *including* the target nullifier in $\mathsf{tx}^{\mathcal{H}}$. This is how we have just replaced $\mathsf{NulEval}(\mathsf{sk}_0, r_0)$ by $\mathsf{NulEval}(\mathsf{sk}_1, r_1)$.

Repeat the procedure for all pairs of nullifiers in $\mathsf{tx}^{\mathcal{H}}$ in any order (i.e. it does not matter what the source and target of the replacement is since all values are pseudorandom).

$\underline{\mathcal{G}_3}$**:** Replace output notes and related ciphertexts to honest secret keys by the notes and ciphertexts for T_1. Recall that output notes to $\mathcal{A}$ keys must have exactly the same inputs in both cases, so their distribution is exactly equal. This step relies on the OTA privacy and again on nullifier pseudorandomness.

By OTA privacy one can replace $\mathsf{OTA.Gen}(\mathsf{pk}_b, (\mathsf{ty}_b, v_b), r)$ and $\mathsf{OTA.Enc}(\mathsf{pk}_b, (\mathsf{ty}_b, v_b, r), \xi)$ from $b = 0$ to $b = 1$ even if there are $\mathsf{Receive}$ calls in the game before and after the replacement. (After, we cannot query the challenge note in a CCA fashion, which is irrelevant since we replace it in the very end of the game.)

The only detail left now is that there are also nullifier evaluations that use the challenge sk_b, but by nullifier pseudorandomness we can replace these nullifiers by consistent random values before the OTA privacy switch (we can do that in the presence of Receive evaluations because of the Receive oracle in the pseudorandomness definition), perform the privacy switch, and then return the real NulEval evaluations back.

$\underline{\mathcal{G}_4}$**:** Replace intermediary commitments com with the values from T_1. Here we apply $\mathrm{HID\text{-}OR}$ for vectors, as described in Corollary 5.2.1.1: replacing one set of typed commitments with another, given that $\Delta_{\mathsf{ty},0} = \Delta_{\mathsf{ty},1}$ for each type. This transition is perfect.

$\underline{\mathcal{G}_5}$**:** Remove the simulation, create real proofs according to the new data in T_1. From the first game, we have not modified anything outside of the scope of EvalTree, so all NIZKs in $\mathcal{O}_{\mathsf{Spend}}$ queries are "restored" from simulations without any issue, since they are for exactly the same data. Now it is a matter of a completeness check to make sure that the new transaction constructed in EvalTree has the same distribution as $\mathsf{tx}_{\mathcal{H},1}$ except for the (yet simulated) proofs, and in particular it satisfies $\mathcal{L}^{\mathsf{spend}}$ and $\mathcal{L}^{\mathsf{output}}$. So now we enable real NIZKs back as well, and by zero-knowledge we obtain honest proofs for T_1.

Now $\mathcal{G}_5$ is equivalent to $\mathsf{Privacy}^1_{\mathcal{A}}(1^\lambda)$, which concludes the privacy proof. □

5.7 Implementation

To show that our construction is practical, we developed a prototype implementation available online[13]. We use the rust framework ark-works[14] and their Groth16 SNARK library. For an efficient hashing circuit, we use Poseidon [Grassi et al., 2019]. As the construction is similar to the Zcash Sapling version, we use this as performance comparison. Our implementation differs in various aspects that aren't material to the core protocol, including the SNARK implementation, hash functions used, and commitments used in the coin commitment Merkle tree. The produc-

[13] `https://github.com/felix-engelmann/zswap-code`
[14] `https://arkworks.rs`

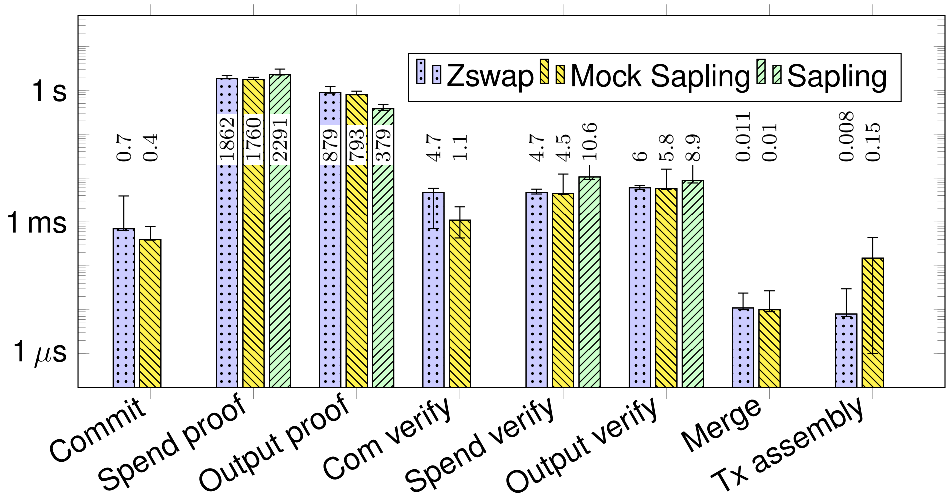

Figure 5.9: Comparison of our protocol to our Mock Sapling implementation with the same hash function and the original Zcash. The measurements of procedures is in milliseconds.

tion Zcash Sapling implementation[15] uses less performant variants for backwards compatibility and integration purposes. A direct performance comparison to Zcash is therefore misleading, and we introduce an artificial "Mock Sapling" code base, that re-implements the Zcash Sapling cryptographic protocol but uses the same primitives as we do. We achieve this by removing the types and type commitments from our construction. The comparison then allows us to detect the performance impact of our changes to the cryptographic protocol compared to Sapling without the measurement error caused by different implementations.

For spend proofs, our protocol requires 25,416 gates, 10% more than Mock Sapling (23,084). For output proofs, we use 14,852 gates, 18% more (12,520). In both cases the difference of 2,332 gates consists of a hash-to-curve operation to associate the type with a curve point. The rest of the constraints further break down into two dominant groups: Symmetric cryptography (hashes, commitments, and Merkle tree verification), and group operations related to the binding signature. For spend proofs, symmetric operations dominate with 66% of constraints (16,935), primarily from the Merkle tree verification (using a tree of height 32). The group operations cover almost all of the remaining 24% of constraints (with 6,114). For output proofs, group operations make up 41% of the constraints (with the same 6,114 constraints), and symmetric operations 43% of constraints (6,403).

[15] `https://github.com/zcash/zcash`

The median, min and max runtimes of 30 executions on a 6th generation i7 CPU@2.7GHz are summarized in Figure 5.9. The times are for creating Commitments, Spend proofs and Output proofs. For verification, we measure the homomorphic commitment comparison, the Spend proof verification and the Output proof verification. Without comparison to Sapling are Merging transaction and transaction assembly. The SNARKs proving time takes around two seconds for each input and one second for each output, dominating the transaction generation. The verification is noticeably slower with support for types but approximately equal to Zcash. Overall, we notice that the impact of the additional constraints required for our protocol are minimal while providing additional functionality.

Chapter 6

Exploding Commitments and Applications to AML

This chapter is based on the ongoing work co-authored by Bernardo David, Felix Engelmann, Tore Frederiksen, Markulf Kohlweiss, and Elena Pagnin.

In this chapter we introduce and study the notion of an *exploding commitment scheme* (ECS), which allows parties in possession of a hint to privately extend a verifiable sequence of commitments.

An exploding commitment scheme is protocol between a set of users which homomorphically update a commitment, and an auditor which wants to know a result of a certain predicate on the committed value. ECS allows the auditor to set a secret threshold t, and later to learn an escrow ecom containing *only* the predicate evaluated on the sum of committed values $P(t, \sum x_i)$. In this case the predicate is positive we say that the commitment "exploded". The important thing is that this escrow ecom, in its more "raw" form that we call a hint, can be updated — thus the users can add their x_i to the escrow without knowing anything about t. In addition, every hint update produces a so-called commitment tag. Both the parties and the auditor can verify that a given sequence of commitment tags, that we call a *history*, has been obtained by successive updates departing from an initial auditing public key. In case the auditor detects that commitment for a given history explodes, they can request an escrow for a prefix of that history (*i.e.* before the latest update) or request the opening of individual commitments in the sequence the results of

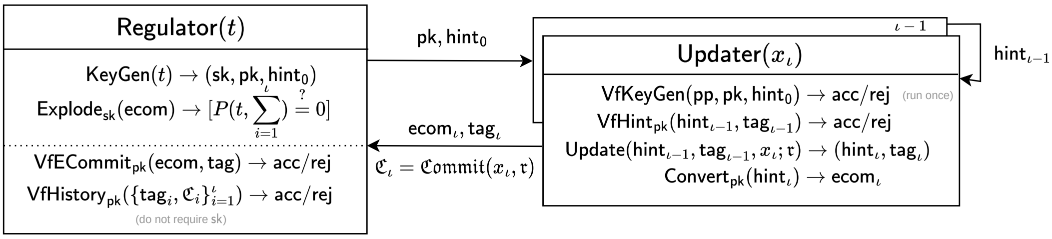

Figure 6.1: Illustration of general workflow of an Exploding Commitment Scheme for $P(t, x)$. Note that authority only learns the predicate value, while parties only know their own x_i, but neither t nor the other parties' x_i. The two bottom methods on the authority's side do not require sk and can be in fact called by a third party if necessary.

which will be correct and consistent by its binding and soundness properties. In terms of privacy, an ECS hides the committed values, and the auditor's choice of predicate. In terms of efficiency, all commitment updates, verification of updates, openings and predicate evaluations can be done non-interactively. This notion can be seen as a multi-user extension of privacy-preserving blueprints [Kohlweiss et al., 2023]. We give detailed security definitions of our new updatable notion in Section 6.2. For a deeper but high-level perspective, see the technical overview in Section 6.0.1. The diagram in Fig. 6.1 summarizes dependencies between different methods in our construction. Parts of it, such as the role of Commit (base commitment scheme), are deferred to the aforementioned more technical sections.

In the context of AML, an ECS can be used to securely compute and validate a joint suspiciousness score for transactions across multiple banks. In this application, an exploding commitment scheme accumulates a suspiciousness score that is appended to each transaction. Upon receiving a transaction to one of its accounts, the sending bank can augment the transaction with a suspiciousness-contribution commitment based on the suspiciousness score of the sending account. This allows the receiving bank to obliviously update the suspiciousness score in the exploding commitment associated with the receiving account. At any moment, an auditor (*e.g.* a tax authority) can validate the sequence of commitments held by each bank and check if they satisfy a predicate that is true if the score exceeds a certain threshold. If so, the auditor can further request the opening of related commitments held by other banks. All of these operations do not require the users nor the banks to perform any extra rounds of communication and only add a minimal overhead to the existing communication and computation involved in transfers. A detailed discussion of this and other applications is given in Section 6.6.

The contributions of this chapter are as follows:

- In Section 6.1 we discuss updatability for algebraic NIZKs, concretely focusing on the CH20 NIZK [Couteau and Hartmann, 2020]. This NIZK has been known to be updatable ([Connolly et al., 2022]), but in this chapter we believe to be first to approach it in a generalized and precise manner. Updatability of CH20 is extensively used in our main efficient protocol.

- In Section 6.2 we provide a formalization of the exploding commitment scheme, defining its semantics and security properties. One complexity of our definition comes from their interplay especially around extractability properties. For example, in soundness we do not require exploding commitments to be extractable, but assume that the tags and external commitments actually contain the "update values", which is important for interoperability.

- In Section 6.3 we present a concretely efficient ECS that allows for additive updates and for computing the threshold predicate used in the aforementioned AML application. This construction critically relies on the updatability property of [Couteau and Hartmann, 2020]. We discuss efficiency of our construction in Section 6.5.

- Finally, Section 6.6 dicusses possible extensions and applications of our primitive. While giving an overview of our techniques, we also discuss how a generic (but inefficient) construction of an ECS scheme for any NP predicate can be obtained from powerful primitives such as fully homomorphic encryption (FHE) and non-interactive zero knowledge (NIZK).

Comparison with related work. A series of recent papers explored accountable law enforcement access system [Goldwasser and Park, 2018, Frankle et al., 2018, Scafuro, 2019, Green et al., 2021, Bartusek et al., 2023, Kohlweiss et al., 2023]. Exploding commitments are most closely related to a subcategory of these works on abuse-resistant regulation compliance [Frankle et al., 2018, Bartusek et al., 2023, Kohlweiss et al., 2023] in which ordinary users have full privacy even against a malicious regulator/auditor. A consequence of full privacy for ordinary users is that the auditors detection policy must be kept private. Otherwise, malicious users could adapt their behavior and avoid detection. This means that the problem is inherently about securely computing a function on private inputs from the user and the regulator.

Contrary to previous work on abuse-resistant regulation compliance, our protocol involves private data of multiple users. In particular, it can be seen as a multi-user extension of blueprint matching, restricted to predicates on aggregate values. A private blueprint scheme [Kohlweiss et al., 2023] allows a user to create an escrow that only reveals the value $f(t, x)$ where t is kept private to the auditor and x is kept private to the user. When f corresponds to the predicate $P(t, x)$ defined as $x \in [t, t + d]$ where $d > x$, then their construction gives a one epoch exploding commitment scheme for $P(t, x)$. Notice that the predicate can only be tested on an input that is fully known to a single user. This is insufficient when inputs come from multiple mutually distrusting users. Exploding commitments extend blueprints with an updatable hint mechanism, that allows iteratively computing the value of the predicate $P(t, x_1 + \cdots + x_n)$.

The concrete threshold predicate for which we present an efficient ECS is closely related to Yao's Millionaire's Problem, *i.e.* performing secure comparison. There are many protocols for secure comparison (*e.g.*, [Damgard et al., 2008, Garay et al., 2007]) that could be used to detect whether an aggregate committed value is higher than a threshold in a privacy preserving manner. However, this would require continuous online involvement of parties and many rounds of interaction. In our setting, no interaction is required from the users after they update commitments and the auditor only needs to come online to check if commitments have exploded.

Updatable NIZKs, such as CH20 [Couteau and Hartmann, 2020] used as our prime technical tool, have been previously investigated in [Belenkiy et al., 2009, Chase et al., 2012, Chase et al., 2013a, Khalili et al., 2019]. While recursive approach to NIZK updatability becomes more practical over time [Bitansky et al., 2013, Ben-Sasson et al., 2014b, Chase et al., 2014, Bowe et al., 2019, Bünz et al., 2020c, Bünz et al., 2020b, Bünz et al., 2021a, Kothapalli et al., 2021], direct malleability without recursion is more lightweight and thus more suitable for tailored application, such as various signature schemes [Dodis et al., 2010, Blazy et al., 2011, Fuchsbauer, 2011, Khalili et al., 2019], anonymous credentials [Acar and Nguyen, 2011], scalable mix-nets [Hébant et al., 2020] etc. The malleability of CH20 was observed in [Connolly et al., 2022] to build anonymous credentials and structure-preserving signatures on equivalence classes. It is also worth noting that RO-based NIZKs are essentially non-malleable unless recursively [Faust et al., 2012, Kohlweiss and Zając, 2021, Ganesh et al., 2022a]. This is why the CH20 NIZK combining the simplicity of a Schnorr-like proof, together with a bilinear setup avoiding the ROM limitation, stands out as a natural candidate for direct updatability.

The notions of fully homomorphic commitments [Damgård et al., 2014, Cascudo et al., 2019] and of additively homomorphic functional commitments [Catalano et al., 2022] are also closely related to exploding commitments. Both notions allow for commitments to be updated by adding private values and for predicates to be computed on committed values without revealing them. However, these schemes require knowledge of the commitment opening for performing updates and predicate evaluations. This requirement precludes the use of such schemes in our multi-user setting, where individual user inputs/updates must be kept private.

A different cryptographic approach for privacy-preserving AML is based on the private computation of similarity scores of transaction graphs [Gama et al., 2020, de Perthuis and Pointcheval, 2022]. The latter work can be seen as a specialized primitive, a two-client inner-product functional encryption scheme that enables non-interactive computation of similarity scores, thus avoiding the full power and interaction of multi-party computation. As computing the similarity between all accounts is expensive, the authors of [de Perthuis and Pointcheval, 2022] suggest that banks preselect accounts. Out techniques can be seen as such a preselection mechanism based on aggregate suspiciousness scores.

6.0.1 Technical Overview

The notion of exploding commitments. An exploding commitment scheme is defined in terms of a base commitment scheme BC $= (\mathfrak{Setup}, \mathfrak{Commit})$ and a public "explosion" predicate $P(T, X)$. The base commitment scheme is needed for interoperability of our ECS scheme with external protocols: by exposing commitments to the update values we allow external protocols to "talk" about these values. The predicet $P(T, X)$ is taking two private inputs: an auditor value t and an aggregate value $\sum_{i=1}^{\iota} x_i$ (where ι denotes the lastest epoch). For a chosen t, the history of base commitments $\mathfrak{C}_1, \ldots, \mathfrak{C}_\iota$ is meant to "explode" if the contained values $x_1, \ldots, x_\iota$ satisfy $P(t, \sum_i x_i) = 1$. The auditor R generates a key pair $(\mathsf{sk}, \mathsf{pk})$

and a hint_0 for predicate $P(t,\cdot)$, where sk and t remain private but pk is published while hint_0 is passed to the first user. Hint values enable users to append base commitments to a history of commitments. Using pk and $\mathsf{hint}_{\iota-1}$, a user can extend the history with their value x_ι, each time deriving an updated hint_ι. for extending the history further.

In our construction, $\mathsf{hint}_{\iota-1}$ will contain the aggregate value $\sum_{i=1}^{\iota-1} x_i$ that users can "update" via homomorphic operations to add their own private input x_ι. A hint can be transformed into a predicate escrow ecom, which reveals nothing to the users but can be used by an auditor who knows sk to learn *only* whether $P(t,\sum_i x_i)=1$, while keeping the x_i values private. Moreover, when a user updates $\mathsf{hint}_{\iota-1}$ to hint_ι, they obtain an update tag (tag_ι) for verifying the validity and consistency of commitment histories by $\mathsf{VfHistory}_{\mathsf{pk}}(\{\mathsf{tag}_i,\mathfrak{C}_i\}_{i=1}^{\iota})$, hints by $\mathsf{VfHint}_{\mathsf{pk}}(\mathsf{hint}_\iota,\mathsf{tag}_\iota)$, and predicate escrows by $\mathsf{VfECommit}_{\mathsf{pk}}(\mathsf{ecom}_\iota,\mathsf{tag}_\iota)$. We have the following three properties: 1. tag_ι can only verify for a single history, i.e., valid histories do not collide in their tags. 2. from a valid history we can extract openings for all its base commitments. 3. if ecom verifies with respect to tag_ι, then it indeed is an escrow of predicate value $P(t,\sum_i x_i)$ for the openings x_i of these base commitments.

We require hints, predicate escrows, and tags derived from valid hints to be hiding. However, as hints contain the aggregate value $\sum_{i=1}^{\iota} x_i$, hiding for hints only holds against adversaries who do not know sk. Thus hints should only be used for updates between users but not given to the auditor. Finally, our scheme preserves the hiding and binding properties of the base commitment scheme.

A generic but inefficient construction. Departing from circuit private Fully Homomorphic Encryption (FHE) and non-interactive zero knowledge (NIZK), we can construct a ECS for arbitrary predicates and updates. The auditor generates a FHE key pair $(\mathsf{pk},\mathsf{sk})$ and publishes pk along with an encryption of t under pk as the ECS public key. Users encrypt their private inputs x_i under pk and use the resulting ciphertext to obtain an updated hint_i by homomorphically evaluating the update function on this ciphertext and hint_{i-1}. The update tag tag_i can be obtained by generating a NIZK showing that hint_i was correctly computed as an update of a previous hint. To generate a predicate escrow ecom, a user homomorphically evaluates the predicate $P(T,X)$ on hint_i and the ciphertext containing t (obtained from the public key) and computes the corresponding tag as a NIZK showing that the predicate was correctly evaluated. Using sk the auditor can decrypt ecom and learn only the output of $P(t, \sum_i x_i)$ but nothing else, while the validity of ecom can be checked by verifying the tag NIZK. This scheme provides support for arbitrary predicates and updates (beyond additive ones) but clearly has a very high concrete computational complexity due to the use of heavy primitives such as FHE and NIZKs for homomorphic computation on FHE ciphertexts. This is a serious issue in a setting such as that of AML, where hundreds of thousands of updates must be processed per second.

An efficient construction for threshold predicates. If we focus on the case of a predicate $P(T, X)$ that outputs 1 if $X > T$, we can obtain a concretely efficient ECS construction. The core of our scheme is a mechanism that relies on Pedersen commitments for predicate evaluations in the multi-user scenario, guaranteeing soundness of the updates.

The auditor starts by generating a polynomial $P_d(t, X)$ with roots at positions $\{t, t+1, \ldots t+d-1\}$ where d is the size of the "exploding range" and t is the auditor's secret threshold. It then ElGamal encrypts the powers $(-t)^i$ for $i \in [d]$, in the exponent. The ciphertexts are included in the initial hint hint_0. Users can combine the ciphertexts to homomorphically evaluate the polynomial $P_d(t, x)$ at their input x, in the exponent. Moreover, given $\mathsf{hint}_{\iota-1}$ – which contains an encryption of $P_d(t, x)$ – a user can update the hint to hint_ι containing an encryption of $P_d(t, x + x')$. This is achieved by standard algebraic transformation on the polynomial representation, as described in Section 6.3.1.0.3.

When converting hints to escrows, users homomorphically reconstruct the ElGamal ciphertext for the value $P_d(t, x)$, and then exponentiate it by a random β obtaining an encryption of $\beta \cdot P_d(t, x)$. If the polynomial evaluates to 0, the randomization has no effect and the auditor is able to decrypt the message G^0 (this is the "explosion" case, since the predicate $P(T, X) = [P_d(T, X) \stackrel{?}{=} 0]$ outputs 1). On the other hand, if the evaluation of the polynomial is not a root, the auditor learns nothing.

The main technical achievement of this work is the design of hints that do not grow with the number of updates, and are reasonably sized for practical applications. We make hints linear in the degree d of the polynomial and endow them with *updatable* proofs that attest to hint consistency. Intuitively, this is achieved by including witness-products in the witness and checking the consistency of those products w.r.t. the minimal witness values by using Pedersen commitments and adding extra checks in the relation to ensure input consistency based on these commitments. Thanks to the homomorphic property of Pedersen commitments, our witness and instance are updatable: randomnesses and x values accumulate additively (see Section 6.3.2 for further details).

For the many use cases which *do require* a linear history of proofs-of-updates we provide a very concise "update trace" consisting of update tags tag, which naturally grows in the number of updates, but allows enforcing update accountability. In addition, as we use Pedersen as base commitment, it is easy to integrate our ECS constructions with applications such as credentials and private payment systems: one simply proves *extra* statements about base commitments, *e.g.*, that their values are equal to a credential attribute or a payment transaction value.

6.1 Updatable Algebraic Arguments

In this work we will use two NIZK proof systems, both of which work with the same class of languages.

- Π: The standard non-updatable Σ-protocol proof system for equality of discrete logarithm relations [Schnorr, 1990, Maurer, 2009]. It is assumed to be straight-line knowledge-sound after non-interactive transformation, e.g. by encrypting witnesses or using Fischlin's technique [Fischlin, 2005].
- Π_{u}: The CH20 NIZK [Couteau and Hartmann, 2020], which is Σ-like, but is updatable, involves bilinear pairings, and has a uniform CRS. We provide an overview of CH20 updatability and security in Section 6.1.

Let $\mathbb{G}$ be a prime ordered group. Define the set of linear polynomials $\mathcal{P} \subset \mathbb{G}[X_1 \ldots X_l]$ in l variables with coefficients in $\mathbb{G}$ as $\mathcal{P} = \{a_0 + \sum_{i=1}^{l} a_i X_i \mid a_0 \in \mathbb{G}, a_1 \ldots a_l \in \mathbb{Z}_q\}$. Both Π and Π_{u} work with the so-called *algebraic* language[1] $\mathcal{L}_M$ defined as follows:

$$\mathcal{L}_M = \left\{ \vec{\mathsf{x}} \in \mathbb{G}^l \mid \exists\, \vec{\mathsf{w}} \in \mathbb{Z}_p^t : M(\vec{\mathsf{x}}) \cdot \vec{\mathsf{w}} = \vec{\mathsf{x}} \right\}$$

where $M(\vec{X}) \in \mathcal{P}^{l \times t}$. In other words, it is a set of DLOG-like linear equations with a common instance, and bases in $M(\vec{X})$ that can potentially depend on the instance

[1] For simplicity, and contrast with [Couteau and Hartmann, 2020], we do not consider arbitrary $\Theta(\vec{\mathsf{x}})$ such that $M(\vec{\mathsf{x}}) \cdot \vec{\mathsf{w}} = \Theta(\vec{\mathsf{x}})$.

x itself. We define the corresponding relation $\mathcal{R}_M$ to be the set $\{(\vec{\mathsf{x}}, \vec{\mathsf{w}}) \in \mathbb{G}^l \times \mathbb{Z}_p^t \mid M(\vec{\mathsf{x}}) \cdot \vec{\mathsf{w}} = \vec{\mathsf{x}}\}$.

Updatability for algebraic languages $\mathcal{L}_M$, due to their group structure, means that there exist four matrices $(T_{\mathsf{xm}}, T_{\mathsf{xa}}, T_{\mathsf{wm}}, T_{\mathsf{wa}})$ such that for all $(\vec{\mathsf{x}}, \vec{\mathsf{w}}) \in \mathcal{R}_M$ it holds that

$$(T_{\mathsf{xm}} \cdot \vec{\mathsf{x}} + T_{\mathsf{xa}}, T_{\mathsf{wm}} \cdot \vec{\mathsf{w}} + T_{\mathsf{wa}}) \in \mathcal{R}_M$$

The functions $T_{\mathsf{x}}, T_{\mathsf{w}}$ required in Definition 2.6.7 are defined as follows: $T_{\mathsf{x}}(\mathsf{x}) := T_{\mathsf{xm}} \cdot \vec{\mathsf{x}} + T_{\mathsf{xa}}$ and $T_{\mathsf{w}}(\mathsf{w}) := T_{\mathsf{wm}} \cdot \vec{\mathsf{w}} + T_{\mathsf{wa}}$. We will show that the algebraic languages we define in this work are updatable by explicitly providing the matrices and proving they satisfy the equation.

We discuss updatable algebraic NIZKs in depth in the following Section 6.1.

For our main construction we use CH20 [Couteau and Hartmann, 2020] NIZK, which we refer to as Π_{u} which has controlled malleability. In this section we recall the proof system and argue why is it updatable.

The Π_{u} proof is using a uniform CRS consisting of a single group element $[z]_2$, where $z \leftarrow \mathbb{Z}_q$ is a uniformly sampled trapdoor. For a language matrix $M(\vec{X}) \in \mathcal{P}^{l \times t}$ where l is the size of the instance and t is the size of the witness, $\mathsf{Prove}([z]_2, \mathsf{x})$ returns $\pi = ([\boldsymbol{a}]_1 \in \mathbb{G}_1^l, [\boldsymbol{d}]_2 \in \mathbb{G}_2^t)$ where elements are constructed as follows:

$$[\boldsymbol{a}]_1 \leftarrow [M(\mathsf{x})]_1 \cdot \boldsymbol{s}$$
$$[\boldsymbol{d}]_2 \leftarrow [z]_2 \cdot \mathsf{w} + \boldsymbol{s}$$

where $\boldsymbol{s} \in \mathbb{Z}_q^t$ is sampled randomly. Then the proof is verified by checking running $\mathsf{Verify}([z]_2, \pi := ([\boldsymbol{a}]_1, [\boldsymbol{d}]_2))$ which checks a single equation

$$\hat{e}([M(\mathsf{x})]_1, [\boldsymbol{d}]_2) \stackrel{?}{=} \hat{e}(\mathsf{x}, [z]_2) \cdot \hat{e}([\boldsymbol{a}]_1, 1)$$

Updates in $\mathcal{L}_M$. Rephrasing the definition of updatability for algebraic languages, $\mathcal{L}$ is updatable w.r.t. transformation $(T_{\mathsf{xm}}, T_{\mathsf{xa}}, T_{\mathsf{wm}}, T_{\mathsf{wa}})$ if for all $(\mathsf{x}, \mathsf{w}) \in \mathcal{R}$, the following relation holds:

$$T_{\mathsf{xm}} \cdot \mathsf{x} + T_{\mathsf{xa}} = M(T_{\mathsf{xm}} \cdot \mathsf{x} + T_{\mathsf{xa}}) \cdot \left(T_{\mathsf{wm}} \cdot \mathsf{w} + T_{\mathsf{wa}}\right)$$

Intuitively, it means that there is a linear way to update both the instance and the witness simultaneously. One simple example is the language of Diffie-Hellman tuples $(C_1, C_2, C_3) := (g^x, g^y, g^{xy})$. The matrix M defines the following relations: $C_1 = G^x, C_2 = G^y, C_3 = C_1^y$. Then, for any α, β it holds that $(C_1^\alpha, C_2^\beta, C_3^{\alpha\beta})$ is also in the relation (this defines T_{xm} implicitly, $T_{\mathsf{xa}} = 0$), since $x' = \alpha x, y' = \beta y$ (this defines T_{wm} with α, β as diagonal, and $T_{\mathsf{wa}} = 0$).

Updates in the CH20 NIZK. Proof updatability in CH20 requires an extra condition: valid language transformation must be compatible in the following way.

Definition 6.1.1 (Blinding-Compatible Transformations)**.** *Let $\mathcal{L}_M$ be an algebraic language defined w.r.t. a matrix $M(\mathsf{x})$. A valid transformation $T := (T_{\mathsf{xm}}, T_{\mathsf{xa}}, T_{\mathsf{wm}}, T_{\mathsf{wa}})$ on $\mathcal{L}$ is said to be* blinding-compatible*, if there exists another pair of matrices $(T_{\mathsf{am}}, T_{\mathsf{aa}}) \in \mathbb{Z}_p^{l\times 2l} \times \mathbb{Z}_p^l$ such that the following equation holds for all $\mathsf{x} \in \mathcal{L}$ and all $\boldsymbol{s} \in \mathbb{Z}_p^t$:*

$$T_{\mathsf{am}} \cdot \binom{M(\mathsf{x}) \cdot \boldsymbol{s}}{\mathsf{x}} + T_{\mathsf{aa}} = M(T_{\mathsf{xm}} \cdot \mathsf{x} + T_{\mathsf{xa}}) \cdot \Big(T_{\mathsf{wm}} \cdot \boldsymbol{s} + T_{\mathsf{wa}}\Big)$$

Some valid T are *trivially blinding-compatible*: a simple $(T_{\mathsf{am}}, T_{\mathsf{aa}}) := (T_{\mathsf{xm}} \mid 0^{l\times l}, T_{\mathsf{xa}})$ will satisfy the equation; but this is not true generally.

Note that this equation is quite different from the one for language updatability. First, it applies to all $\boldsymbol{s}$ completely independently from the instance x. Second, $\boldsymbol{s} \in \mathbb{Z}_p^t$ is uniform and might not be a valid witness (e.g. $\mathcal{L}$ may require $w_2 = w_1^2$, but the equation must work for completely independent s_1, s_2). Finally, T_{am} applies not only to $M(\mathsf{x}) \cdot \boldsymbol{s}$, but also to the instance x itself; this actually relaxes the requirement.

Overall, this much stronger condition is necessary due to the linearity that Schnorr-like NIZKs use for blinding w with a uniform $\boldsymbol{s}$.

Define $\mathsf{Update}(([\boldsymbol{a}]_1, [\boldsymbol{d}]_2), T := (T_{\mathsf{am}}, T_{\mathsf{aa}}, T_{\mathsf{xm}}, T_{\mathsf{xa}}, T_{\mathsf{wm}}, T_{\mathsf{wa}}))$ as a function returning $\pi' = ([\boldsymbol{a}']_1, [\boldsymbol{d}']_2)$ constructed as follows:

$$[\boldsymbol{a}']_1 = T_{\mathsf{am}} \cdot \begin{pmatrix} [\boldsymbol{a}]_1 \\ \mathsf{x} \end{pmatrix} + [T_{\mathsf{aa}}]_1 + [M(\mathsf{x})\hat{\boldsymbol{s}}]_1$$
$$[\boldsymbol{d}']_2 = T_{\mathsf{wm}} \cdot [\boldsymbol{d}]_2 + [z]_2 T_{\mathsf{wa}} + [T_{\mathsf{wa}}]_2 + [\hat{\boldsymbol{s}}]_2$$

where $\hat{s}$ is sampled uniformly at random.

Theorem 6.1.1 (Update Completeness of CH20). *The CH20 proof system* Π_{u} *with* Update_0 *satisfies update completeness with respect to all blinding-compatible transformations* $T = (T_{\mathsf{am}}, T_{\mathsf{aa}}, T_{\mathsf{xm}}, T_{\mathsf{xa}}, T_{\mathsf{wm}}, T_{\mathsf{wa}})$ *on an algebraic language* $\mathcal{L}_M$.

Proof. Consider first the case with $\hat{\boldsymbol{s}} = 0$ — this element rerandomises the proof, which we will cover in the second step. Observe how the new proof elements are expressed in terms of the old instance and witness:

$$[\boldsymbol{a}']_1 = T_{\mathsf{am}} \cdot \begin{pmatrix} [M(\mathsf{x})]_1 \cdot \boldsymbol{s} \\ \mathsf{x} \end{pmatrix} + [T_{\mathsf{aa}}]_1$$
$$[\boldsymbol{d}']_2 = [z]_2(T_{\mathsf{wm}} \cdot \mathsf{w} + T_{\mathsf{wa}}) + [T_{\mathsf{wm}} \cdot \boldsymbol{s}]_2 + [T_{\mathsf{wa}}]_2$$

For convenience, define $\mathsf{x}' = T_{\mathsf{xm}} \cdot \mathsf{x} + T_{\mathsf{xa}}$, $\mathsf{w}' = T_{\mathsf{wm}} \cdot \mathsf{w} + T_{\mathsf{wa}}$, $\boldsymbol{s}' = T_{\mathsf{wm}} \cdot \boldsymbol{s} + T_{\mathsf{wa}}$. Let us check the verification equation directly; looking at the left and right hand sides separately:

$$\begin{aligned} \hat{e}([M(\mathsf{x}')]_1, [\boldsymbol{d}']_2) &= [z \cdot M(\mathsf{x}') \cdot (T_{\mathsf{wm}} \cdot \mathsf{w} + T_{\mathsf{wa}}) + M(\mathsf{x}')(T_{\mathsf{wm}} \cdot \boldsymbol{s} + T_{\mathsf{wa}})]_3 \\ \hat{e}(\mathsf{x}', [z]_2) \cdot \hat{e}([\boldsymbol{a}']_1, 1) &= [z \cdot \mathsf{x}' \qquad + T_{\mathsf{am}} \cdot \begin{pmatrix} M(\mathsf{x}) \cdot \boldsymbol{s} \\ \mathsf{x} \end{pmatrix} + T_{\mathsf{aa}}]_3 \end{aligned}$$

Since T is a valid transformation, we first have that $M(\mathsf{x}') \cdot \mathsf{w}' = \mathsf{x}'$. Since, furthermore, T is blinding-compatible, we have $T_{\mathsf{am}} \cdot \begin{pmatrix} M(\mathsf{x}) \cdot \boldsymbol{s} \\ \mathsf{x} \end{pmatrix} + T_{\mathsf{aa}} = M(\mathsf{x}')(T_{\mathsf{wm}} \cdot \boldsymbol{s} + T_{\mathsf{wa}})$. This means that the verification equations will be satisfied by $(\boldsymbol{a}', \boldsymbol{d}')$.

Now, considering $\hat{\boldsymbol{s}} \neq 0$, observe the form of the updated proof elements:

$$[\boldsymbol{a}'']_1 = [\boldsymbol{a}']_1 + [M(\mathsf{x})\hat{\boldsymbol{s}}]_1$$
$$[\boldsymbol{d}'']_2 = [\boldsymbol{d}']_2 + [\hat{\boldsymbol{s}}]_2$$

This merely sets the new challenge randomness to $\boldsymbol{s}' + \hat{\boldsymbol{s}}$, essentially rerandomising the proof. □

Note that setting $(T_{\mathsf{am}}, T_{\mathsf{aa}}, T_{\mathsf{xm}}, T_{\mathsf{xa}}, T_{\mathsf{wm}}, T_{\mathsf{wa}}) \leftarrow (I_l | 0^{l \times l}, 0^l, I_l, 0^l, I_t, 0^t)$ (where I_l is the identity matrix of size l) turns Update into a rerandomisation function without transforming the instance.

Theorem 6.1.2 (Derivation Privacy of CH20). *The CH20 proof system* Π_{u} *with* Update *satisfies derivation privacy for any valid transformation* T *on algebraic languages.*

Proof. This follows from the form of the update function, and the remark on randomisation. The updated proofs which are additionally randomised are distributed in the same way as the honest proofs, because randomisation has exactly the same form as the honest Prove. □

6.1.1 Example of Updatability for Algebraic Languages

We will illustrate updatability of algebraic languages and Π_{u} on the example of a concrete small language encoding a Diffie-Hellman tuple. Given a group $\mathbb{G}_1$ of finite prime order p with a generator G, define the language as follows:

$$\mathcal{L}_{\mathsf{dh}} := \{(C_1, C_2, C_3) \mid \exists (a, b) \text{ s.t. } C_1 = G^a \wedge C_2 = G^b \wedge C_3 = G^{ab}\}$$

It is clear that the relation $\mathcal{R}_{\mathsf{dh}}$ is hard under DDH in $\mathbb{G}_1$, therefore proving $\mathsf{x} \in \mathcal{L}_{\mathsf{dh}}$ is non-trivial.

We start by formally defining how $\mathcal{L}_{\mathsf{dh}}$ is expressed algebraically:

$$\begin{gathered}\mathsf{x} = (C_1, C_2, C_3) \in \mathbb{G}_1^3 \\ \mathsf{w} = (a, b) \in \mathbb{Z}_p^2\end{gathered} \qquad \text{and} \qquad M(\mathsf{x}) = \begin{bmatrix} G & 0 \\ 0 & G \\ 0 & C_1 \end{bmatrix}$$

Recall that our formalisation defines $M(\vec{X}) \in \mathcal{P}^{l \times t}$ — in our example, the only instance variable in M is C_1, and all other elements are constants. It is easy to see that for a fixed (x, w) the fact that $M(\mathsf{x}) \cdot \mathsf{w} = \mathsf{x}$ directly implies $(\mathsf{x}, \mathsf{w}) \in \mathcal{R}_{\mathsf{dh}}$.

First, examining transformations on $\mathcal{L}_{\mathsf{dh}}$ we observe that the following matrices can be used:

$$T_{\mathsf{wm}} = \begin{bmatrix} \gamma & 0 \\ 0 & \delta \end{bmatrix} \quad T_{\mathsf{xm}} = \begin{bmatrix} \gamma & 0 & 0 \\ 0 & \delta & 0 \\ 0 & 0 & \gamma\delta \end{bmatrix} \quad T_{\mathsf{wa}} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad T_{\mathsf{xa}} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

With them we can derive the new instance $\mathsf{x}' := T_{\mathsf{xm}} \cdot \mathsf{x} = (G^{\gamma a}, G^{\delta b}, G^{(\gamma a)(\delta b)})$ that corresponds to the transformed witness $\mathsf{w}' := T_{\mathsf{wm}} \cdot \mathsf{w} = (\gamma a, \delta b)$. Indeed, as required by Definition 2.6.7, with these two matrices the following equation holds:

$$M(T_{\mathsf{xm}} \cdot \mathsf{x}) \cdot \big(T_{\mathsf{wm}} \cdot \mathsf{w}\big) = T_{\mathsf{xm}} \cdot \big(M(\mathsf{x}) \cdot \mathsf{w}\big)$$

Finally, let there be a CH20 proof of $\mathsf{x} \in \mathcal{L}_{\mathsf{dh}}$ created as

$$([\boldsymbol{a}]_1, [\boldsymbol{d}]_2) = ([M(\mathsf{x})]_1 \cdot \boldsymbol{s}, [z]_2 \cdot \mathsf{w} + \boldsymbol{s}) = \left(\begin{pmatrix} [s_1]_1 \\ [s_2]_1 \\ [a \cdot s_2]_1 \end{pmatrix}, \begin{pmatrix} [za + s_1]_2 \\ [zb + s_2]_2 \end{pmatrix} \right)$$

for some random $\boldsymbol{s}$. When verifying this proof, we will compute, on each side:

$$\hat{e}([M(\mathsf{x})]_1, [\boldsymbol{d}]_2) = \begin{pmatrix} [za + s_1]_3 \\ [zb + s_2]_3 \\ [a(zb + s_2)]_3 \end{pmatrix}$$

$$\hat{e}(\mathsf{x}, [z]_2) \cdot \hat{e}([\boldsymbol{a}]_1, 1) = \begin{pmatrix} [a \cdot z]_3 \\ [b \cdot z]_3 \\ [ab \cdot z]_3 \end{pmatrix} + \begin{pmatrix} [s_1]_1 \\ [s_2]_1 \\ [a \cdot s_2]_1 \end{pmatrix}$$

which are exactly equal. The transformation is trivially blinding-compatible, which means that $(T_{\mathsf{am}}, T_{\mathsf{aa}}) := (T_{\mathsf{xm}} \mid 0^{l \times l}, T_{\mathsf{xa}})$ will do. Then:

$$([\boldsymbol{a}']_1, [\boldsymbol{d}']_2) = (T_{\mathsf{xm}} \cdot [\boldsymbol{a}]_1, T_{\mathsf{wm}} \cdot [\boldsymbol{d}]_2) = \left(\begin{pmatrix} [\gamma s_1]_1 \\ [\delta s_2]_1 \\ [\gamma a \cdot \delta s_2]_1 \end{pmatrix}, \begin{pmatrix} [z\gamma a + \gamma s_1]_2 \\ [z\delta b + \delta s_2]_2 \end{pmatrix} \right)$$

will be a (non-rerandomized) transformed proof for x'. It is easy to visually verify that the transformed proof is structured exactly the same as the original proof, but w.r.t. new instance and witness.

6.1.1.1 Introducing Additive Matrices

Our example can be further extended to commitments on DH tuples; this would use additive matrices as well. Assume $H \in \mathbb{G}_1$ is uniformly sampled.

$$\mathcal{L}_{\mathsf{dh}+} := \{(C_1, C_2, C_3) \mid \exists (a, b) \text{ s.t. } C_1 = G^a H^{r_1} \wedge C_2 = G^b H^{r_2} \wedge C_3 = G^{ab} H^{r_1 b + r_3}\}$$

Because Pedersen commitments are perfectly hiding, every triple of group elements is in the language. But proving membership in it with an argument of *knowledge* is meaningful, because it shows a way to extract a witness computationally, and by computational binding of Pedersen it will be a unique one[2]. In this case we have:

$$\begin{aligned} \mathsf{x} &= (C_1, C_2, C_3) \in \mathbb{G}_1^3 \\ \mathsf{w} &= (a, b, r_1, r_2, r_3) \in \mathbb{Z}_p^5 \end{aligned} \qquad \text{and} \qquad M(\mathsf{x}) = \begin{bmatrix} G & 0 & H & 0 & 0 \\ 0 & G & 0 & H & 0 \\ 0 & C_1 & 0 & 0 & H \end{bmatrix}$$

[2] Even though CH20 proofs are only sound, in practice this issue can be overcome with pairing CH20 with a regular Schnorr, which is ignored in this example.

The matrices we will use are:

$$T_{\mathsf{wm}} = \begin{bmatrix} \gamma & 0 & 0 & 0 & 0 \\ 0 & \delta & 0 & 0 & 0 \\ 0 & 0 & \gamma & 0 & 0 \\ 0 & 0 & 0 & \delta & 0 \\ 0 & 0 & 0 & 0 & \gamma\delta \end{bmatrix} \quad T_{\mathsf{xm}} = \begin{bmatrix} \gamma & 0 & 0 \\ 0 & \delta & 0 \\ 0 & 0 & \gamma\delta \end{bmatrix} \quad T_{\mathsf{wa}} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ r_2' \\ r_3' \end{bmatrix} \quad T_{\mathsf{xa}} = \begin{bmatrix} 0 \\ H^{r_2'} \\ H^{r_3'} \end{bmatrix}$$

To see that this matrix constitutes a valid transformation consider how the new instance and witness look:

$$\mathsf{x}' = \begin{pmatrix} C_1^{\gamma} \\ C_2^{\delta} H^{r_2'} \\ C_3^{\gamma\delta} H^{r_3'} \end{pmatrix} = \begin{pmatrix} G^{\gamma a} H^{\gamma r_1} \\ G^{\delta b} H^{\delta r_2 + r_2'} \\ G^{(\gamma a)(\delta b)} H^{(\gamma r_1)(b\delta) + (\gamma\delta r_3 + r_3')} \end{pmatrix}$$

$$\mathsf{w}' = (\gamma a, \delta b, \gamma r_1, \delta r_2 + r_2', \gamma\delta r_3 + r_3')$$

which clearly belong to the relation $\mathcal{R}_{\mathsf{dd}+}$. Proving the proof transformation is similarly to how it was done with the previous example, except now we need to consider the non-zero additive matrices. The proof is structured as follows (where h is a logarithm of H):

$$([\boldsymbol{a}]_1, [\boldsymbol{d}]_2) = ([M(\mathsf{x})]_1 \cdot \boldsymbol{s}, [z]_2 \cdot \mathsf{w} + \boldsymbol{s}) = \left(\begin{pmatrix} [s_1 + hs_3]_1 \\ [s_2 + hs_4]_1 \\ [as_2 + h(r_1 s_2 + s_5)]_1 \end{pmatrix}, \begin{pmatrix} [za + s_1]_2 \\ [zb + s_2]_2 \\ [zr_1 + s_3]_2 \\ [zr_2 + s_4]_2 \\ [zr_3 + s_5]_2 \end{pmatrix} \right)$$

Now, after updating the proof (again without rerandomisation for simplicity) it will look as follows:

$$[\boldsymbol{a}']_1 = T_{\mathsf{xm}} \cdot [\boldsymbol{a}]_1 + [T_{\mathsf{xa}}]_1 = \begin{pmatrix} [\gamma s_1 + h(\gamma s_3)]_1 \\ [\delta s_2 + h(\delta s_4 + r_2')]_1 \\ [(\gamma a)(\delta s_2) + h((\gamma r_1)(\delta s_2) + (\gamma\delta s_5 + r_3'))]_1 \end{pmatrix}$$

$$[\boldsymbol{d}']_2 = T_{\mathsf{wm}} \cdot [\boldsymbol{d}]_2 + [z]_2 T_{\mathsf{wa}} + [T_{\mathsf{wa}}]_2 = \begin{pmatrix} [z \cdot \gamma a + \gamma s_1]_2 \\ [z \cdot \delta b + \delta s_2]_2 \\ [z \cdot \gamma r_1 + \gamma s_3]_2 \\ [z \cdot (\delta r_2 + r_2') + \delta s_4 + r_2']_2 \\ [z \cdot (\gamma\delta r_3 + r_3') + \gamma\delta s_5 + r_3']_2 \end{pmatrix}$$

It is easy to verify that this is structurally equal to the fresh proof, but w.r.t. $(\mathsf{x}', \mathsf{w}')$; formally, $([\boldsymbol{a}]_1, [\boldsymbol{d}]_2) = ([M(\mathsf{x}')]_1 \cdot \boldsymbol{s}', [z]_2 \cdot \mathsf{w}' + \boldsymbol{s}')$ with $\boldsymbol{s}' = T_{\mathsf{wm}} \boldsymbol{s} + T_{\mathsf{wa}}$.

6.2 Exploding Commitments

In this section we introduce the definition of an exploding commitment scheme and provide a security model for this new primitive.

Following [Kohlweiss et al., 2023] we use non-interactive commitments to bind inputs to externally committed values. Our construction can be seen as an extension of a *base* commitment scheme. One should think of the base commitment (BC) as a standard single-base commitment scheme (e.g. Pedersen), satisfying SHC definition in Section 2.5. We use it to provide a way in our construction to prove external properties about update values. To clarify the notation however, in this chapter we will use gothic fonts to denote the algorithms. In our main instantiation we will use the Pedersen commitment scheme in group $\mathbb{G}_1$ of prime order q, which exists as part of a bilinear group setup $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$.

Definition 6.2.1 (Base Commitment Scheme). *A tuple* $\mathsf{BC} = (\mathfrak{Setup}, \mathfrak{Commit})$ *will commonly refer to the "base" non-interactive commitment scheme. By that we will mean* SHC *as per Definition 2.5.1, instantiated for a single default type* $\mathtt{ty}$ *(omitted in notation), for a message space* $\mathbb{V}$ *and randomness space* $\mathbb{R}$*. In security definitions we will assume* BC *satisfies statistical hiding and computational binding.*

An *exploding commitment scheme* extends a base commitment for message space $\mathbb{V}$, and is parameterized by a predicate $P(T, X) : \mathbb{V} \times \mathbb{V} \to \{0, 1\} \in \mathcal{P}_{\mathbb{V}}$. $\mathcal{P}_{\mathbb{V}}$ is a family of efficiently computable predicates over $\mathbb{V}$ that define when commitments "explode". For a concrete example consider $\mathbb{V} = \mathbb{Z}_q$ and the family of predicates to be range checks, i.e., predicates are parametrized by a public distance value $d \in \mathbb{Z}_q$, and a private threshold value $t \in \mathbb{Z}_q$, i.e., $P_d(t, X)$, and return 1 if $x \in [t, t + d - 1]$, and 0 otherwise. We present a diagram summarizing interaction between ECS protocols in Fig. 6.1.

Definition 6.2.2 (Exploding Commitment Scheme). *Let* $\mathsf{BC} = (\mathfrak{Setup}, \mathfrak{Commit})$ *be an base commitment defined w.r.t.* $(\mathbb{V}, \mathbb{R})$*. Let* $P(T, X) \in \mathcal{P}_{\mathbb{V}}$ *be an efficiently computable binary predicate defined over* $\mathbb{V}$*. Then an* exploding commitment scheme (ECS) *for* (BC, P) *is defined by the following set of algorithms:*

$\mathsf{Setup}(1^\lambda, \mathfrak{pp}) \xrightarrow{\$} (\mathsf{pp}, \mathsf{td})$**:** *the setup algorithm is randomized, takes as input the security parameter* λ*, and base commitment parameters* $\mathfrak{pp}$*. It returns public parameters that contain at least a description of a special value denoted by*

0. It also returns a trapdoor td *that is only used in security definitions.* pp *and* $\mathfrak{pp}$ *are implicit inputs to all other algorithms.*

$\mathsf{KeyGen}(t) \xrightarrow{\$} (\mathsf{sk}, \mathsf{pk}, \mathsf{hint}_0)$**:** *the key generation algorithm is randomized, it takes as input a threshold value* $t \in \mathbb{V}$*. It outputs a key pair* $(\mathsf{sk}, \mathsf{pk})$*; a hint* hint_0 *implicitly encoding 0 — this first hint is also the only public one.*

$\mathsf{VfKeyGen}(\mathsf{pk}, \mathsf{hint}_0) \to \mathsf{acc}/\mathsf{rej}$: *the verify key generation algorithm is deterministic, it verifies the validity of the public output of* KeyGen*.*

$\mathsf{Update}_{\mathsf{pk}}(\mathsf{hint}, \mathsf{tag}, x, \mathfrak{r}) \xrightarrow{\$} (\mathsf{hint}', \mathsf{tag}')$**:** *the update algorithm is randomized, it takes as input an hint and its tag, a value, and external base commitment randomness. When* $\mathsf{hint} = \mathsf{hint}_0$*,* $\mathsf{tag} = \perp$*. It returns an updated* hint' *and the new update* tag'*. To keep track of the update history (also called trace) of a commitment we introduce an epoch index* $\iota \geq 1$*, e.g.,* $\mathsf{Update}_{\mathsf{pk}}(\mathsf{hint}_{\iota-1}, \mathsf{tag}_{\iota-1}, x_\iota, \mathfrak{r}_\iota) \xrightarrow{\$} (\mathsf{hint}_\iota, \mathsf{tag}_\iota)$*.*

$\mathsf{VfHistory}_{\mathsf{pk}}(\{\mathsf{tag}_i, \mathfrak{C}_i\}_{i=1}^{\iota}) \to \mathsf{acc}/\mathsf{rej}$: *the verify history algorithm is deterministic, it takes as input an ordered sequence of update tags and base commitments and verifies the consistency of the update history (also called trace, see Definition 6.2.3).*

$\mathsf{VfHint}_{\mathsf{pk}}(\mathsf{hint}_\iota, \mathsf{tag}_\iota) \to \mathsf{acc}/\mathsf{rej}$: *the verify hint algorithm is deterministic, it takes as input a hint and the last corresponding update tag. It returns* acc *if the inputs are deemed to be consistent (see Definition 6.2.3), and* rej *otherwise.*

$\mathsf{Convert}_{\mathsf{pk}}(\mathsf{hint}) \xrightarrow{\$} \mathsf{ecom}$**:** *the show algorithm is randomized, it takes as input a hint for the last tag of a history. It returns a predicate escrow* ecom *that prepares the history for audit evaluation.*

$\mathsf{VfECommit}_{\mathsf{pk}}(\mathsf{ecom}_\iota, \mathsf{tag}_\iota) \to \mathsf{acc}/\mathsf{rej}$**:** *the verify escrow algorithm is deterministic, it takes as input a predicate escrow* ecom*, and the update tag* tag *used in the last update. It returns* acc *if the inputs are consistent, and* rej *otherwise.*

$\mathsf{Explode}_{\mathsf{sk}}(\mathsf{ecom}) \to 1(\mathsf{expl})/0(\mathsf{rej})$**:** *the explode algorithm is deterministic, it takes as input the auditor's secret key* sk *and a predicate escrow. It returns* expl *if the commitment history is deemed to explode (see Definition 6.2.3), and* rej *otherwise. (Intuitively, explosion is determined by the evaluation of the predicate* $P(t, \hat{x})$*, where* $\hat{x} = \sum_{i=1}^{\iota} x_i$ *is the sum of the committed values of the history).*

The notion of *correctness* covers the honest execution of the protocol. It ensures that: (1) the honestly generated key always verifies, (2) honestly updated histories of base commitments verify, and (3) the result of explosion is consistent with the evaluation of the predicate on the sum of update values.

Definition 6.2.3 (Correctness). *Let* $\lambda \in \mathbb{N}$, $\mathsf{BC} = (\mathfrak{Setup}, \mathfrak{Commit}, \mathbb{V}, \mathbb{R})$ *be a base commitment scheme, and* $P \in \mathcal{P}_{\mathbb{V}}$. *An exploding commitment scheme (ECS) for* (BC, P) *is* correct *if the following statements hold for all* $\mathfrak{pp} \xleftarrow{\$} \mathfrak{Setup}(1^\lambda)$, $(\mathsf{pp}, \cdot) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, \mathfrak{pp})$ *(remember these are implicit in all the algorithms):*

- *Full correctness: for all* $t \in \mathbb{V}$, *all poly-sized sequences of values* $x_1, \ldots, x_n \in \mathbb{V}$ *and* $\mathfrak{r}_1, \ldots, \mathfrak{r}_n \in \mathbb{R}$*:*

$$\Pr\left[\begin{array}{l} (\mathsf{sk}, \mathsf{pk}, \mathsf{hint}_0) \xleftarrow{\$} \mathsf{KeyGen}(t) \\ \textbf{for } i \in [n]: \\ \quad (\mathsf{hint}_i, \mathsf{tag}_i) \xleftarrow{\$} \mathsf{Update}_{\mathsf{pk}}(\mathsf{hint}_{i-1}, \mathsf{tag}_{i-1}, x_i, \mathfrak{r}_i) \\ \quad \mathsf{ecom}_i \xleftarrow{\$} \mathsf{Convert}_{\mathsf{pk}}(\mathsf{hint}_i) \\ \quad \mathfrak{C}_i \leftarrow \mathfrak{Commit}(x_i; \mathfrak{r}_i) \\ \textbf{return } \mathsf{VfKeyGen}(\mathsf{pk}, \mathsf{hint}_0) = \mathsf{acc} \wedge \\ \qquad\qquad \textbf{for all } i \in [n]: \\ \qquad\qquad\quad \mathsf{VfHint}_{\mathsf{pk}}(\mathsf{hint}_i, \mathsf{tag}_i) = \mathsf{acc} \wedge \\ \qquad\qquad\quad \mathsf{VfHistory}_{\mathsf{pk}}(\{\mathsf{tag}_j, \mathfrak{C}_j\}_{j=1}^{i}) = \mathsf{acc} \wedge \\ \qquad\qquad\quad \mathsf{VfECommit}_{\mathsf{pk}}(\mathsf{ecom}_i, \mathsf{tag}_i) = \mathsf{acc} \wedge \\ \qquad\qquad\quad \mathsf{Explode}_{\mathsf{sk}}(\mathsf{ecom}_i) = P(t, \sum_{j=1}^{i} x_j) \end{array}\right] = 1$$

- *Update correctness: for all* $\mathsf{pk}, \mathsf{hint}_0$ *s.t.* $\mathsf{VfKeyGen}(\mathsf{pk}, \mathsf{hint}_0) = \mathsf{acc}$, *and all* $\mathsf{hint}_n, \{\mathsf{tag}_j, \mathfrak{C}_j\}_{i=1}^{n}$ *such that* $\mathsf{VfHint}_{\mathsf{pk}}(\mathsf{hint}_n, \mathsf{tag}_n) = \mathsf{acc}$ *and* $\mathsf{VfHistory}_{\mathsf{pk}}(\{\mathsf{tag}_j, \mathfrak{C}_j\}_{j=1}^{n}) = \mathsf{acc}$, *and for all* $x, \mathfrak{r}$*:*

$$\Pr\left[\begin{array}{l} (\mathsf{hint}_{n+1}, \mathsf{tag}_{n+1}) \xleftarrow{\$} \mathsf{Update}_{\mathsf{pk}}(\mathsf{hint}_n, \mathsf{tag}_n, x, \mathfrak{r}) \\ \mathsf{ecom}_{n+1} \xleftarrow{\$} \mathsf{Convert}_{\mathsf{pk}}(\mathsf{hint}_{n+1}) \\ \mathfrak{C}_{n+1} \leftarrow \mathfrak{Commit}(x; \mathfrak{r}) \\ \textbf{return } \mathsf{VfHint}_{\mathsf{pk}}(\mathsf{hint}_{n+1}, \mathsf{tag}_{n+1}) = \mathsf{acc} \wedge \\ \qquad\qquad \mathsf{VfHistory}_{\mathsf{pk}}(\{\mathsf{tag}_j, \mathfrak{C}_j\}_{j=1}^{n+1}) = \mathsf{acc} \wedge \\ \qquad\qquad \mathsf{VfECommit}_{\mathsf{pk}}(\mathsf{ecom}_{n+1}, \mathsf{tag}_{n+1}) = \mathsf{acc} \end{array}\right] = 1$$

In both statements the probability is taken over the random coins internally sampled by the randomized algorithms of ECS.

6.2.1 Security Properties

We say that a history and predicate escrows are valid if they verify under a verifying public key. Valid histories and escrows must satisfy two properties – history binding and soundness.

History binding enforces that for any valid history, its prefix must also be valid, and no alternative prefix can ever be valid. This means that after verifying a commitment history one can use the last tag as a commitment to the whole history and is guaranteed of the validity of each step in the history.

Definition 6.2.4 (History Binding)**.** *An ECS for* (BC, P) *is* history binding *if for all PPT $\mathcal{A}$, it holds that:*

$$\Pr\left[\begin{array}{l} \mathfrak{pp} \xleftarrow{\$} \mathfrak{Setup}(1^\lambda);\ (\mathsf{pp}, \cdot) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, \mathfrak{pp}) \\ (\mathsf{pk}, \mathsf{hint}_0, \{\{\mathsf{tag}_i^{(0)}, \mathfrak{C}_i^{(0)}\}_{i=1}^{\iota}\}_{b\in\{0,1\}}) \xleftarrow{\$} \mathcal{A}(\mathsf{pp}) \\ \textbf{return } \mathsf{VfKeyGen}(\mathsf{pk}, \mathsf{hint}_0) = \mathsf{acc} \wedge \\ \qquad\qquad \mathsf{VfHistory}_{\mathsf{pk}}(\{\mathsf{tag}_i^{(0)}, \mathfrak{C}_i^{(0)}\}_{i=1}^{\iota}) = \mathsf{acc} \wedge \\ \qquad\qquad \Big(\mathsf{VfHistory}_{\mathsf{pk}}(\{\mathsf{tag}_i^{(0)}, \mathfrak{C}_i^{(0)}\}_{i=1}^{\iota-1}) \neq \mathsf{acc} \vee \\ \qquad\qquad \mathsf{VfHistory}_{\mathsf{pk}}(\{\mathsf{tag}_i^{(1)}, \mathfrak{C}_i^{(1)}\}_{i=1}^{\iota}) = \mathsf{acc} \wedge \\ \qquad\qquad \mathsf{tag}_\iota^{(0)} = \mathsf{tag}_\iota^{(1)} \wedge \exists i.\, (\mathsf{tag}_i^{(0)}, \mathfrak{C}_i^{(0)}) \neq (\mathsf{tag}_i^{(1)}, \mathfrak{C}_i^{(1)})\Big) \end{array}\right] = \mathsf{negl}(\lambda)$$

In practice, history binding prevents history manipulation: assuming an updater that produced tag as a "receipt" of their update is later approached by the regulator for presenting their escrow ecom, the updater will not be able to deceive the regulator by saying "this tag tag I produced for a different history". So history binding is crucial for "tracking back" the history of changes done to the escrow ecom; it enforces history linearity.

Soundness focuses on what $\mathsf{VfECommit}$ and $\mathsf{VfHistory}$ functions mean *together*: (1) any verifying history "contains" a set of update values, and (2) if ecom verifies w.r.t. the last tag of this history, it must explode according to predicate P evaluated on the sum of its values.

Definition 6.2.5 (Soundness)**.** *An ECS for* (BC, P) *is* sound *if there exists a deterministic poly-time black-box extractor* Ext*, such that for all PPT $\mathcal{A}$:*

1. *Valid history can be explained in terms of commitments: for all* $\iota > 0$,

$$\Pr\left[\begin{array}{l} \mathfrak{pp} \xleftarrow{\$} \mathfrak{Setup}(1^\lambda) \\ (\mathsf{pp}, \mathsf{td}) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, \mathfrak{pp}) \\ (\mathsf{pk}, \mathsf{hint}_0, \{\mathsf{tag}_i, \mathfrak{C}_i\}_{i=1}^{\iota}) \leftarrow \mathcal{A}(\mathsf{pp}) \\ (x_\iota, \mathfrak{r}_\iota) \leftarrow \mathsf{Ext}(\mathsf{td}, \mathsf{tag}_\iota) \\ \textbf{return } \mathsf{VfKeyGen}(\mathsf{pk}, \mathsf{hint}_0) = \mathsf{acc} \wedge \\ \qquad\qquad \mathsf{VfHistory}_{\mathsf{pk}}(\{\mathsf{tag}_i, \mathfrak{C}_i\}_{i=1}^{\iota}) = \mathsf{acc} \wedge \\ \qquad\qquad \mathfrak{C}_\iota \neq \mathfrak{Commit}(x_\iota, \mathfrak{r}_\iota) \end{array}\right] = \mathsf{negl}(\lambda)$$

2. *Explosions are always w.r.t. the sum of update values: for all* $t \in \mathbb{V}$, $\iota > 0$,

$$\Pr\left[\begin{array}{l} \mathfrak{pp} \xleftarrow{\$} \mathfrak{Setup}(1^\lambda) \\ (\mathsf{pp}, \mathsf{td}) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, \mathfrak{pp}) \\ (\mathsf{sk}, \mathsf{pk}, \mathsf{hint}_0) \xleftarrow{\$} \mathsf{KeyGen}(t) \\ (\mathsf{ecom}^\star, \{\mathsf{tag}_i, \mathfrak{C}_i\}_{i=1}^{\iota}) \xleftarrow{\$} \mathcal{A}(\mathsf{pp}, \mathsf{pk}, \mathsf{hint}_0) \\ \text{for } i \in [1, \iota] : \\ \quad (x_i, \mathfrak{r}_i) \leftarrow \mathsf{Ext}(\mathsf{td}, \mathsf{tag}_i) \\ \textbf{return } \mathsf{VfHistory}_{\mathsf{pk}}(\{\mathsf{tag}_i, \mathfrak{C}_i\}_{i=1}^{\iota}) = \mathsf{acc} \wedge \\ \qquad\qquad \mathsf{VfECommit}_{\mathsf{pk}}(\mathsf{ecom}^\star, \mathsf{tag}_\iota) = \mathsf{acc} \wedge \\ \qquad\qquad \mathsf{Explode}_{\mathsf{sk}}(\mathsf{ecom}^\star) \neq P(t, \sum_{i=1}^{\iota} x_i) \end{array}\right] = \mathsf{negl}(\lambda)$$

The extractor is the same in both clauses of the definition, and works the same given the same inputs. This means that the two parts are composable: the extracted value in the second part satisfies $\forall i.\, \mathfrak{C}_i = \mathfrak{Commit}(x_i, \mathfrak{r}_i)$ with overwhelming probability. Together with the binding property of BC, this guarantees that any values x_ι that is opened, by revealing $\mathfrak{r}_\iota$ or that is used externally in proofs of knowledge about $\mathfrak{C}_\iota$, must be the same as that used to evaluate P.

The first part of soundness considers dishonest keys (emulating a view of a third party observing the history, e.g. on the bulletin board), while the second part has honest key because it is viewed from the honest regulator's perspective.

Our *hiding* definitions provide privacy guarantees, capturing the following properties: (1) output of KeyGen does not leak the threshold value t (Definition 6.2.6), (2) tags do not leak the update value (Definition 6.2.7), (3) hints do not leak the update value, without sk (Definition 6.2.8), and (4) exploding commitments only leak the result of the explosion predicate (Definition 6.2.9).

Threshold hiding states that it is computationally impossible to determine the threshold value t chosen upon key generation from public key, without the secret key.

Definition 6.2.6 (Threshold Hiding). *An ECS for* (BC, P) *is* threshold hiding *if for all PPT* $\mathcal{A}$ *it holds that:*

$$\Pr\left[\begin{array}{l} \mathfrak{pp} \xleftarrow{\$} \mathfrak{Setup}(1^\lambda);\ (\mathsf{pp}, \cdot) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, \mathfrak{pp}) \\ (t_0, t_1) \leftarrow \mathcal{A}(\mathsf{pp}),\ b \xleftarrow{\$} \{0,1\} \\ (\cdot, \mathsf{pk}, \mathsf{hint}_0) \xleftarrow{\$} \mathsf{KeyGen}(t_b) \\ b^\star \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{hint}_0) \\ \textbf{return } b \stackrel{?}{=} b^\star \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

Tag hiding states that tags do not reveal any additional information than already revealed by $\mathfrak{C}$ itself.

Definition 6.2.7 (Tag Hiding). *An ECS for* (BC, P) *is* hiding in tags *if, for* $(\mathsf{pp}, \mathsf{td}) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, \mathbb{V}, \mathcal{P})$ *all* $t \in \mathbb{V}$*, all* pk*, all pairs* $(\mathsf{hint}, \mathsf{tag})$ *such that* $\mathsf{VfHint}_{\mathsf{pk}}(\mathsf{hint}, \mathsf{tag}) = \mathsf{acc}$*, and for all* $x \in \mathbb{V}$*,* $\mathfrak{r} \in \mathbb{R}$*, there exists a PPT* $\mathcal{S}$ *such that:*

$$\left\{\mathsf{tag}' \mid (\cdot, \mathsf{tag}') \xleftarrow{\$} \mathsf{Update}_{\mathsf{pk}}(\mathsf{hint}, \mathsf{tag}, x, \mathfrak{r})\right\} \stackrel{p}{\approx} \left\{\mathcal{S}(\mathsf{td}, \mathsf{pk}, \mathsf{tag}, \mathfrak{C} := \mathfrak{Commit}(x, \mathfrak{r}))\right\}$$

where distributions are over the internal randomness of the $\mathtt{Update}$ *algorithm and the simulator. For the first update, this holds conditioned on* $\mathsf{hint} := \mathsf{hint}_0$*,* $\mathsf{tag} := \perp$*.*

Note that the simulation-style definition here is dictated by tags being verifiable w.r.t. base commitments in histories. This allows composable reasoning: tags are hiding regardless of the base commitments hiding property; whereas IND-style definition would imply that the base scheme needs to be hiding which we avoid. Tag hiding also implies hiding for any *sequence* of tags, and thus for any history $\{\mathsf{tag}_i, \mathfrak{C}_i\}_{i=1}^{\iota}$: using $\mathcal{S}$ and $\{\mathfrak{C}_i\}$ we can simulate all the tags one by one, without any hints.

Hint hiding states that without the secret key, hints do not leak information update values.

Definition 6.2.8 (Hint Hiding). *An ECS for* (BC, P) *is* (computationally value) hiding in hints *if, for all $t \in \mathbb{V}$ and all PPT $\mathcal{A}$, it holds that:*

$$\Pr\left[\begin{array}{l} \mathfrak{pp} \xleftarrow{\$} \mathfrak{Setup}(1^\lambda);\ (\mathsf{pp}, \cdot) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, \mathfrak{pp}) \\ (\cdot, \mathsf{pk}, \mathsf{hint}_0) \xleftarrow{\$} \mathsf{KeyGen}(t);\ b \xleftarrow{\$} \{0,1\} \\ (\mathsf{hint}^\star, \mathsf{tag}^\star, x^{(0)}, x^{(1)}, \mathfrak{r}) \xleftarrow{\$} \mathcal{A}(\mathsf{pp}, \mathsf{pk}, \mathsf{hint}_0) \\ (\mathsf{hint}, \cdot) \xleftarrow{\$} \mathsf{Update}(\mathsf{hint}^\star, \mathsf{tag}^\star, x^{(b)}, \mathfrak{r}) \\ b^\star \xleftarrow{\$} \mathcal{A}(\mathsf{hint}) \\ \textbf{return } b^\star \stackrel{?}{=} b \wedge \mathsf{VfCommit}(\mathsf{hint}^\star, \mathsf{tag}^\star) \stackrel{?}{=} \mathsf{acc} \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

Explosion hiding models that even with the knowledge of the secret key, the escrow ecom does not leak anything about the values inside the history besides the predicate result itself.

Definition 6.2.9 (Explosion Hiding)**.** *An ECS for* (BC, P) *is* explosion hiding *if there exists a PPT simulator $\mathcal{S}$ such that for all PPT $\mathcal{A}$, it holds that:*

$$\Pr\left[\begin{array}{l} (\mathsf{pp}, \mathsf{td}) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, \mathbb{V}, \mathcal{P}) \\ (t, \mathsf{pk}, \mathsf{hint}_0, \{\mathsf{tag}_i, x_i, \mathfrak{r}_i\}_{i=1}^{\iota}, \mathsf{hint}_\iota) \xleftarrow{\$} \mathcal{A}(\mathsf{pp}) \\ b \xleftarrow{\$} \{0,1\} \\ \mathsf{ecom} \leftarrow \textbf{if } b = 0 \textbf{ then } \mathsf{Convert}_{\mathsf{pk}}(\mathsf{hint}_\iota) \\ \qquad\qquad \textbf{else } \mathcal{S}(\mathsf{td}, \mathsf{pk}, P(t, \sum_{i\in[\iota]} x_i), \mathsf{tag}_\iota) \\ b^\star \xleftarrow{\$} \mathcal{A}(\mathsf{ecom}) \\ \textbf{return } b^\star = b \wedge \\ \qquad\qquad \mathsf{VfKeyGen}(\mathsf{pk}, \mathsf{hint}_0) = \mathsf{acc} \wedge \\ \qquad\qquad \mathsf{VfHistory}_{\mathsf{pk}}(\{\mathsf{tag}_i, \mathfrak{Commit}(x_i, \mathfrak{r}_i)\}_{i=1}^{\iota}) = \mathsf{acc} \wedge \\ \qquad\qquad \mathsf{VfHint}(\mathsf{hint}_\iota, \mathsf{tag}_\iota) = \mathsf{acc} \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

6.3 Efficient Realization of Exploding Commitments

Our construction ECS is presented in the Figures 6.3 (main algorithms), 6.2 (helper functions) and 6.4 (verification algorithms). The construction is instantiated with Pedersen commitment scheme PedersenBCS as a base commitment, and the predicate $P_d(T, X)$ that returns 1 if and only if x is in the range $\{t, t+1, \ldots, t+d-1\}$. It also uses the updatable proof system Π_u instantiated by CH20, and a straight-line simulation-extractable Π (instantiated by Fiat-Shamir transformed Σ-protocols for proofs of equality of discrete logarithm representations with witness encryption). Next, we proceed with an overview that gradually builds intuition on the techniques employed in our construction, and conclude with how to achieve privacy and soundness.

6.3.1 Basic Construction

6.3.1.0.1 Explosion Predicate We consider the predicate that returns 1 if the value x concealed in the exploding commitment is *above* the given threshold t. For efficiency, we limit the check to a reasonable interval, i.e. the commitment only explodes if the value x is in $[t, t+d-1]$ for a small value d: for generic X and T the predicate is defined as

$$P_d = P_d(T, X) = \left[\prod_{\delta=0}^{d-1} (X - T - \delta) \stackrel{?}{=} 0 \right] \in \{0, 1\}. \tag{6.1}$$

Clearly, when evaluated, the predicate $P_d(t, x)$ returns 1 if and only if x is in the "critical range" $[t, t+d-1]$ on which the core polynomial evaluates to 0. The polynomial is built in such a way to allow for efficient updates as discussed next.

6.3.1.0.2 Setup and Key Generation The setup takes as input a group $\mathbb{G}$ generated by $\mathfrak{Setup}$ of the base commitment scheme (Pedersen), together with generator $G_1 = G$ and $\mathfrak{H}$. It finishes the bilinear group setup, creating $\mathsf{pp}_{\mathsf{BLG}}$ w.r.t. $\mathbb{G}_1$. It also sets up common reference strings and trapdoors for the NIZK proofs (more on this in Section 6.3.2). Most importantly, it generates d random masking values $W_1, \ldots, W_d \stackrel{\$}{\leftarrow} \mathbb{G}$ which are needed for blinding ElGamal ciphertexts in the hints.

To run the key generation process, the regulator needs to choose a threshold value t. In a nutshell, KeyGen samples $\mathsf{sk} \stackrel{\$}{\leftarrow} \mathbb{Z}_q$ and computes its corresponding DH public key $H \leftarrow G^{\mathsf{sk}}$. The public key consists of H and some additional data to prove consistency. The key generation process additionally returns a hint consisting of: 1) a sequence of d ElGamal ciphertexts encrypting the powers of $t, t^2, \ldots, t^d$ under the public key H; 2) a NIZK consistency proof $\pi_{\mathsf{c},0}$ for these powers; and 3) additional dummy information for correct hint formatting. This allows public verifiability of the correctness of the key generation procedure and of the hints in epoch $\iota = 0$.

6.3.1.0.3 Hints and Updatability Hints are used to update the value concealed in an exploding commitment. In our construction, updates perform addition of a new known value x_ι to the already concealed one (which is possibly unknown).

At each epoch ι hints have two main components: 1) a sequence of d ElGamal ciphertexts in the exponent for a public key H (see Equation (6.5) for definition); and 2) a base Pedersen commitment $\mathfrak{X}$ for the accumulated value $\hat{x} = \sum_{i\in[\iota]} x_i$ embedded in the ciphertexts (the meaning of "embedded" will become clear in a moment).

To understand the mechanics behind updatability, we need to look back at the polynomial in our predicates P_d (see Equation (6.1)). This is a product of expressions consisting of a sum of a private/unknown value $(X - T)$ and a public value δ. Hence, the polynomial in Equation (6.1) can be written as a linear combination of powers of $(X - T)$, namely:

$$\prod_{\delta=0}^{d-1}((X-T)-\delta) = \sum_{i=0}^{d} U_i(X-T)^i \tag{6.2}$$

where all U_i are well-determined public coefficients that depend solely on i and d.[3] By the binomial theorem it is possible to rewrite terms on the right side of Equation (6.2) as:

$$(X-T+Y)^i = \sum_{j=0}^{i}\left(\binom{i}{j}Y^{i-j}\right)\cdot(X-T)^j. \tag{6.3}$$

Equation (6.3) shows that we can always build $((x+y)-t)^i$ linearly from (lower powers) $(x-t)^j$ and y. This property is exploited by the $\mathsf{UpdatePowers}$ helper function (Fig. 6.2) to compute hints for $x + y$ as a linear combination of the old hints values (dependent only on x and t), and values dependent only on the new (known) input $y = x_\iota$. For easy reference, we define the y-dependent values as

$$V_{i,j}(y) = \binom{i}{j}y^{i-j} \in \mathbb{Z}_q. \tag{6.4}$$

Recall that hints contain a sequence of ElGamal ciphertexts, in our construction the initial hint, produced at epoch $\iota = 0$ during key generation, contains encryptions of powers of $-t^i$, i.e.,

$$\{A_{0,i} = G^{r_{0,i}}, B_{0,i} = G^{(-t)^i}H^{r_{0,i}}\}_{i\in[d]},$$

[3] To be precise, the U_i are the Stirling coefficients, i.e., Stirling numbers of the first kind are defined as the coefficients in the expansion of the falling factorial polynomial $(x)_n = \prod_{i=0}^{n-1}(x-i) = \sum_{k=0}^{n} s(n,k)x^k$, and have closed form $s(n,k) = \left[\begin{array}{c} n \\ k \end{array}\right] = (-1)^{n-k}\cdot \sum_{1\leq i_1<\ldots<i_{n-k}\leq n-1} (\prod_{j=1}^{n-k} i_j)$ [Konvalina, 2000].

where the $\{r_{0,i}\}_{i\in[d]}$ are the random values and t it the regulator's secret threshold for explosion. Hints get progressively updated (as we show momentarily) into the following form:

$$\{A_{\iota,i} = G^{\hat{r}_{\iota,i}}, B_{\iota,i} = G^{(\hat{x}-t)^i} H^{\hat{r}_{\iota,i}}\}_{i\in[d]}, \tag{6.5}$$

where $\hat{r}_{0,i}$ denotes accumulated randomness, $\hat{x} = \sum_{i\in[\iota]} x_i$ is the committed value accumulated at epoch ι. (Note that when $\iota = 0$ the ciphertexts conceal the value $(0-t)^i$).

By linear homomorphism, it is possible to add a new known value y to the quantity $(x-t)$ that is concealed in the hints of the previous epoch via the expression:

$$B_{\iota+1,i} = \prod_{j=0}^{i} B_{\iota,j}^{V_{i,j}(y)} = G^{\sum_{j=0}^{i}(x-t)^j \cdot V_{i,j}(y)} \cdot (H^{r_{\iota,i}})^{V_{i,j}(y)} = G^{(x+y-t)^i} H^{r_{\iota,i} V_{i,j}(y)}$$

where the last equality comes for Equation (6.3) and the definition in Equation (6.4) and $B_{0,0} = G^{t^0} H^0$. Noting that $V_{i,0}(y) = y^i$, each $B_{\iota+1,i}$ can be computed solely from hints of epoch ι with $j > 0$ in the following way:

$$B_{\iota+1,i} = G^{y^i} \prod_{j=1}^{i} B_{\iota,j}^{V_{i,j}(y)}$$

where we isolate the $j = 0$ term $G^{y^i} = B_{\iota,0}^{V_{i,0}(y)}$ to the left.

6.3.1.0.4 Preparing Hints for Explosion This procedure is performed by the Convert algorithm (Fig. 6.3). Intuitively, the ElGamal ciphertexts are extracted from the hint, and "evaluated". The Evaluate algorithm raises both ciphertext components to $U_i \cdot \beta$, where β is a random non-zero value (used for masking non-explosion data), and the U_i are the Stirling coefficients described in Section 6.3.1.0.3. Specifically:

$$E_1 = \prod_{i\in[d]} (A_i^{U_i})^{\beta} = G^{\beta\cdot(\sum\limits_{i\in[d]} r_{\iota,i}\cdot U_i)}$$

$$E_2 = \prod_{i\in[d]} (B_i^{U_i})^{\beta} = \left(G^{\sum\limits_{i\in[d]} U_i (x-t)^i} H^{\sum\limits_{i\in[d]} r_{\iota,i}\cdot U_i}\right)^{\beta} = G^{\beta\cdot(\prod\limits_{\delta=0}^{d-1}(x-t-\delta))} H^{\beta\cdot(\sum\limits_{i\in[d]} r_{\iota,i}\cdot U_i)}$$

where the last equality comes for Equation (6.2). As a result, the holder of the ElGamal secret key cannot efficiently decrypt the evaluated ciphertext. Decryption corresponds to solving the discrete logarithm problem since β is random (and unknown to the authority), unless the ciphertext encrypt the value "0". Note that we built the predicate in such a way that the ciphertext encodes 0 only on the roots of the polynomial, which correspond to values in the "critical range". The Convert procedure outputs the predicate escrow which, in addition to the evaluated ElGamal ciphertext, contains additional components needed to prove consistency, and verify the correctness of the procedure.

6.3.1.0.5 Testing for Explosion This procedure is run by the regulator and simply attempts to decrypt the ciphertext (E_1, E_2) using the secret key sk corresponding to the ElGamal encryption public key H. This entails computing $M = E_2 \cdot (E_1)^{-\mathsf{sk}}$, which by construction is $M = G^{\beta \cdot (\prod_{\delta=0}^{d-1}(X-T-\delta))}$, where the reader should recognize the core polynomial of the predicate (see Equation (6.1)). Note that β (unknown to the regulator) acts as a random mask that prevents efficient decryption whenever the polynomial evaluates to a value other than 0. This makes M gibberish unless the predicate P_d evaluates to 1 (the polynomial evaluates to 0), which yields to $M = G^0 = 1_{\mathbb{G}}$.

6.3.1.0.6 Achieving Privacy Up to this point we discussed correctness of our construction. Now we focus on how to achieve privacy, i.e., the authority only learns $P_d(t, x)$ and nothing else, and updators learn nothing about the concealed value.

The IND-CPA property of ElGamal ciphertexts $\{A_i = G^{r_i}, B_i = G^{(x-t)^i} H^{r_i}\}_{i \in [d]}$ prevents updaters from seeing the concealed value. The regulator however can obtain $\{G^{(x-t)^i}\}_{i \in [d]}$ by decrypting the ciphertexts, and even though encoding values in the exponent makes generic decryption inefficient, it does not prevent the regulator from obtaining x by when the encoded values are in small, predictable ranges (which is the setting of our application). To hide x properly, updaters will *blind* hints before sending them to the regulator with the escrow.

The blinding is performed by BlindPowers and consists of multiplying each B_i component by a value W_i^α, where the $\{W_i\}_{i=1}^d$ are public group elements generated upon system setup, and α is a freshly sampled random value. Specifically, blinded ciphertexts are of the form: $\{A_i, D_i := B_i \cdot W_i^\alpha\}_{i \in [d]}$. To achieve efficient updatable proofs, the α component will be zero while the hints are updated, which means parties will exchange unblinded hints; and α will only be set while the hints are converted to an escrow (more details on this in the upcoming description of updatable proofs).

6.3.2 Achieving Soundness Using NIZKs

Intuitively speaking, soundness means then whenever the data (primarily hints and escrows) is valid, it must be "good" – bind to the history, contain only updates that are relevant to commitments, etc. As of now, hints and escrows can be malformed, and lacks these guarantees. We overcome these issues by employing NIZKs to ensure data correctness.

Our construction employs four kinds of proofs. The *key proof* (π_{pk}) will show that the public key was built correctly. The *consistency proof* (π_{c}) will show consistency of all the components in a hint, and it will be updatable (details of which are the main technical contribution of the construction). The *trace proof* (π_{t}) will show that the new hint — obtained updating a hint from the previous epoch — is computed correctly, and that the update value is the same as in the external commitment $\mathfrak{C}$. Trace proofs are included in tags tag and form the trace. The *escrow proof* (π_{e}) will show that the escrow ecom was produced correctly from a tag and its hint.

6.3.2.1 Key Proof (π_{pk})

During the key generation phase, the regulator produces the ElGamal encryptions of the powers of the threshold (as explained in Section 6.3.1.0.2), in particular, we will use the first ciphertext $(A_{0,1}, B_{0,1}) := (G^{r_{0,1}}, G^t H^{r_{0,1}})$ that is a standard ElGamal encryption of t for the auditor's public key H. In addition, the regulator computes a Pedersen commitment to the threshold $\mathfrak{T} = G^t \mathfrak{H}^{r_{\mathsf{t}}}$ (which is included in the public key pk). The public key additionally contains π_{pk} that proves knowledge of the sk corresponding to the ElGamal public key $H = G^{\mathsf{sk}}$, and knowledge of a threshold value t and randomnesses that realize the public components $B_{0,1}$ (contained in hint_0) and $\mathfrak{T}$. Specifically, the language $\mathcal{L}_{\mathsf{pk}}$ is defined by:

$$\begin{aligned} \mathsf{x} &= (H, B_{0,1}, \mathfrak{T}) \in \mathbb{G}^3 \\ \mathsf{w} &= \boxed{(\mathsf{sk}, t, r_{0,1}, r_{\mathsf{t}})} \in \mathbb{Z}_q^4 \end{aligned} \qquad \text{and} \qquad M(\mathsf{x}) = \begin{bmatrix} G & 0 & 0 & 0 \\ 0 & G & H & 0 \\ 0 & G & 0 & \mathfrak{H} \end{bmatrix}.$$

where witness here and in the following is highlighted in gray.

6.3.2.2 Consistency Proof (π_{c})

This proof shows that all the hint values (including the $\{A_{0,i}, B_{0,i}\}$ produced by KeyGen) are indeed encodings of $(\hat{x} - t)^i$, where $\hat{x}$ and t are the current accumulated value, and the original threshold selected by the regulator. It works for both unblinded (B_i) and blinded (D_i) hints. This proof is produced (1) originally by the regulator in KeyGen to prove the consistency of powers in $\mathsf{hint}_{\mathsf{pk}}$, and (2) by updating parties to prove that the hints they send further are still consistent, (3) by updating parties to prove to the regulator that the blinded hints in the escrow are consistent. The consistency proof will always refer to $\mathfrak{T}$ (the Pedersen commitment to the threshold included in pk) to make sure that the witness t used in all proof iterations is the same as the initial one.

At epoch ι, the consistency proof π_{c} proves the following statement: for an instance

$$\mathsf{x} = (H, \{A_{\iota,i}, D_{\iota,i}\}_{i\in[d]}, \mathfrak{T}, \mathfrak{X}_\iota, \mathfrak{A}) \in \mathbb{G}^{2d+4}$$

there exists a witness

$$\mathsf{w} = \left(\begin{array}{c} t, r_{\mathsf{t}}, \hat{x}_\iota, \hat{r}_{\mathsf{x},\iota}, \alpha, r_\alpha, \{\hat{r}_{\iota,i}\}_{i\in[d]}, \\ (\hat{x}_\iota - t), \{r_{\iota,i}(\hat{x}_\iota - t)\}_{i=1}^{d-1}, \alpha(\hat{x}_\iota - t), r_\alpha(\hat{x}_\iota - t) \end{array} \right) \in \mathbb{Z}_q^{2d+8}$$

such that the following relations are satisfied:

1. $\mathfrak{T} = G^{t}\mathfrak{H}^{r_{\mathsf{t}}}$ (r_{t} is the randomness used to create $\mathfrak{T}$, the Pedersen commitment to the threshold)
2. $\mathfrak{X}_\iota = G^{\hat{x}_\iota}\mathfrak{H}^{\hat{r}_{\mathsf{x},\iota}}$ (Pedersen commitment to $\hat{x}_\iota$, the accumulated value)
3. $\mathfrak{A}_\iota = G^{\alpha}\mathfrak{H}^{r_\alpha}$ (Pedersen commitment to the randomness for blinding factors)
4. $A_{\iota,1} = G^{\hat{r}_{\iota,1}}$ ($r_{\iota,i}$ is the randomness used to create the ElGamal ciphertext)
5. $D_{\iota,1} = G^{\hat{x}_\iota - t}H^{\hat{r}_{\iota,1}}W_1^{\alpha}$ (the blinded ciphertext encrypts $(\hat{x} - t)$)
6. $\forall\, i \in [2, d]$:
   (a) $A_{\iota,i} = G^{\hat{r}_{\iota,i}}$
   (b) $D_{\iota,i} = (D_{\iota,i-1})^{\hat{x}_\iota - t}(H^{-1})^{\hat{r}_{\iota,i-1}(\hat{x}_\iota - t)}H^{\hat{r}_{\iota,i}}(W_{i-1}^{-1})^{\alpha(\hat{x}_\iota - t)}W_i^{\alpha}$
7. *Witness products* (needed for step 6):
   (a) $1 = G^{\hat{x}_\iota}(G^{-1})^{t}(G^{-1})^{\hat{x}_\iota - t}$
   (b) $1 = \mathfrak{A}_\iota^{\hat{x}_\iota - t}(G^{-1})^{\alpha(\hat{x}_\iota - t)}(\mathfrak{H}^{-1})^{r_\alpha(\hat{x}_\iota - t)}$
   (c) $1 = A_{\iota,i}^{\hat{x}_\iota - t}(G^{-1})^{\hat{r}_{\iota,i}(\hat{x}_\iota - t)}$, for $i \in [d-1]$:

The complexity of this formula is due to the fact that we need to prove the relationship between the powers of $(\hat{x} - t)^i$, which we do recursively. Note that we do not store powers as additional witnesses; the only witness is the first power $(\hat{x}_\iota - t)$.

When simplified, the recursive formulas reduce to the following four relations:

1. $\mathfrak{T} = G^{t}\mathfrak{H}^{r_{\mathsf{t}}}$
2. $\mathfrak{X} = G^{\hat{x}}\mathfrak{H}^{\hat{r}_{\mathsf{x}}}$
3. $\mathfrak{A} = G^{\alpha}\mathfrak{H}^{r_{\alpha}}$
4. $(A_i, D_i) = (G^{\hat{r}_i}, G^{(\hat{x}-t)^i} H^{\hat{r}_i} W_i^{\alpha})$, for $i \in [d]$

As briefly mentioned before, we use α in two different ways depending on the scenario: (1) while updating the hints blinding is disabled: users will set $\alpha = r_{\alpha} = 0$, and thus $\mathfrak{A} = 1$; thus D_i will be actually just B_i; (2) while creating escrow, the blinding values α, r_{α} will be introduced, $\mathfrak{A} \neq 1$ will be sent to the regulator, but α, r_{α} will not, which will ensure hiding of the blinding approach.

Therefore, to verify the consistency proof, the party (user or regulator) needs an instance x, which consists of the original $D_{0,1}$ produced during key generation; a collection of ciphertexts $\{(A_{\iota,i}, D_{\iota,i})\}_{i\in[d]}$ (unblinded or blinded); a tag tag_{ι} containing a commitment $\mathfrak{X}$ to the accumulated $\hat{x}_{\iota}$; and a special commitment $\mathfrak{A}$ to the blinding randomness α (either trivial $\mathfrak{A} = 1$ for users, or nontrivial for regulator).

Consistency proofs are instantiated by Π_{u}, which is linear in the size of the hints, but is also updatable, meaning that the proof for new hints is a transformation of the previous consistency proof. Practically, this is quite efficient, since otherwise consistency proofs would need to be aggregated, and hints would thus grow in size; this is especially expensive given that the consistency language is linear in d. Because of updatability, no updating party (except for the regulator, who creates the initial proof) ever knows the whole witness “contained” in the proof.

6.3.2.2.1 Updating Hints and Consistency Proof. Th consistency proof language $\mathcal{L}_{\mathsf{c}}$ is structured in such a way, that it supports a transformation that we will call T_{upd}, which can change all the necessary witnesses, including our target aggregated commitment value $\hat{x}_{\iota}$. To fit within algebraic language updatability framework, we must be able to represent the new instance and witness as linear combination of the old instance and witness values correspondingly.

We first start with the instance, which implicitly defines $T_{\mathsf{xm}}, T_{\mathsf{xa}}$:

1. Using the (plaintext) update value x_{ι}, sample rerandomisation factors $r_{\iota,i}$, and compute the new hints:
   (a) $A_{\iota,i} = \left(\prod_{j=1}^{i}(A_{\iota-1,j})^{V_{i,j}(x_{\iota})}\right)G^{r_{\iota,i}}$.

(b) $B_{\iota,i} = G^{x_\iota^i}\left(\prod_{j=1}^{i}(B_{\iota-1,j})^{V_{i,j}(x_\iota)}\right)H^{r_{\iota,i}}$, where $G^{x_\iota^i}$ covers the role of implicit $(B_{\iota-1,0})^{V_{i,0}(x_\iota)}$.

2. Sample $r_{\mathsf{x},\iota}$, update the tag commitment $\mathfrak{X}_\iota = \mathfrak{X}_{\iota-1}G^{x_\iota}\mathfrak{H}^{r_{\mathsf{x},\iota}}$.
3. Sample α, r_α, create the special commitment $\mathfrak{A} = G^\alpha\mathfrak{H}^{r_\alpha}$ (optionally, or still assume $\mathfrak{A} = 1$).

Next we show how to update the witness, which implicitly defines $T_{\mathsf{wm}}, T_{\mathsf{wa}}$:

$$\begin{aligned}
\hat{x}_\iota &= \hat{x}_{\iota-1} + x_\iota \\
\hat{x}_\iota - t &= \hat{x}_{\iota-1} - t + x_\iota \\
\hat{r}_{\iota,i} &= \sum_{j=1}^{i} \hat{r}_{\iota-1,i} \cdot V_{i,j}(x_\iota) + r_{\iota,i} \\
\hat{r}_{\iota,i}(\hat{x}_\iota - t) &= \sum_{j=1}^{i} \hat{r}_{\iota-1,i}(\hat{x}_{\iota-1} - t) \cdot V_{i,j}(x_\iota) + \sum_{j=1}^{i} \hat{r}_{\iota-1,i} \cdot x_\iota \cdot V_{i,j}(x_\iota) + r_{\iota,i} \cdot (\hat{x}_{\iota-1} - t) + r_{\iota,i}x_\iota \\
\hat{r}_{\mathsf{x},\iota} &= \hat{r}_{\mathsf{x},\iota-1} + r_{\mathsf{x},\iota} \\
\hat{\alpha} &= \alpha \\
\hat{r}_\alpha &= r_\alpha \\
\hat{\alpha}(\hat{x}_\iota - t) &= \alpha \cdot (\hat{x}_{\iota-1} - t) + \alpha \cdot x_\iota \\
\hat{r}_\alpha(\hat{x}_\iota - t) &= r_\alpha \cdot (\hat{x}_{\iota-1} - t) + r_\alpha \cdot x_\iota
\end{aligned}$$

The language transformation T_{upd} is formally a set of matrices $(T_{\mathsf{xm}}, T_{\mathsf{xa}}, T_{\mathsf{wm}}, T_{\mathsf{wa}})$ as implicitly defined above, that is parameterised by a vector of update values $\mathsf{w}_{\mathsf{upd,c}} = (x_\iota, \{r_{\iota,i}\}_{i\in[d]}, r_{\mathsf{x},\iota}, \alpha, r_\alpha)$, where all the "product witnesses" can be defined in terms of this tuple.

Note that we do not describe the last four witnesses as "accumulatable" — if we try to update w with (α, r_α) more than once, $\hat{\alpha}$ will not be equal to the sum of previous α unlike e.g. $\hat{r}_{\mathsf{x},\iota}$. This is due to our setup: (1) we apply T_{upd} incrementally parameterised with $(x_\iota, \{r_{\iota,i}\}_{i\in[d]}, r_{\mathsf{x},\iota}, \alpha = 0, r_\alpha = 0)$ with *blinding turned off*; (2) and then, given $\alpha = r_\alpha = 0$ and $\mathfrak{A} = 1$, we can *introduce blinding*, applying T_{upd} parameterised with $\alpha, r_\alpha \neq 0$ only once. In fact, this separation is a result of a deeper limitation of Π_{u}, discussion of which we defer to Section 6.3.3:

Theorem 6.3.1 (Validity of T_{upd} (Informal))**.** *The transformation T_{upd} is valid w.r.t. $\mathcal{L}_{\mathsf{c}}$, and the Π_{u} proof system for $\mathcal{L}_{\mathsf{c}}$ satisfies update completeness and derivation privacy w.r.t. T_{upd} when applied according to the two distinct parametrisations de-*

scribed above.

Proof. In Section 6.1 we explained that Π_{u} is complete and secure (Theorem 6.1.1) w.r.t. blinding-compatible transformations (Definition 6.1.1). Validity of T_{upd} can be straightforwardly derived from the instance and witness transformation matrices presented above. In Lemma 6.3.2, that follows the current more high-level overview section, we give details on the blinding compatibility of the transformation, and show that there exist a transformation (a variant of T_{upd}) that is indeed blinding compatible. $\square$

6.3.2.3 Trace Proof (π_{t})

Trace proofs are small aggregatable proofs that allow parties to linearise their updates. At epoch ι, the party performing an update with local value x_ι will prove the following statement. For an instance $\mathsf{x} = (H, \mathfrak{X}_{\iota-1}, \mathfrak{X}_\iota, \mathfrak{C}_\iota, \pi_{\mathsf{t},\iota-1})$ (where $\mathfrak{C}_\iota = G^{x_\iota}\mathfrak{H}^{\mathfrak{r}}$), there exists a witness $\mathsf{w} = (x_\iota, r_{\mathsf{x},\iota}, \mathfrak{r}_\iota)$ such that:

1. $\mathfrak{X}_\iota = \mathfrak{X}_{\iota-1} \cdot G^{x_\iota}\mathfrak{H}^{r_{\mathsf{x},\iota}}$ (the new tag is computed the form the previous one, and the updating information is completely known to the updator).
2. $\mathfrak{C}_\iota = \mathfrak{Commit}(x_\iota, \mathfrak{r}_\iota)$(the updated value x_ι is the same as in $\mathfrak{C}_\iota$).

Note that the value $\pi_{\mathsf{t},\iota-1}$ is in the instance, and thus bound by the NIZK being a signature of knowledge, but it does not appear in any equations. In practice this translates with hashing the additional value when computing a Fiat-Shamir challenge, but not using it otherwise. This proof will be instantiated with a standard non-updatable Π Σ-protocol.

As a potential future-work extension of our scheme, one can consider parties including their signatures on these elements, to sign the update act, which can be used for extending updater accountability w.r.t. the regulator.

6.3.2.4 Escrow Proof (π_{e})

This proof is produced upon conversion of a hint into an escrow ecom. The ecom contains an ElGamal encryption $E = (E_1, E_2)$ of $\beta \cdot P_d(\hat{x}, t)$ for some masking value β (random), an escrow proof, a consistency proof (rerandomized, and with α introduced), and information needed to check the proofs: a commitment $\mathfrak{X}$ to the accumulated value, a commitment $\mathfrak{B}$ to the randomness β, a commitment $\mathfrak{A}$

$\mathsf{BlindPowers}(\{B_i\}_{i\in[d]}, \alpha)$:

return $\{D_i := B_i \cdot W_i^{\alpha}\}_{i\in[d]}$

$\mathsf{Evaluate}(\{A_i, B_i\}_{i\in[d]}, \beta)$:

Let $\{U_i\}_{i=1}^{d}$ be Stirling numbers as defined in **??** 6.3.1.0.3.

return $\left(\prod_{i\in[d]} (A_i^{U_i})^{\beta}, \prod_{i\in[d]} (B_i^{U_i})^{\beta}\right)$

$\mathsf{UpdatePowers}_{\mathsf{pk}}(\{A_{\iota-1,i}, B_{\iota-1,i}\}_{i\in[d]}, x_\iota, \{r_{\iota,i}\}_{i\in[d]})$:

Let $V_{i,j}(X) := \binom{i}{j} X^{i-j}$

for $i \in [d]$ **do**

$\quad A_{\iota,i} \leftarrow \left(\prod_{j=1}^{i} (A_{\iota-1,j})^{V_{i,j}(x_\iota)}\right) G^{r_{\iota,i}}$

$\quad B_{\iota,i} \leftarrow \left(G^{x_\iota^i} \prod_{j=1}^{i} (B_{\iota-1,j})^{V_{i,j}(x_\iota)}\right) H^{r_{\iota,i}}$

return $\{A_{\iota,i}, B_{\iota,i}\}_{i\in[d]}$

Figure 6.2: Helper functions for the main protocol. The values $\{W_i\}_{i\in[d]}$ are independent bases, being part of the public parameters, and $\{V_{i,j}\}, \{U_i\}_{i\in[d]}$ are public values as defined in **??** 6.3.1.0.3.

to the introduced accumulated blinding exponent α, and, most importantly, *blinded* ElGamal ciphertexts $\{A_i, D_i\}_{i\in[d]}$ (with α).

The escrow proof for $\mathcal{L}_{\mathsf{e}}$ proves the following statement. For an instance $\mathsf{x} = (H, E_1, E_2, \mathfrak{B}, \mathfrak{A}, \prod A_i^{U_i}, \prod D_i^{U_i})\}_{i=1}^{d})$, there exists a witness $\mathsf{w} = (\alpha, r_\alpha, \beta, r_\beta, \beta\alpha, r_\beta\alpha)$ such that the following conditions are satisfied:

1. $\mathfrak{A} = G^{\alpha}\mathfrak{H}^{r_\alpha}$
2. $\mathfrak{B} = G^{\beta}\mathfrak{H}^{r_\beta}$
3. $1 = \mathfrak{B}^{\alpha}(G^{-1})^{\beta\alpha}(\mathfrak{H}^{-1})^{r_\beta\alpha}$
4. $E_1 = \prod_i (A_i^{U_i})^{\beta}$
5. $E_2 = \prod_i (D_i^{U_i})^{\beta} \cdot \prod_i (W_i^{-U_i})^{\beta\alpha}$

The language is compact, so the proof π_{e} can be created from scratch, and since it is not necessary for it to be updatable performance-wise we can also use standard Π as a proof system.

$\mathsf{Setup}(1^\lambda, \mathfrak{pp})$:

% To ensure $\mathbb{G}_1$ is the same for Pedersen BCS and the pairing system

Parse $\mathfrak{pp}$ as $(\mathbb{G}_1, G, \mathfrak{H}, P_d)$

$\mathsf{pp}_{\mathsf{BLG}} \leftarrow \mathsf{BLG.Setup}(1^\lambda; \mathbb{G}_1, G)$

% Blinding factors for $\{D_i\}_{i=1}^{d}$

$\{W_i\}_{i\in[d]} \xleftarrow{\$} \mathbb{G}_1$ % d comes from $P_d \in \mathfrak{pp}$

$(\boldsymbol{\sigma}_\Pi, \mathsf{td}_\Pi) \leftarrow \Pi.\mathsf{Setup}(1^\lambda, \mathsf{pp}_{\mathsf{BLG}})$

$(\boldsymbol{\sigma}_{\Pi_u}, \mathsf{td}_{\Pi_u}) \leftarrow \Pi_u^{\mathcal{L}_c}.\mathsf{Setup}(1^\lambda, \mathsf{pp}_{\mathsf{BLG}})$

$\mathsf{pp} \leftarrow (\mathfrak{pp}, \mathsf{pp}_{\mathsf{BLG}}, \{W_i\}_{i\in[d]}, 0, P_d, \boldsymbol{\sigma}_\Pi, \boldsymbol{\sigma}_{\Pi_u})$

$\mathsf{td} \leftarrow (\mathsf{td}_\Pi, \mathsf{td}_{\Pi_u})$

return $(\mathsf{pp}, \mathsf{td})$

$\mathsf{KeyGen}(t)$:

$\mathsf{sk} \xleftarrow{\$} \mathbb{Z}_q$, $H \leftarrow G^{\mathsf{sk}}$

$\{r_{0,i}\}_{i=1}^{d}, r_{\mathsf{t}} \xleftarrow{\$} \mathbb{Z}_q$

for $i \in [d]$ **do**

  % ElGamal encryptions of t^i

  $A_{0,i} \leftarrow G^{r_{0,i}}$, $B_{0,i} \leftarrow G^{((-t)^i)} H^{r_{0,i}}$

$\mathfrak{T} \leftarrow \mathfrak{Commit}(t; r_{\mathsf{t}})$ % $\mathfrak{T} = G^t \mathfrak{H}^{r_{\mathsf{t}}}$

$\mathfrak{X}_0, \mathfrak{A}_0 \leftarrow \mathfrak{Commit}(0; 0)$ % $\mathfrak{X}_0 = \mathfrak{A}_0 = 1_{\mathbb{G}_1}$

$\mathsf{x_c} \leftarrow (H, \{A_{0,i}, B_{0,i}\}_{i\in[d]}, \mathfrak{T}, \mathfrak{X}_0, \mathfrak{A}_0)$

$\mathsf{w_c} \leftarrow \left(\begin{array}{c} t, r_{\mathsf{t}}, \{r_{0,i}\}_{i\in[d]}, \hat{x} := 0, \\ \hat{r}_{\mathsf{x}} := 0, \{r_{0,i} \cdot (0 - t)\}_{i\in[d]}, \\ \alpha := 0, r_\alpha := 0 \\ \alpha \cdot (\hat{x} - t) := 0, r_\alpha(\hat{x} - t) := 0 \end{array} \right)$

$\pi_{\mathsf{c}} \xleftarrow{\$} \Pi_u^{\mathcal{L}_c}.\mathsf{Prove}(\mathsf{x_c}, \mathsf{w_c})$

$\pi_{\mathsf{pk}} \xleftarrow{\$} \Pi^{\mathcal{L}_{\mathsf{pk}}}.\mathsf{Prove}((H, B_{0,1}, \mathfrak{T}), (\mathsf{sk}, t, r_{0,1}, r_{\mathsf{t}}))$

$\mathsf{pk} \leftarrow (H, \mathfrak{T}, \pi_{\mathsf{pk}})$

$\mathsf{hint}_0 \leftarrow (\{A_{0,i}, B_{0,i}\}_{i\in[d]}, \mathfrak{X}_0, \pi_{\mathsf{c}})$

return $(\mathsf{sk}, \mathsf{pk}, \mathsf{hint}_0)$

$\mathsf{Update}_{\mathsf{pk}}(\mathsf{hint}_{\iota-1}, \mathsf{tag}_{\iota-1}, x_\iota; \mathfrak{r})$:

Parse $\mathsf{tag}_{\iota-1}$ as $(\pi_{\mathsf{t},\iota-1}, \mathfrak{X}_{\iota-1})$

$r_{\mathsf{x},\iota} \xleftarrow{\$} \mathbb{Z}_q$

$\mathsf{hint}_\iota \leftarrow \mathsf{UpdateHint}(\mathsf{hint}_{\iota-1}, x_\iota, r_{\mathsf{x},\iota})$

$\mathfrak{C}_\iota \leftarrow \mathfrak{Commit}(x_\iota, \mathfrak{r})$

Parse $\mathsf{hint}_{\iota-1}$ as $(\cdot, \mathfrak{X}_{\iota-1}, \cdot)$

Parse hint_ι as $(\cdot, \mathfrak{X}_\iota, \cdot)$

$\mathsf{x_t} \leftarrow \left(\begin{array}{c} H, \mathfrak{X}_{\iota-1}, \mathfrak{X}_\iota, \\ \mathfrak{C}_\iota, \pi_{\mathsf{t},\iota-1} \end{array} \right)$

$\pi_{\mathsf{t},\iota} \xleftarrow{\$} \Pi^{\mathcal{L}_{\mathsf{t}}}.\mathsf{Prove}(\mathsf{x_t}, (x_\iota, r_{\mathsf{x},\iota}, \mathfrak{r}_\iota))$

$\mathsf{tag}_\iota \leftarrow (\pi_{\mathsf{t},\iota-1}, \mathfrak{X}_\iota)$

return $(\mathsf{hint}_\iota, \mathsf{tag}_\iota)$

$\mathsf{UpdateHint}_{\mathsf{pk}}(\mathsf{hint}_{\iota-1}, x_\iota, r_{\mathsf{x},\iota})$: (Helper)

Parse $\mathsf{hint}_{\iota-1}$ as $\left(\begin{array}{c} \{A_{\iota-1,i}, B_{\iota-1,i}\}_{i\in[d]}, \\ \mathfrak{X}_{\iota-1}, \pi_{c,\iota-1}, \end{array} \right)$

$\{r_{\iota,i}\}_{i\in[d]} \xleftarrow{\$} \mathbb{Z}_q$

% $r_{\mathsf{x},\iota}$ is the input to UpdateHint

$\mathfrak{X}_\iota \leftarrow \mathfrak{X}_{\iota-1} \cdot \mathfrak{Commit}(x_\iota; r_{\mathsf{x},\iota})$

% $= \mathfrak{Commit}(\sum_{i\in[\iota]} x_i; \sum r_{\mathsf{x},i})$

$\{A_{\iota,i}, B_{\iota,i}\}_{i\in[d]} \leftarrow$

  $\mathsf{UpdatePowers}(\{A_{\iota-1,i}, B_{\iota-1,i}\}_{i\in[d]},$

    $x_\iota, \{r_{\iota,i}\}_{i\in[d]})$

$\mathsf{x_c} \leftarrow (H, \{A_{\iota-1,i}, B_{\iota-1,i}\}_{i\in[d]}, \mathfrak{T}, \mathfrak{X}_{\iota-1}, \mathfrak{A} := 1)$

$\mathsf{w_{upd,c}} \leftarrow (x_\iota, \{r_{\iota,i}\}_{i\in[d]}, r_{\mathsf{x},\iota}, \alpha := 0, r_\alpha := 0)$

$\pi_{\mathsf{c},\iota} \xleftarrow{\$} \Pi_u^{\mathcal{L}_c}.\mathsf{Update}(\pi_{\mathsf{c},\iota-1}; \mathsf{x_c}, \vec{T}_{\mathsf{upd}}(\mathsf{w_{upd,c}}))$

$\mathsf{hint}_\iota \leftarrow (\{A_{\iota,i}, B_{\iota,i}\}_{i\in[d]}, \mathfrak{X}_\iota, \pi_{\mathsf{c},\iota})$

return hint_ι

$\mathsf{Convert}_{\mathsf{pk}}(\mathsf{hint}_\iota)$:

% Partially rerandomize the hint

$(\{A_i, B_i\}_{i\in[d]}, \mathfrak{X}, \pi_{\mathsf{c}}) \leftarrow \mathsf{UpdateHint}_{\mathsf{pk}}(\mathsf{hint}_\iota, 0, 0)$

$\alpha, \beta, r_\alpha, r_\beta \xleftarrow{\$} \mathbb{Z}_q^\star$

$\mathfrak{A} \leftarrow \mathfrak{Commit}(\alpha; r_\alpha)$

$\mathfrak{B} \leftarrow \mathfrak{Commit}(\beta; r_\beta)$

$\{D_i\}_{i\in[d]} \leftarrow \mathsf{BlindPowers}(\{B_i\}_{i\in[d]}, \alpha)$

$(E_1, E_2) \leftarrow \mathsf{Evaluate}(\{A_i, B_i\}_{i\in[d]}, \beta)$

$\mathsf{x_c} \leftarrow (H, \{A_i, B_i\}_{i\in[d]}, \mathfrak{T}, \mathfrak{X}, \mathfrak{A} := 1)$

$\mathsf{w_{upd,c}} \leftarrow (x_\iota := 0, \{r_{\iota,i} := 0\}_{i\in[d]}, r_{\mathsf{x},\iota} := 0, \alpha, r_\alpha)$

$\pi'_{\mathsf{c}} \xleftarrow{\$} \Pi_u^{\mathcal{L}_c}.\mathsf{Update}(\pi_{\mathsf{c}}; \mathsf{x_c}, T_{\mathsf{upd}}(\mathsf{w_{upd,c}}))$

$\mathsf{w_e} \leftarrow (\alpha, r_\alpha, \beta, r_\beta)$

$\pi_{\mathsf{e}} \leftarrow \Pi^{\mathcal{L}_{\mathsf{e}}}.\mathsf{Prove}\left(\left(\begin{array}{c} H, E_1, E_2, \mathfrak{B}, \\ \mathfrak{A}, \{A_i, D_i\}_{i\in[d]} \end{array} \right), \mathsf{w_e}\right)$

$\mathsf{ecom} \leftarrow \left(\begin{array}{c} E_1, E_2, \pi_{\mathsf{e}}, \pi'_{\mathsf{c}}, \mathfrak{X}, \\ \{A_i, D_i\}_{i\in[d]}, \mathfrak{A}, \mathfrak{B} \end{array} \right)$

return ecom

$\mathsf{Explode}_{\mathsf{sk}}(\mathsf{ecom})$:

Parse ecom as $(E_1, E_2, \cdot)$

$M \leftarrow \mathsf{Decrypt}_{\mathsf{sk}}(E_1, E_2)$

% $M = G^{\beta P(t, \hat{x})}$

return $[M \stackrel{?}{=} 1_{\mathbb{G}_1}]$

Figure 6.3: Our ECS protocol for the PedersenBCS $= (\mathfrak{Setup}, \mathfrak{Commit})$ and the predicate $P_d(T, X) = \left[\prod_{\delta=0}^{d-1}(X - T - \delta) \stackrel{?}{=} 0\right]$, and CH20 as NIZK proof system. Main Algorithms.

$\underline{\mathsf{VfKeyGen}(P_d, \mathsf{pk}, \mathsf{hint}_0)}$:

Parse pk as $(H, \mathfrak{T}, \pi_{\mathsf{pk}})$
Parse hint_0 as $\big(\{A_i, B_i\}_{i \in [d]}, \mathfrak{X}, \pi_{\mathsf{c}}\big)$
assert $\mathfrak{X} = 1_{\mathbb{G}}$
assert $\Pi^{\mathcal{L}_{\mathsf{t},0}}.\mathsf{Verify}(\pi_{\mathsf{pk}}; (H, B_1, \mathfrak{T}))$
assert $\Pi_{\mathsf{u}}^{\mathcal{L}_{\mathsf{c}}}.\mathsf{Verify}\left(\pi_{\mathsf{c}}; \begin{pmatrix} \{A_i, B_i\}_{i \in [d]}, \\ \mathfrak{T}, \mathfrak{X}, H, \mathfrak{A} := 1 \end{pmatrix}\right)$
return acc

$\underline{\mathsf{VfHistory}_{\mathsf{pk}}(\{\mathsf{tag}_i, \mathfrak{C}_i\}_{i=1}^{\iota})}$:

Set $\mathfrak{X}_0 \leftarrow 1_{\mathbb{G}}, \pi_{\mathsf{t},0} \leftarrow \pi_{\mathsf{pk}}$
Parse tag_ι as $(\pi_{\mathsf{t},\iota}, \mathfrak{X}_\iota)$ for all $i \in [\iota]$
for $i \in [\iota]$ **do**
    assert $\Pi^{\mathcal{L}_{\mathsf{t}}}.\mathsf{Verify}\left(\pi_{\mathsf{t},i}; \begin{pmatrix} H, \mathfrak{X}_{i-1}, \mathfrak{X}_i, \\ \mathfrak{C}_i, \pi_{\mathsf{t},i-1} \end{pmatrix}\right)$
return acc

$\underline{\mathsf{VfHint}_{\mathsf{pk}}(\mathsf{hint}, \mathsf{tag})}$:

Parse hint as $\big(\{A_i, B_i\}_{i \in [d]}, \mathfrak{X}, \pi_{\mathsf{c}}\big)$
Parse tag as $(\pi_{\mathsf{t}}, \mathfrak{X})$
return $\Pi_{\mathsf{u}}^{\mathcal{L}_{\mathsf{c}}}.\mathsf{Verify}\left(\pi_{\mathsf{c}}; \begin{pmatrix} H, \{A_i, B_i\}_{i \in [d]} \\ \mathfrak{T}, \mathfrak{X}, \mathfrak{A} := 1 \end{pmatrix}\right)$

$\underline{\mathsf{VfECommit}_{\mathsf{pk}}(\mathsf{ecom}, \mathsf{tag})}$:

Parse ecom as $\begin{pmatrix} E_1, E_2, \pi_{\mathsf{e}}, \pi_{\mathsf{c}}, \mathfrak{X} \\ \{A_i, D_i\}_{i \in [d]}, \mathfrak{A}, \mathfrak{B} \end{pmatrix}$
Parse tag as $(\cdot, \mathfrak{X}')$
assert $\mathfrak{X}' = \mathfrak{X}$
assert $\Pi_{\mathsf{u}}^{\mathcal{L}_{\mathsf{c}}}.\mathsf{Verify}\left(\pi_{\mathsf{c}}; \begin{pmatrix} H, \{A_i, D_i\}_{i \in [d]}, \\ \mathfrak{T}, \mathfrak{X}, \mathfrak{A} \end{pmatrix}\right)$
assert $\Pi^{\mathcal{L}_{\mathsf{e}}}.\mathsf{Verify}\left(\pi_{\mathsf{e}}; \begin{pmatrix} H, E_1, E_2, \mathfrak{B}, \\ \mathfrak{A}, \{A_i, D_i\}_{i \in [d]} \end{pmatrix}\right)$
return acc

Figure 6.4: Verification Algorithms for the ECS protocol. Continuation of Fig. 6.3.

6.3.3 Updatability for the Consistency Language

In Section 6.3.2.2 we presented the consistency language $\mathcal{L}_{\mathsf{c}}$ together with a general language transformation, that affects virtually all the witness elements simultaneously. In this section we provide technical details on Theorem 6.3.1 that claims that such transformation can indeed be used within Π_{u}.

While we describe transformation for $\mathcal{L}_{\mathsf{c}}$ in a unitary manner, in fact there exist *two distinct* blinding-compatible transformations for $\mathcal{L}_{\mathsf{c}}$. The first one, T_{upd}, will assume that the current witness has $\hat{\alpha} = \hat{r}_\alpha = 0$, and therefore $\mathfrak{A} = 1$ in the instance, and will update all the internal randomness values, including the commitment value $\hat{x}_\iota$. This transformation may be parameterised with either $\alpha = r_\alpha = 0$, in which case it can be applied to the transformed proof further; or with $\alpha, r_\alpha \neq 0$, in which case the new hints will become properly blinded, but the transformation will no longer apply. The second one, T_{blind} does not assume $\hat{\alpha} = \hat{r_\alpha} = 0$, can introduce non-zero (α, r_α) and other randomisers, except it does *not* update the value x_ι.

The intuition here is that although the consistency language is fully updatable, we can only update the proof (1) with x_ι, while (A_i, B_i) are not blinded, (2) with the blinder α, but then we cannot update x_ι further. In both cases all other randomisers are included and non-exclusive. The matrices $T_{\mathsf{am}}, T_{\mathsf{aa}}$ for both of these transformations look exactly the same structurally, except they will need to be parameterised with zero or non-zero update values depending on the case.

We will not further consider T_{blind} in this section because for our application it is only necessary to use T_{upd} in two modes: either with $\alpha = r_\alpha = 0$ when running Update, or with non-zero α, r_α when blinding within $\mathsf{Convert}$.

We will describe the matrices $T_{\mathsf{am}}, T_{\mathsf{aa}}$ in the form of a list for conciseness — we will present the linear transformations, per row, that characterize the mapping $(\mathsf{x}, S) \mapsto T_{\mathsf{am}} \cdot (S, \mathsf{x})^T + T_{\mathsf{aa}}$, where $S = M(\mathsf{x}) \cdot \boldsymbol{s}$. For clarity, unlike in Section 6.3, we will use $U_\cdot$ notation to denote update values explicitly, e.g. U_x is an update value corresponding to the variable x. In this syntax S_{A_i} corresponds to the line of $M(\mathsf{x}) \cdot \boldsymbol{s}$ that would produce A_i if there was w instead of $\boldsymbol{s}$. Note that S_{3+2d+i} for $i \in [d+1]$ are generally not $1_{\mathbb{G}}$, unlike the real counterparts in x. Given the update values $(U_x, U_{r_{\mathsf{x},i}}, \{U_{r_{\iota,i}}\}_{i=1}^{d}, U_\alpha, U_{r_\alpha})$, the update transformation is as follows:

1. $S'_{\mathfrak{T}} = S_{\mathfrak{T}}$.

2. $S'_{\mathfrak{X}} = S_{\mathfrak{X}} G^{U_x} \mathfrak{H}^{U_{r_\mathsf{x}}}$.
3. $S'_{\mathfrak{A}} = G^{U_\alpha} \mathfrak{H}^{U_{r_\alpha}}$.
4. $S'_{A_1} = S_{A_1} G^{U_{r_1}}$
5. $S'_{D_1} = S_{D_1} G^{U_x} H^{U_{r_1}} W_1^{U_\alpha}$
6. $S'_{A_i} = \Big(\prod_{j=1}^{i} (S_{A_j})^{V_{i,j}(U_x)} \Big) G^{U_{r_{\iota,i}}}$ for $i \in [1, d]$
7. $S'_{D_i} = G^{U_x{}^i_\iota} \Big(\prod_{j=1}^{i} S_{D_j}^{\binom{i-1}{j-1} U_x^{i-j}} \Big) \Big(\prod_{j=1}^{i-1} D_j^{\binom{i-1}{j} U_x^{i-j}} \Big) H^{U_{r_i}} W_i^{U_\alpha}$ for $i \in [1, d]$
8. $S'_{3+2d+1} = S_{3+2d+1}$
9. $S'_{3+2d+2} = 1$
10. $S'_{5+2d+i} = \Big(\prod_{j=1}^{i} (S_{5+2d+j})^{V_{i,j}(U_x)} \Big) \Big(\prod_{j=1}^{i} (A_j)^{\binom{i}{j} U_x^{i-j+1}} \Big) \Big(\prod_{j=1}^{i} (S_{A_j})^{-\binom{i}{j} U_x^{i-j+1}} \Big)$ for $i \in [1, d-1]$

Now, we need to show why $T_{\mathsf{am}}, T_{\mathsf{aa}}$ defined implicitly so satisfies the blinding-compatibility equation:

$$T_{\mathsf{am}} \cdot \begin{pmatrix} M(\mathsf{x}) \cdot \boldsymbol{s} \\ \mathsf{x} \end{pmatrix} + T_{\mathsf{aa}} = M(T_{\mathsf{xm}} \cdot \mathsf{x} + T_{\mathsf{xa}}) \cdot \Big(T_{\mathsf{wm}} \cdot \boldsymbol{s} + T_{\mathsf{wa}} \Big)$$

More precisely, we need to prove that the equation is satisfied for all $\mathsf{x} \in \mathcal{L}$, $\boldsymbol{s}$ such that $\mathsf{x}_{\mathfrak{X}} = 1$, and $\boldsymbol{s}_\alpha = \boldsymbol{s}_{r_\alpha} = 0$.

Lemma 6.3.2. *The transformation T_{upd} described above is blinding-compatible w.r.t. $(x, w) \in \mathcal{L}_{\mathsf{c}}$ for which $\alpha = r_\alpha = \boldsymbol{s}_\alpha = \boldsymbol{s}_{r_\alpha} = 0$.*

Proof. Assume $\boldsymbol{s}_\alpha = \boldsymbol{s}_{r_\alpha} = 0$. Since the $(T_{\mathsf{am}}, T_{\mathsf{aa}})$ (defined through the set of 10 clauses we just mentioned) determines the left hand side (LHS) of the blinding-compatibility equation, we will focus on showing that the RHS is equal to it.

Let us look at $\boldsymbol{s}' := (T_{\mathsf{wm}} \cdot \boldsymbol{s} + T_{\mathsf{wa}})$:

1. $s'_t = s_t$
2. $s'_{r_\mathsf{t}} = s_{r_\mathsf{t}}$
3. $s'_x = U_x + s_x$
4. $s'_{r_\mathsf{x}} = U_{r_\mathsf{x}} + s_{r_\mathsf{x}}$
5. $s'_\alpha = U_\alpha$
6. $s'_{r_\alpha} = U_{r_\alpha}$
7. $s'_{x-t} = U_x + s_{x-t}$
8. $s'_{\alpha(x-t)} = U_x \cdot U_\alpha + U_\alpha \cdot s_{x-t}$
9. $s'_{r_\alpha(x-t)} = U_{r_\alpha} \cdot U_x + U_{r_\alpha} \cdot s_{x-t}$
10. $s'_{r_i} = \Big(\sum_{j=1}^{i} \binom{i}{j} U_x^{i-j} s_{r_j} \Big) + U_{r_i}$, for $i \in [d]$.

11. $s'_{r_i(x-t)} = \left(\sum_{j=1}^{i} \binom{i}{j} U_x^{i-j+1} s_{r_j}\right) + \left(\sum_{j=1}^{i} \binom{i}{j} U_x^{i-j} s_{r_j(x-t)}\right) + U_{r_i} U_x + U_{r_i} \cdot s_{x-t}$, for $i \in [d-1]$.

Now let us look at $M(T_{\mathsf{xm}} \cdot \mathsf{x} + T_{\mathsf{xa}}) \cdot \left(T_{\mathsf{wm}} \cdot \boldsymbol{s} + T_{\mathsf{wa}}\right)$, which we expect to be equal to LHS defining $T_{\mathsf{am}}, T_{\mathsf{aa}}$.

1. "$\mathfrak{T}$": $G^{s_t} \mathfrak{H}^{s_{r_\mathsf{t}}}$
2. "$\mathfrak{X}$": $G^{U_x + s_x} \mathfrak{H}^{U_{r_\mathsf{x}} + s_{r_\mathsf{x}}}$
3. "$\mathfrak{A}$": $G^{U_\alpha} \mathfrak{H}^{U_{r\alpha}}$
4. "A_1": $G^{U_{r_1} + s_{r_1}}$
5. "D_1": $G^{U_x + s_{x-t}} H^{U_{r_1} + s_{r_1}} W_1^{U_\alpha}$
6. "A_i",$i \in [2,d]$: $G^{\left(\sum_{j=1}^{i} \binom{i}{j} U_x^{i-j} s_{r_j}\right) + U_{r_i}}$
7. "D_i",$i \in [2,d]$:
$$(D'_{i-1})^{U_x + s_{x-t}} \times (H^{-1})^{\left(\sum_{j=1}^{i-1} \binom{i-1}{j} U_x^{i-j} s_{r_j}\right) + \left(\sum_{j=1}^{i-1} \binom{i-1}{j} U_x^{i-j-1} s_{r_j(x-t)}\right) + U_{r_{i-1}} U_x + U_{r_{i-1}} \cdot s_{x-t}}$$
$$\times H^{\left(\sum_{j=1}^{i} \binom{i}{j} U_x^{i-j} s_{r_j}\right) + U_{r_i}}$$
$$\times (W_{i-1}^{-1})^{U_x \cdot U_\alpha + U_\alpha \cdot s_{x-t}} W_i^{U_\alpha}$$
8. "$3+2d+1$": $G^{s_x + U_x} (G^{-1})^{s_t} (G^{-1})^{s_{x-t} + U_x}$
9. "$3+2d+2$": $(G^{U_\alpha} \mathfrak{H}^{U_{r\alpha}})^{U_x + s_{x-t}} (G^{-1})^{U_x \cdot U_\alpha + U_\alpha \cdot s_{x-t}} (\mathfrak{H}^{-1})^{U_{r\alpha} \cdot U_x + U_{r\alpha} \cdot s_{x-t}}$
10. "$5+2d+i$", $i \in [d-1]$:
$$(A'_i)^{U_x + s_{x-t}} (G^{-1})^{\left(\sum_{j=1}^{i} \binom{i}{j} U_x^{i-j+1} s_{r_j}\right) + \left(\sum_{j=1}^{i} \binom{i}{j} U_x^{i-j} s_{r_j(x-t)}\right) + U_{r_i} U_x + U_{r_i} \cdot s_{x-t}}$$

Clause 1 is completely unchanged. In clauses 2,3,4,5,6,8 M does not change ($M(\mathsf{x})[i] = M(\mathsf{x}')[i]$), but the witness $\boldsymbol{s}'$ does. In the rest clauses 7,9,10 both M and the witness change.

It is easy to verify that RHS clauses 1-6 are equal to the LHS presented earlier. Let us examine the other clauses one by one.

6.3.3.0.1 Clause 8.

The LHS is $S'_{3+2d+1} = S_{3+2d+1}$, while the RHS is

$$G^{s_x + U_x} (G^{-1})^{s_t} (G^{-1})^{s_{x-t} + U_x}$$

equal to $G^{s_x} (G^{-1})^{s_t} (G^{-1})^{s_{x-t}}$, which is in turn exactly equal to S_{3+2d+1}, as expected.

6.3.3.0.2 Clause 9.

The LHS is $S_{3+2d+2} = 1$, while the RHS is

$$(G^{U_\alpha} \mathfrak{H}^{U_{r\alpha}})^{U_x + s_{x-t}} (G^{-1})^{U_x \cdot U_\alpha + U_\alpha \cdot s_{x-t}} (\mathfrak{H}^{-1})^{U_{r\alpha} \cdot U_x + U_{r\alpha} \cdot s_{x-t}}$$

which is also equal to 1 when all the exponent terms cancel.

6.3.3.0.3 Clause 10. The RHS is equal to:

$$
\begin{aligned}
&(A'_i)^{U_x+s_{x-t}}(G^{-1})^{\left(\sum_{j=1}^{i}\binom{i}{j}U_x^{i-j}(U_x s_{r_j}+s_{r_j(x-t)})\right)+U_{r_i}U_x+U_{r_i}\cdot s_{x-t}} \\
&= (G^{\sum_{j=1}^{i}\binom{i}{j}r_j U_x^{i-j}+U_{r_i}})^{U_x+s_{x-t}}(G^{-1})^{\left(\sum_{j=1}^{i}\binom{i}{j}U_x^{i-j}(U_x s_{r_j}+s_{r_j(x-t)})\right)+U_{r_i}U_x+U_{r_i}\cdot s_{x-t}} \\
&= G^{\left(\sum_{j=1}^{i}\binom{i}{j}U_x^{i-j}(r_j(U_x+s_{x-t})-U_x s_{r_j}-s_{r_j(x-t)})\right)}
\end{aligned}
$$

Now, consider the LHS:

$$
\begin{aligned}
&\Big(\prod_{j=1}^{i}(S_{5+2d+j})^{V_{i,j}(U_x)}\Big)\times\Big(\prod_{j=1}^{i}(A_j)^{\binom{i}{j}x_\iota^{i-j+1}}\Big)\times\Big(\prod_{j=1}^{i}(S_{A_j})^{-\binom{i}{j}x_\iota^{i-j+1}}\Big) \\
&= \Big(\prod_{j=1}^{i}(S_{5+2d+j})^{V_{i,j}(U_x)}\Big)\times\Big(\prod_{j=1}^{i}G^{\binom{i}{j}x_\iota^{i-j+1}(r_j-s_{r_j})}\Big)
\end{aligned}
$$

Recall that $S_{5+2d+j}=A_j^{s_{x-t}}G^{-s_{r_j(x-t)}}=G^{r_j s_{x-t}-s_{r_j(x-t)}}$, then

$$
\begin{aligned}
&= \Big(\prod_{j=1}^{i}(G^{r_j s_{x-t}-s_{r_j(x-t)}})^{\binom{i}{j}U_x^{i-j}}\Big)\times\Big(\prod_{j=1}^{i}G^{\binom{i}{j}x_\iota^{i-j+1}(r_j-s_{r_j})}\Big) \\
&= \Big(\prod_{j=1}^{i}G^{\binom{i}{j}U_x^{i-j}(r_j s_{x-t}-s_{r_j(x-t)}+U_x(r_j-s_{r_j}))}\Big)
\end{aligned}
$$

which is exactly equal to the RHS.

6.3.3.0.4 Clause 7. We start with the RHS:

$$
\begin{aligned}
(D'_{i-1})^{U_x+s_{x-t}} &\times (H^{-1})^{\left(\sum_{j=1}^{i-1}\binom{i-1}{j}U_x^{i-j-1}(U_x s_{r_j}+s_{r_j(x-t)}\right)+U_{r_{i-1}}U_x+U_{r_{i-1}}\cdot s_{x-t}} \\
&\times H^{\left(\sum_{j=1}^{i}\binom{i}{j}U_x^{i-j}s_{r_j}\right)+U_{r_i}} \\
&\times (W_{i-1}^{-1})^{U_x\cdot U_\alpha+U_\alpha\cdot s_{x-t}}W_i^{U_\alpha}
\end{aligned}
$$

Expanding the first term with D'_{i-1} into

$$
=(G^{(x+U_x-t)^{i-1}}H^{\sum_{j=1}^{i-1}\binom{i-1}{j}r_j U_x^{i-j-1}+U_{r_{i-1}}}W_{i-1}^{U_\alpha})^{U_x+s_{x-t}}\times\dots
$$

we immediately see that the W_{i-1} terms and $H^{U_{r_{i-1}}(U_x+s_{x-t})}$ terms cancel:

$$
\begin{aligned}
=(G^{(x+U_x-t)^{i-1}}H^{\sum_{j=1}^{i-1}\binom{i-1}{j}r_j U_x^{i-j-1}})^{U_x+s_{x-t}} &\times (H^{-1})^{\left(\sum_{j=1}^{i-1}\binom{i-1}{j}U_x^{i-j-1}(U_x s_{r_j}+s_{r_j(x-t)}\right)} \\
&\times H^{\left(\sum_{j=1}^{i}\binom{i}{j}U_x^{i-j}s_{r_j}\right)+U_{r_i}}\times W_i^{U_\alpha}
\end{aligned}
$$

Rearrange the terms, grouping them by base:

$$
=H^{\sum_{j=1}^{i-1}\binom{i-1}{j}U_x^{i-j-1}(r_j(U_x+s_{x-t})-U_x s_{r_j}-s_{r_j(x-t)})}\times H^{\sum_{j=1}^{i}\binom{i}{j}U_x^{i-j}s_{r_j}}
$$

$$\times G^{(x+U_x-t)^{i-1}(U_x+s_{x-t})} H^{U_{r_i}} W_i^{U_\alpha}$$

Given that $\binom{i}{j} = \binom{i-1}{j-1} + \binom{i-1}{j}$, we have $H^{\sum_{j=1}^{i} \binom{i}{j} U_x^{i-j} s_{r_j}} = H^{s_{r_i}} H^{\sum_{j=1}^{i-1} (\binom{i-1}{j-1} + \binom{i-1}{j}) U_x^{i-j} s_{r_j}}$. Plugging it in into the previous equation we arrive at

$$= H^{\sum_{j=1}^{i-1} \binom{i-1}{j} U_x^{i-j-1}(r_j(U_x+s_{x-t}) - s_{r_j(x-t)})} \times H^{\sum_{j=1}^{i} \binom{i-1}{j-1} U_x^{i-j} s_{r_j}}$$
$$\times G^{(x+U_x-t)^{i-1}(U_x+s_{x-t})} H^{U_{r_i}} W_i^{U_\alpha}$$
$$= H^{\sum_{j=1}^{i-1} \binom{i-1}{j} U_x^{i-j-1}(r_j(U_x+s_{x-t}) - s_{r_j(x-t)} + s_{r_{j+1}})} \times G^{(x+U_x-t)^{i-1}(U_x+s_{x-t})} H^{U_{r_i} + U_x^{i-1} s_{r_1}} W_i^{U_\alpha}$$

Now, we will switch to the LHS, and show that it is equal to this last reduced version of the RHS.

$$G^{U_x^i} \left(\prod_{j=1}^{i} S_{D_j}^{\binom{i-1}{j-1} U_x^{i-j}} \right) \left(\prod_{j=1}^{i-1} D_j^{\binom{i-1}{j} U_x^{i-j}} \right) H^{U_{r_i}} W_i^{U_\alpha}$$
$$= \left(G^{U_x^i} \cdot \prod_{j=1}^{i-1} D_j^{\binom{i-1}{j} U_x^{i-j-1} U_x} \right) \left(\prod_{j=1}^{i} S_{D_j}^{\binom{i-1}{j-1} U_x^{i-j}} \right) \times H^{U_{r_i}} W_i^{U_\alpha}$$

Expand product over D_j:

$$= \left(G^{U_x^i} \cdot \prod_{j=1}^{i-1} D_j^{\binom{i-1}{j} U_x^{i-j-1} U_x} \right) \left(\prod_{j=1}^{i} S_{D_j}^{\binom{i-1}{j-1} U_x^{i-j}} \right) \times H^{U_{r_i}} W_i^{U_\alpha}$$
$$= \left(G^{U_x^{i-1}} \prod_{j=1}^{i-1} (G^{(x-t)^j} H^{r_j})^{\binom{i-1}{j} U_x^{i-j-1}} \right)^{U_x} \left(\prod_{j=1}^{i} S_{D_j}^{\binom{i-1}{j-1} U_x^{i-j}} \right) \times H^{U_{r_i}} W_i^{U_\alpha}$$
$$= \left(G^{(x+U_x-t)^{i-1}} H^{\sum_{j=1}^{i-1} r_j \binom{i-1}{j} U_x^{i-j-1}} \right)^{U_x} \left(\prod_{j=1}^{i} S_{D_j}^{\binom{i-1}{j-1} U_x^{i-j}} \right) \times H^{U_{r_i}} W_i^{U_\alpha}$$

Recalling that $S_{D_1} = G^{s_{x-t}} H^{s_{r_1}}$ and $S_{D_j} = D_{j-1}^{s_{x-t}} (H^{-1})^{s_{r_{j-1}(x-t)}} H^{s_{r_j}}$, substitute:

$$= H^{\sum_{j=1}^{i-1} r_j \binom{i-1}{j} U_x^{i-j}} (G^{s_{x-t}} H^{s_{r_1}})^{U_x^{i-1}} \left(\prod_{j=2}^{i} (D_{j-1}^{s_{x-t}} H^{s_{r_j} - s_{r_{j-1}(x-t)}})^{\binom{i-1}{j-1} U_x^{i-j}} \right)$$
$$\times G^{U_x (x+U_x-t)^{i-1}} H^{U_{r_i}} W_i^{U_\alpha}$$
$$= H^{\sum_{j=1}^{i-1} r_j \binom{i-1}{j} U_x^{i-j}} (G^{s_{x-t}} H^{s_{r_1}})^{U_x^{i-1}} \left(\prod_{j=1}^{i-1} (D_j^{s_{x-t}} H^{s_{r_{j+1}} - s_{r_j(x-t)}})^{\binom{i-1}{j} U_x^{i-j-1}} \right)$$
$$\times G^{U_x (x+U_x-t)^{i-1}} H^{U_{r_i}} W_i^{U_\alpha}$$
$$= H^{\sum_{j=1}^{i-1} r_j \binom{i-1}{j} U_x^{i-j}} (G^{s_{x-t}} H^{s_{r_1}})^{U_x^{i-1}} \left(\prod_{j=1}^{i-1} ((G^{(x-t)^j s_{x-t}} H^{r_j s_{x-t} + s_{r_{j+1}} - s_{r_j(x-t)}})^{\binom{i-1}{j} U_x^{i-j-1}} \right)$$

$$\times\, G^{U_x(x+U_x-t)^{i-1}} H^{U_{r_i}} W_i^{U_\alpha}$$

$$= \left(\prod_{j=1}^{i-1} H^{\binom{i-1}{j} U_x^{i-j-1}(r_j U_x + r_j s_{x-t} + s_{r_{j+1}} - s_{r_j(x-t)})}\right) \times G^{(s_{x-t}+U_x)(x+U_x-t)^{i-1}} H^{U_{r_i}+s_{r_1}U_x^{i-1}} W_i^{U_\alpha}$$

It is easy to see now that LHS is equal to the RHS, so clause 7 holds. □

6.4 Security Proofs for ECS Construction

We show that our ECS protocol w.r.t. $(\mathsf{PedersenBCS}, P_d(T,X))$ introduced in Section 6.3 is secure according to the security definitions in Section 6.2 under: hiding and binding of PedersenBCS; DDH in $\mathbb{G}_1$ that in particular implies ElGamal IND-CPA; completeness, strong simulation-extractability, and ZK of Π; and (update) completeness, soundness, derivation privacy, and ZK of Π_u.

Theorem 6.4.1 (History Binding)**.** *The* ECS *protocol is a* history binding *ECS for* $(\mathsf{PedersenBCS}, P_d(T,X))$ *by strong simulation-extractability of* Π*.*

Proof. The first winning condition for the history binding game, that VfHistory must verify for any prefix, is trivially satisfied by the construction: our VfHistory checks every proof linearly, so if all proofs are valid, any prefix is also valid.

The second condition says that two histories are "merged": the last tags are the same, $\mathsf{tag}_\iota^{(0)} = \mathsf{tag}_\iota^{(1)}$, but the histories are different: $\exists i.\ (\mathsf{tag}_i^{(0)}, \mathfrak{C}_i^{(0)}) \neq (\mathsf{tag}_i^{(1)}, \mathfrak{C}_i^{(1)})$. Locate the greatest index j satisfying the second condition. This also implies that $(\mathsf{tag}_{j+1}^{(0)}, \mathfrak{C}_{j+1}^{(0)}) = (\mathsf{tag}_{j+1}^{(1)}, \mathfrak{C}_{j+1}^{(1)})$. We will refer to $\mathsf{tag}_{j+1}^{(0)} = \mathsf{tag}_{j+1}^{(1)}$ as just tag_{j+1}.

Because $\mathsf{tag}_j \leftarrow (\pi_{\mathsf{x},j}, \mathfrak{X}_j)$, our condition is equivalent to either $\pi_{\mathsf{x},j}$, $\mathfrak{X}_j$, or $\mathfrak{C}_j$ being different in the two cases. Note that both $\mathfrak{C}_j$ and $\mathfrak{X}_j$ are inputs to $\pi_{\mathsf{x},j}$; and that $\mathfrak{X}_j$ is an input to $\pi_{\mathsf{x},j+1}$ which is equal in both cases.

By strong simulation-extractability, proofs with different instances are distinct, so by SE of $\pi_{\mathsf{x},j+1}$ we get $\mathfrak{X}_j^{(0)} = \mathfrak{X}_j^{(1)}$ and $\pi_{\mathsf{x},j}^{(0)} = \pi_{\mathsf{x},j}^{(1)}$. Applying the same SE logic to $\pi_{\mathsf{x},j}$ now, we obtain $\mathfrak{C}_j^{(0)} = \mathfrak{C}_j^{(1)}$. This contradicts the second winning condition. □

Theorem 6.4.2 (Soundness)**.** *The* ECS *protocol is a* sound *ECS for* $(\mathsf{PedersenBCS}, P_d(T,X))$*, if* Π *is straightline-extractable knowledge sound,* Π_u *is sound, and binding of* PedersenBCS*.*

Proof. To prove soundness we must show existence of an extractor Ext that satisfies two probabilistic statements for any PPT $\mathcal{A}$. Our extractor Ext, when given $\mathsf{tag}_\iota = (\pi_{\mathsf{x},\iota}, \mathfrak{X}_\iota)$, will use td_Π produced by Setup to extract witness $(x_\iota, r_{\mathsf{x},\iota}, \mathfrak{r}_\iota)$ from $\pi_{\mathsf{x},\iota}$ such that $\mathfrak{C}_\iota = \mathsf{Commit}(x_\iota, \mathfrak{r}_\iota)$ and $\mathfrak{X}_\iota = \mathfrak{X}_{\iota-1} G^{x_\iota} \mathfrak{H}^{r_{\mathsf{x},\iota}}$.

The first part of the soundness proof is then a trivial application of knowledge-soundness of Π, since it guarantees the well-formedness of $\mathfrak{C}_\iota$ w.r.t. the extracted values. In the rest of the proof we will focus on the second statement. The second

part of the soundness proof uses the same extractor multiple times, using which we now obtain $\{x_i, r_{\mathsf{x},i}, \mathfrak{r}_i\}_{i=1}^{\iota}$ from the whole trace. In addition we rely on KS of Π for the escrow proof, and on Π_u soundness for the consistency proof.

Right after the extraction from $\mathfrak{X}_i = \mathfrak{X}_{i-1} G^{x_i} \mathfrak{H}^{r_{\mathsf{x},i}}$ and $\mathfrak{X}_0 = 1$ we conclude that $\mathfrak{X}_\iota = G^{\hat{x}_\iota} \mathfrak{H}^{\hat{r}_{\mathsf{x},\iota}}$ where $\hat{x}_\iota := \sum_{i=1}^{\iota} x_i$ and similarly $\hat{r}_{\mathsf{x},\iota} := \sum_{i=1}^{\iota} r_{\mathsf{x},\iota}$. Because $\mathfrak{T}$ is honestly constructed, we know $\mathfrak{T} = G^t \mathfrak{H}^{r_\mathsf{t}}$.

By soundness of the escrow proof π_e we know that $\exists \alpha, r_\alpha, \beta, r_\beta, \beta\alpha, r_\beta\alpha$ such that

$$\begin{aligned}
\mathfrak{A} &= G^\alpha \mathfrak{H}^{r_\alpha} \\
\mathfrak{B} &= G^\beta \mathfrak{H}^{r_\beta} \\
1 &= \mathfrak{B}^{\hat{\alpha}} (G^{-1})^{\beta\alpha} (\mathfrak{H}^{-1})^{r_\beta \alpha} \\
(E_1, E_2) &= \left(\prod_{i=1}^{d} (A_i^{U_i})^\beta, \prod_{i=1}^{d} (D_i^{U_i})^\beta \cdot \prod_{i=1}^{d} (W_i^{-U_i})^{\beta\alpha} \right)
\end{aligned}$$

The purpose of lines (2) and (3) is to merely introduce the product variables correctly (without revealing β): by binding of $\mathfrak{B}$ (which is binding of $\mathsf{PedersenBCS}$) we know that the witness standing for $\beta\alpha$ is actually equal to $\beta \cdot \alpha$, and same stands for $r_\beta\alpha$.

From the lines (1) and (4), simplifying, we now deduce that $\exists \alpha, r_\alpha, \beta$:

$$\begin{aligned}
\mathfrak{A} &= G^\alpha \mathfrak{H}^{r_\alpha} \\
(E_1, E_2) &= \left(\prod_i A_i^{U_i \beta}, \prod_i D_i^{U_i \beta} \cdot \prod_i (W_i^{-U_i})^{\beta \cdot \alpha} \right)
\end{aligned}$$

Let us now analyse the consistency proof π_c. It can be shown similarly that by binding of the commitments, soundness of π_c, and in particular set of equations in line 7 in the description of $\mathcal{L}_c$ in Section 6.3.2.2, that the product witness elements

$\alpha(\hat{x}_\iota - t)$ and $\hat{r}_{\iota,i}(\hat{x}_\iota - t)$ are product of, correspondingly, witness elements α with $(\hat{x}_\iota - t)$ and $\hat{r}_{\iota,i}$ with $(\hat{x}_\iota - t)$. In addition, we thus deduce that witness variable $(\hat{x}_\iota - t)$ is equal to the difference of other two witness variables $\hat{x}_\iota$ and t. With these simplifications in mind, and collapsing recursive relations in line 6.b of $\mathcal{L}_c$ description, we obtain that $\exists t, r_{\mathsf{t}}, \hat{x}_\iota, \hat{r}_{\mathsf{x},\iota}, \alpha, r_\alpha, \{\hat{r}_i\}_{i=1}^d$ such that

$$\begin{aligned}
\mathfrak{T} &= G^t \mathfrak{H}^{r_{\mathsf{t}}} \\
\mathfrak{X}_\iota &= G^{\hat{x}_\iota} \mathfrak{H}^{\hat{r}_{\mathsf{x},\iota}} \\
\mathfrak{A} &= G^\alpha \mathfrak{H}^{r_\alpha} \\
(A_i, D_i) &= (G^{\hat{r}_i}, G^{(\hat{x}_\iota - t)^i} H^{\hat{r}_i} W_i^\alpha) \qquad \text{for } i \in [d]
\end{aligned}$$

By binding of $\mathfrak{T}$ and all $\mathfrak{X}_\iota$, we know that the existentially introduced $t, r_{\mathsf{t}}, \hat{x}_\iota, \hat{r}_{\mathsf{x},\iota}$ are the same that we constructed from the output of an earlier extractor. Therefore they can be removed from the existential statement together with commitment-validity lines.

By binding of $\mathfrak{A}$ we know that the existentially introduced variables (α, r_α) are equal in both statements, and the statement $\mathfrak{A} = G^\alpha \mathfrak{H}^{r_\alpha}$ itself can be removed from both.

Combining two sets of equations for π_{c} and π_{e} we now have: $\exists \alpha, \beta, \{\hat{r}_i\}_{i=1}^d$:

$$\begin{aligned}
(A_i, D_i) &= (G^{\hat{r}_i}, G^{(\hat{x}_\iota - t)^i} H^{\hat{r}_i} W_i^\alpha) \qquad \text{for } i \in [d] \\
(E_1, E_2) &= \left(\prod_i A_i^{U_i \beta}, \prod_i D_i^{U_i \beta} \cdot \prod_i (W_i^{-U_i})^{\beta \cdot \alpha} \right)
\end{aligned}$$

Substituting first into the second we arrive at: $\exists \alpha, \beta, \{\hat{r}_i\}_{i=1}^d$:

$$\begin{aligned}
(E_1, E_2) &= \left(\prod_i G^{\hat{r}_i U_i \beta}, \prod_i (G^{(\hat{x}_\iota - t)^i} H^{\hat{r}_i} W_i^\alpha)^{U_i \beta} \cdot \prod_i (W_i^{-U_i})^{\beta \cdot \alpha} \right) \\
&= \left(\prod_i G^{\hat{r}_i U_i \beta}, \prod_i G^{(\hat{x}_\iota - t)^i U_i \beta} H^{\hat{r}_i U_i \beta} \right) \\
&= \left(G^{\beta \sum_i \hat{r}_i U_i}, G^{\beta \sum_i (\hat{x}_\iota - t)^i U_i} H^{\beta \sum_i \hat{r}_i U_i} \right) \\
&= \left(G^{\beta \sum_i \hat{r}_i U_i}, G^{\beta P(t, \hat{x}_\iota)} H^{\beta \sum_i \hat{r}_i U_i} \right)
\end{aligned}$$

By combining everything, we deduce that (E_1, E_2) is an encryption of $\beta P(t, \hat{x}_\iota := \sum_{i=1}^\iota x_i)$; by ElGamal completeness, this will be the explosion value. This last transition used soundness of $\pi_{\mathsf{c}}, \pi_{\mathsf{e}}$ and binding of the commitment scheme. $\square$

Theorem 6.4.3 (Threshold Hiding)**.** *The* ECS *protocol is a* threshold hiding *ECS for* $(\mathsf{PedersenBCS}, P_d(T, X))$*, under DDH in* $\mathbb{G}_1$*, and ZK of both* Π *and* Π_u*.*

Proof. First, recall that the honest $\mathsf{KeyGen}(t)$ outputs

$$\begin{aligned} \mathsf{pk} &= (H, \mathfrak{T}, \pi_{\mathsf{pk}}) \\ \mathsf{hint}_0 &= \big(\{A_{0,i}, D_{0,i}\}_{i \in [d]}, \mathfrak{X}_0 = 1_{\mathbb{G}}, \pi_{\mathsf{c}}\big) \end{aligned}$$

We proceed in the sequence of hybrid games.

In the first game transition, we will replace $(\mathsf{pp}, \cdot) \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$ by $(\mathsf{pp}, \mathsf{td}) \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$, and using td will simulate both π_{pk} and π_{c} by zero-knowledge of Π and Π_u. The transition is perfect.

In the second game, we will replace $\mathfrak{T}$ by a randomly sampled element. Since the honest $\mathfrak{T} = G^t \mathfrak{H}^{r_t}$ for uniformly random r sampled in KeyGen, $\mathfrak{T}$ is uniform in $\mathfrak{T}$, and thus can be replaced. This transition is also perfect.

At this stage the game directly reduces to the IND-CPA of ElGamal (m-message variant) w.r.t. the public key of the regulator, which is known to be implied by DDH in the underlying group. Assuming $\mathcal{A}$ breaks threshold hiding of ECS, we describe an attacker $\mathcal{B}$ interacting with the d-IND-CPA challenger $\mathcal{C}$. First, $\mathcal{B}$ obtains $H = G^{\mathsf{sk}}$ from $\mathcal{C}$, which is a public key of the regulator. Then, $\mathcal{B}$ gets two threshold values t_1, t_2 from $\mathcal{A}$, gives $\mathcal{C}$ threshold powers $\{t_1^i\}_{i=1}^d$ and $\{t_2^i\}_{i=1}^d$, obtains ciphertexts $\{A_{0,i}, B_{0,i}\}_{i=1}^d$ which are encryptions of those powers. Remembering that no hint blinding is involved at this step, $\mathcal{B}$ can assemble $(\mathsf{pk}, \mathsf{hint}_0)$ (with simulated proofs and randomly chosen $\mathfrak{T}$ as before) and give it to $\mathcal{A}$. Now, if $\mathcal{A}$ can return $b^\star$ that decides threshold secrecy, then $\mathcal{B}$ can pass this $b^\star$ to $\mathcal{C}$ and decide d-IND-CPA. □

Theorem 6.4.4 (Tag Hiding)**.** *The* ECS *protocol is a* tag hiding *ECS for* $(\mathsf{PedersenBCS}, P_d(T, X))$*, assuming ZK of the trace proof, and hiding of* $\mathsf{PedersenBCS}$*.*

Proof. To prove tag hiding, we must construct $\mathcal{S}(\mathsf{td}, \mathsf{pk}, \mathsf{tag}_{\iota-1}, \mathfrak{C}_\iota := \mathfrak{Commit}(x_\iota, \mathfrak{r}_\iota))$ that outputs tag distributed equally to the honest $\mathsf{Update}_{\mathsf{pk}}(\mathsf{hint}_{\iota-1}, x_\iota, \mathfrak{r}_\iota)$.

Our simulator $\mathcal{S}$ will return $\mathsf{tag}_\iota = (\pi_{\mathsf{x},\iota}, \mathfrak{X}_\iota)$, where $\mathfrak{X}_\iota$ is randomly sampled, and $\pi_{\mathsf{x},\iota}$ is fully simulated using td. Since the honest $\mathfrak{X}_\iota = \mathfrak{X}_{\iota-1} \cdot \mathfrak{Commit}(x_\iota; r_{\mathsf{x},\iota}) = \mathfrak{X}_{\iota-1} G^{x_\iota} \mathfrak{H}^{r_{\mathsf{x},\iota}}$ is perfectly hiding due to $r_{\mathsf{x},\iota}$ being sampled randomly, a randomly sampled $\mathfrak{X}_\iota \in \mathbb{G}$ is in distributed exactly the same. Because Π is perfect zero-knowledge, $\mathcal{S}$ with td can simulate $\pi_{\mathsf{x},\iota}$ for the instance $\mathsf{x} = (H, \mathfrak{X}_{\iota-1}, \mathfrak{X}_\iota, \mathfrak{C}_\iota, \pi_{\mathsf{x},\iota-1})$, where H (part of pk), $\mathsf{tag}_{\iota-1} = (\pi_{\mathsf{x},\iota-1}, \mathfrak{X}_{\iota-1})$, and $\mathfrak{C}_\iota$ are all provided as an input to $\mathcal{S}$. □

Theorem 6.4.5 (Hint Hiding). *The* ECS *protocol is a* hint hiding *ECS for* $(\mathsf{PedersenBCS}, P_d(T, X))$, *under DDH in* $\mathbb{G}_1$*, ZK of both* Π *and* Π_{u}*, and hiding of* PedersenBCS.

Proof. Recall the hint structure; as an output of Update we obtain:

$$\mathsf{hint}_\iota = \left(\{A_{\iota,i}, D_{\iota,i}\}_{i\in[d]}, \mathfrak{X}_\iota, \pi_{\mathsf{c},\iota}\right)$$

This proof is very similar to how threshold hiding is proven, since KeyGen also produces a hint.

We start from the hint hiding game. As a first step, we switch to simulated setup Setup, and simulate $\pi_{\mathsf{c},\iota}, \pi_{\mathsf{x},\iota}$. One difference with the threshold hiding proof is that while $\pi_{\mathsf{x},\iota}$ is produced by Prove, $\pi_{\mathsf{c},\iota}$ is produce as a result of update of $\pi_{\mathsf{c},\iota-1}$. However, we can still instead simulate $\pi_{\mathsf{c},\iota-1}$ for the new instance, because Π_{u} is derivation private (updated proof is distributed as a completely fresh one for the new instance) and standard zero-knowledge. This transition is perfect.

In the next game we sample $\mathfrak{X}_\iota$ uniformly at random, as in the threshold hiding proof. Since it is perfectly hiding, this transition is also perfect.

Now, we can directly embed d-IND-CPA of ElGamal into our game. This is very similar to the threshold hiding case, except we will request encryptions of $\{(x^{(0)})^i\}_{i=1}^d$ and $\{(x^{(1)})^i\}_{i=1}^d$ from d-IND-CPA challenger, and then homomorphically append them to $\{A_{\iota-1,i}, B_{\iota-1,i}\}_{i=1}^d$. It follows that if $\mathcal{A}$ wins this last variant of the hint hiding game, than the reduction we just described will break d-IND-CPA of ElGamal, which is known to hold under DDH in $\mathbb{G}_1$. □

Before we formulate and prove explosion hiding, we state simple lemmas explaining why our blinding technique with $D_i = B_i W_i^\alpha$ is hiding.

Lemma 6.4.6. *Let* $\mathbb{G}$ *be a finite group with prime order* q *and generator* G*. Then for any polynomial* d*, DDH in* $\mathbb{G}$ *is equivalent to following problem we call* d*-VDDH:*

$$(G^x, \{G^{y_i}, G^{xy_i}\}_{i=1}^d) \overset{c}{\approx} (G^x, \{G^{y_i}, G^{z_i}\}_{i=1}^d)$$

Proof. The basic idea is that under DDH we can always create the $d+1$ challenge tuple $(G^{y_{d+1}}, G^{xy_{d+1} \text{ or } z_{d+1}})$ as $((G^{y_d})^\beta, (G^{xy_d \text{ or } z_d})^\beta)$ for a uniformly sampled β. This proves $(d+1)$-VDDH $\implies$ d-VDDH; everything else is straightforward.

It is clear that 1-VDDH is exactly DDH. It is also straightforward that d-VDDH implies standard $(d-1)$-VDDH and thus DDH, since VDDH tuple contains DDH tuple as a subset (take (G^x, G^{y_1}, G^{xy_1})); so whenever $\mathcal{A}$ can break $(d-1)$-VDDH, $\mathcal{A}$ can break d-VDDH – we just pass $\mathcal{A}$ a subset of the bigger challenge. We will now focus on the DDH $\implies$ d-VDDH direction.

Consider 2-VDDH, a problem to distinguish

$$(G^x, G^{y_1}, G^{xy_1}, G^{y_2}, G^{xy_2}) \text{ and } (G^x, G^{y_1}, G^{z_1}, G^{y_2}, G^{z_2})$$

where z_1, z_2 are uniform in $\mathbb{Z}_p$. When we are given DDH challenge $(C_1, C_2, C_3) := (G^x, G^{y_1}, G^{xy_1} \text{ or } G^{z_1})$, we can always pick β, and pass $\mathcal{A}$ the following VDDH challenge: $(C_1, C_2, C_3, C_2^\beta, C_3^\beta)$. The fourth element of the tuple is $G^{y_1\beta}$, so call $y_2 := y_1\beta$; the last element of the tuple is either $G^{x(y_1\beta)} = G^{xy_2}$, or it is $G^{z_1\beta}$. This last $z_1\beta$ term is not quite a perfectly uniform G^{z_2}, but it is hard to tell because $G^{z_1\beta}$ can be viewed as a DH challenge itself.

Formally, let $\mathcal{G}_0$ be the original 2-VDDH game, where we generate all the challenges by ourselves. In the $\mathcal{G}_1$ the challenger $\mathcal{C}$ will generate fourth and fifth element of the tuple as in our reduction, instead of picking fresh (y_2, z_2). We claim that if adversary $\mathcal{B}$ can distinguish $\mathcal{G}_0$ from $\mathcal{G}_1$, we can break DDH. Let us build reduction $\mathcal{R}_\mathcal{B}$. Our DDH challenge be $(C_1, C_2, C_3) := (G^\beta, G^{y_1}, G^{\beta y_1} \text{ or } G^{y_2})$, where y_2 is assumed to be uniformly random. $\mathcal{R}$ embeds the challenge as follows: it samples x, z_1, and returns either $(G^x, C_2, C_2^x, C_3, C_3^x)$ (real) or $(G^x, C_2, G^{z_1}, C_3, C_1^{z_1})$ (random). If the DDH challenge is real, then $\mathcal{R}$ behaves identical to $\mathcal{G}_1$, returning either $(G^x, G^{y_1}, G^{y_1 x}, G^{\beta y_1}, G^{\beta y_1 x})$ (real) or $(G^x, G^{y_1}, G^{z_1}, G^{\beta y_1}, G^{\beta z_1})$ (random). If the DDH challenge is random, $\mathcal{R}$ is identical to $\mathcal{G}_0$, returning either $(G^x, G^{y_1}, G^{y_1 x}, G^{y_2}, G^{y_2 x})$ (real) or $(G^x, G^{y_1}, G^{z_1}, G^{y_2}, G^{\beta z_1})$ (random). In the latter case, $\beta z_1 \overset{\text{p}}{\approx} z_2$, because β is uniformly random and is used in the tuple only there, once. This proves that $\mathcal{R}_\mathcal{B}$ solves DDH with the same probability of success that $\mathcal{B}$ has in distinguishing $\mathcal{G}_0$ from $\mathcal{G}_1$.

Now, $\mathcal{G}_0 \overset{\text{c}}{\approx} \mathcal{G}_1$ under DDH, and $\mathcal{G}_0$ is essentially VDDH. This concludes the proof, because the probability of $\mathcal{A}$ to win $\mathcal{G}_1$ is itself negligible under DDH, since it allows for a direct reduction that constructs 2-VDDH tuple from the DDH tuple.

Now it is easy to prove $(d-1)$-VDDH $\implies$ d-VDDH. Consider the d case:

$$(G^x, \{G^{y_i}, G^{xy_i}\}_{i=1}^d) \overset{c}{\approx} (G^x, \{G^{y_i}, G^{z_i}\}_{i=1}^d)$$

The idea is the same: $\mathcal{G}_0$ is d-VDDH, but in $\mathcal{G}_1$ we return $((G^{y_{d-1}})^\beta, (G^{xy_{d-1} \text{ or } z_{d-1}})^\beta)$ as the last two tuple elements instead of honest version. By DDH with challenge $(C_1, C_2, C_3) := (G^\beta, G^{y_{d-1}}, G^{\beta y_{d-1}} \text{ or } G^{y_d})$ we argue that $\mathcal{G}_1 \overset{c}{\approx} \mathcal{G}_2$. This is done as before: the reduction needs to sample $x, z_{d-1}, \{(y_i, z_i)\}_{i=1}^{d-2}$, and simulate the rest of the challenge: it will return either $(G^x, \{G^{y_i}, G^{xy_i}\}_{i=1}^{d-2}, C_2, C_2^x, C_3, C_3^x)$ or $(G^x, \{G^{y_i}, G^{z_i}\}_{i=1}^{d-2}, C_2, G^{z_{d-1}}, C_3, C_1^{z_{d-1}})$. The argument proceeds as before.

Now, $\mathcal{G}_2$ depends on $y_1 \dots y_{d-1}$ and $z_1 \dots z_{d-1}$, without depending on y_d, z_d, so if $\mathcal{A}$ can solve $\mathcal{G}_2$, then $\mathcal{R}_\mathcal{A}$ can solve $(d-1)$-VDDH. This concludes the proof. □

Lemma 6.4.7. *Let $\mathbb{G}$ be a finite group with prime order q and generator G, and d poly-sized. If DDH holds in $\mathbb{G}$, then for all PPT $\mathcal{A}$ the following holds:*

$$\Pr\left[\begin{array}{l} \{W_i\}_{i=1}^d \xleftarrow{\$} \mathbb{G} \\ (H, \{x_i\}_{i=1}^d) \xleftarrow{\$} \mathcal{A}(1^\lambda, \{W_i\}_{i=1}^d) \\ \{z_i, r_i\}_{i=1}^d, \alpha \xleftarrow{\$} \mathbb{Z}_q \\ b \xleftarrow{\$} \{0,1\} \\ b^\star \leftarrow \mathcal{A}(\{A_i := G^{r_i}, \\ \qquad\qquad D_i := \textbf{if } b = 0 \textbf{ then } G^{x_i} H^{r_i} W_i^\alpha \textbf{ else } G^{z_i}\}_{i=1}^d) \\ \textbf{return } b^\star = b \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

Proof. What $\mathcal{A}$ gets is essentially encryptions of $G^{x_i} W_i^\alpha$, where x_i is either random or chosen, for a random α. The structure seems similar to Pedersen commitments, except we use same randomness α for every commitment, but with different bases W_i. We show that these exponents $x_i + w_i\alpha$ are computationally hiding x_i, assuming d-VDDH, proven secure in Lemma 6.4.6 under DDH.

Our d-VDDH challenge is $(C_0, C_1, \dots C_{2d}) := (G^\alpha, \{(G^{w_i}, G^{\alpha w_i \text{ or } z_i})\}_{i=1}^d)$. It is straightforwardly embedded into our game. First, set all $W_i := C_{2*i-1}$ — the distribution of these is still uniform, so $\mathcal{A}$ does not see the difference when it is first for the first time. After $\mathcal{A}$ responds, as before, generate $\{r_i\}_{i=1}^d \xleftarrow{\$} \mathbb{Z}_q$, and now set all $(A_i := G^{r_i}, D_i := G^{x_i} H^{r_i} C_{2*i})$. The distribution of D_i is as in the honest game too: if VDDH instance is real, $D_i = G^{x_i} H^{r_i} W_i^\alpha$; if VDDH instance is random, then $D_i = G^{x_i} H^{r_i} G^{z_i}$ which is uniform in $\mathbb{G}$, which is the same as G^{z_i} for z_i uniform in $\mathbb{Z}_q$. □

Theorem 6.4.8 (Explosion Hiding). *The* ECS *protocol is an* explosion hiding *ECS for* $(\mathsf{PedersenBCS}, P_d(T, X))$*, assuming DDH in* $\mathbb{G}_1$*, zero-knowledge of* Π, Π_{u}*, knowledge-soundness of* Π*, soundness of* Π_{u}*, and hiding and binding of* $\mathsf{PedersenBCS}$*.*

Proof. An exploding commitment has the following form:

$$\mathsf{ecom} = \big(E_1, E_2, \pi_{\mathsf{e}}, \pi_{\mathsf{c}}, \mathfrak{X}, \{A_i, D_i\}_{i\in[d]}, \mathfrak{B}, \mathfrak{A}\big)$$

In the explosion hiding game we have a number of conditions before we claim equality of distributions. Let us first present the simulator, and then argue, based on these conditions, why the distributions are indeed equal.

The simulator $\mathcal{S}(\mathsf{td}, \mathsf{pk}, v_P := P(t, \sum_{i\in[\iota]} x_i), \mathsf{tag}_\iota)$ will:

1. Sample $\beta, r_{\mathsf{E}} \xleftarrow{\$} \mathbb{Z}_q$, create fresh encryption $(E_1, E_2) = (G^{r_{\mathsf{E}}}, G^{\beta v_P} H^{r_{\mathsf{E}}})$ based on the predicate value v_P;
2. Sample $\{A_i, D_i\}_{i\in[d]}, \mathfrak{B}, \mathfrak{A} \xleftarrow{\$} \mathbb{G}$ all independently uniformly at random.
3. Set $\mathfrak{X} := \mathsf{tag}_\iota.\mathfrak{X}$.
4. Simulate $\pi_{\mathsf{e}}, \pi_{\mathsf{c}}$ for these elements using td.

To formally prove that $\Big\{\mathsf{Convert}_{\mathsf{pk}}(\mathsf{hint}_\iota)\Big\} \overset{\mathsf{c}}{\approx} \Big\{\mathcal{S}(\mathsf{td}, \mathsf{pk}, P(t, \sum_{i\in[\iota]} x_i), \mathsf{tag}_\iota)\Big\}$ we will proceed in the series of games, starting from $\mathcal{G}_0$, where $\mathcal{A}$ observes $\mathsf{Convert}_{\mathsf{pk}}(\mathsf{hint}_\iota)$, towards $\mathcal{S}$, implementing parts of the $\mathcal{S}$ step by step.

$\mathcal{G}_1$: Simulate both $\pi_{\mathsf{e}}, \pi_{\mathsf{c}}$ in the output of $\mathsf{Convert}$. Because both Π and Π_{u} are zero-knowledge, the simulation is perfect, so $\mathcal{G}_1 \overset{\mathsf{p}}{\approx} \mathcal{G}_2$.

$\mathcal{G}_2$: Sample $\mathfrak{B}, \mathfrak{A} \xleftarrow{\$} \mathbb{G}$ uniformly at random. These two are honest commitments with $\mathfrak{H}$ as a base, and fresh randomness r_α, r_β generated in $\mathsf{Convert}$. By perfect hiding of Pedersen commitments, $\mathcal{G}_2 \overset{\mathsf{p}}{\approx} \mathcal{G}_1$.

$\mathcal{G}_3$: In this game we simulate setup for Π and extract from all $\{\pi_{\mathsf{x},i}\}_{i=1}^{\iota}, \pi_{\mathsf{pk}}$, similarly to how it is done in the soundness proof. The extractor will return $(t', r_{\mathsf{t}}, \{x'_i, \mathfrak{r}'_i, r_{\mathsf{x},i}\}_{i=1}^{\iota})$. Additionally, we assert that $(t', \{x'_i, \mathfrak{r}'_i\}_{i=1}^{\iota})$ are the same as provided by $\mathcal{A}$; we will refer to these values $(t, \{x_i, \mathfrak{r}_i\}_{i=1}^{\iota})$ (without "primes") as before.

This assertion does not fail unless with negligible probability because commitments $\mathfrak{T}, \{\mathfrak{X}_i\}_{i=1}^{\iota}$ are binding. This transition is thus computational by knowledge-soundness of Π and DH in $\mathbb{G}_1$ (which guarantees Pedersen binding).

$\mathcal{G}_4$: In this game we replace (E_1, E_2) by a *fresh* encryption of βv_P instead; except for that, all other part of ecom are as before. This is in contrast with the previous game, where we still honestly call $(E_1, E_2) \leftarrow \mathsf{Evaluate}(\{A_i, B_i\}_{i=1}^d, \beta)$.

We start by establishing the form that hint_ι takes, using soundness of $\pi_{\mathsf{c},\iota}$ similarly to the proof of Theorem 6.4.2 (soundness). From the previous game we know $x_i, \{r_{\mathsf{x},i}\}_{i\in[d]}$ that each $\mathfrak{X}_i = \mathfrak{X}_{i-1} G^{x_i} \mathfrak{H}^{r_{\mathsf{x},i}}$, and $\mathfrak{X}_0 = 1_{\mathbb{G}}$. Inductively, this implies that $\mathfrak{X}_\iota = G^{\hat{x}_\iota} \mathfrak{H}^{\hat{r}_{\mathsf{x},\iota}}$, where $\hat{x}_\iota := \sum_{i=1}^\iota x_i$, $\hat{r}_{\mathsf{x},\iota} := \sum_{i=1}^\iota r_{\mathsf{x},i}$. By soundness of the input $\pi_{\mathsf{c},\iota}$ and binding of PedersenBCS (both of which are needed to assert that witness-products are formed correctly; similar to the soundness proof), we deduce that $\exists \{\hat{r}_{\iota,i}\}_{i=1}^d, \hat{x}'_\iota, \hat{r}'_{\mathsf{x},\iota}, t', r'_{\mathsf{t}}, \alpha', r'_\alpha$ such that for $i \in [d]$, $A_{\iota,i} = G^{\hat{r}_{\iota,i}}$, $B_{\iota,i} = G^{(\hat{x}'_\iota - t')^i} H^{\hat{r}'_{\iota,i}} W_i^{\alpha'}$, $\mathfrak{X}_\iota = G^{\hat{x}'_\iota} \mathfrak{H}^{\hat{r}'_{\mathsf{x},\iota}}$,$\mathfrak{T} = G^{t'} \mathfrak{H}^{r'_{\mathsf{t}}}$, $1 = \mathfrak{A} = G^{\alpha'} \mathfrak{H}^{r'_\alpha}$. First, since $\pi_{\mathsf{c},\iota}$ verifies w.r.t. $\mathfrak{A} = 1$, we must conclude by binding of PedersenBCS that $\alpha' = r'_\alpha = 0$, and thus $B_{\iota,i} = G^{(\hat{x}'_\iota - t')^i} H^{\hat{r}'_{\iota,i}}$ is not blinded. Also by binding of $(\mathfrak{T}, \{\mathfrak{X}_i\}_{i=1}^\iota)$, we know that $\hat{x}'_\iota$ guaranteed existentially by π_{c} is the same as the extracted $\hat{x}_\iota$; same applies to $t', r'_{\mathsf{t}}, \hat{r}'_{\mathsf{x},\iota}$, so we can remove the “primes”. Now we simply know that $\exists \{\hat{r}_{\iota,i}\}_{i=1}^d$ such that for $i \in [d]$, $A_{\iota,i} = G^{\hat{r}_{\iota,i}}$, $B_{\iota,i} = G^{(\hat{x}_\iota - t)^i} H^{\hat{r}_{\iota,i}}$.

Honest Evaluate will return $(E_1, E_2) = \left(\prod_{i\in[d]} (A_i G^{r''_{\iota,i}})^{U_i\beta}, \prod_{i\in[d]} (B_i H^{r''_{\iota,i}})^{U_i\beta} \right)$, where $r''_{\iota,i}$ are freshly sampled on the line 1 of Convert. Denote $\hat{r}''_{\iota,i} := \hat{r}_{\iota,i} + r''_{\iota,i}$. Then together with the previous existential statement in mind we conclude that $\exists \{\hat{r}_{\iota,i}\}_{i=1}^d$ such that

$$
\begin{aligned}
(E_1, E_2) &= \left(\prod_{i\in[d]} G^{(\hat{r}_{\iota,i} + r''_{\iota,i}) \cdot U_i\beta}, \prod_{i\in[d]} G^{(\hat{x}_\iota - t)^i \cdot U_i\beta} H^{(\hat{r}_{\iota,i} + r''_{\iota,i}) \cdot U_i\beta} \right) \\
&= \left(G^{\sum_{i=1}^d \hat{r}''_{\iota,i} \cdot U_i\beta}, G^{\beta P(t, \hat{x}_\iota)} H^{\sum_{i=1}^d \hat{r}''_{\iota,i} \cdot U_i\beta} \right)
\end{aligned}
$$

which is exactly the encryption of $\beta P(t, \hat{x}_\iota)$ with the uniformly random $\sum_{i=1}^d \hat{r}''_{\iota,i} \cdot U_i\beta$.

This means that our honestly-produced encryption of $v_P = \beta P(t, \hat{x}_\iota)$ will be distributed equally to freshly encrypted (E_1, E_2). The computational transition is by soundness of π_{c} and binding of PedersenBCS that implies binding of various $\mathfrak{H}$-based Pedersen commitments.

$\mathcal{G}_5$: Replace $\{A_i, D_i\}_{i\in[d]}$ with randomly sampled elements, instead of returning a blinded re-randomized version of $A_i, B_{i i\in[d]}$ with locally sampled α. If $\mathcal{G}_5 \not\approx^{c}_{\mathcal{B}} \mathcal{G}_4$, then we can build a d-VDDH attacker from $\mathcal{B}$ by embedding the $\{A_i, D_i\}_{i\in[d]}$ challenge directly into our game (and changing W_i in the setup on the first line of the game). Note that important aspect is that Convert fully rerandomises honest $\{A_i, B_i\}_{i\in[d]}$ at line 1 with $\{r''_{\iota,i}\}_{i\in[d]}$. Therefore, by Lemma 6.4.7, $\mathcal{G}_5 \overset{c}{\approx} \mathcal{G}_4$ under DDH in $\mathbb{G}_1$.

The last game is exactly equivalent to the output of $\mathcal{S}$ — note that the honest and simulated $\mathfrak{X}$ is exactly the same, so it does not appear in any game transitions. This concludes the proof. □

6.5 Instantiation and Performance

In this section we evaluate the complexity of our exploding commitments: giving suggestions for concrete instantiations, analysing concrete bottlenecks, and giving asymptotic comparison of our algorithms.

6.5.0.1 General Algorithmic Choices

We consider an instantiation of our exploding commitments in the "standard model", with a caveat that we apply the Fiat-Shamir heuristic for the non-updatable proof system Π. The updatable proof system Π_u, that is instantiated by the CH20 [Couteau and Hartmann, 2020] NIZK, is non-interactive by design.

We realize the base commitment BC using Pedersen's scheme [Pedersen, 1992] over $\mathbb{G}_1$ of a practical type III pairing friendly elliptic curve, where source groups $\mathbb{G}_1$ and $\mathbb{G}_2$ have no efficiently computable isomorphism. While we do not consider a specific curve, we remark that several curves could be used, such Barreto-Naehrig [Barreto and Naehrig, 2006] or BLS12-381 [Barreto et al., 2003]. While the computational security of the Barreto-Naehrig 256 bit curve (BN256) is generally considered to have less than 128 bits of computational security [Tibouchi, 2016] due to algorithmic advancements [Kim and Jeong, 2017], this curve is supported by native operations in Solidity on Ethereum[4] and can thus be evaluated with a cheap gas price, in contrast to any other pairing scheme whose operations would

[4] Specified in EIP-1108 (`https://eips.ethereum.org/EIPS/eip-1108`) and introduced in Istanbul fork (`https://eips.ethereum.org/EIPS/eip-1679`)

have to be manually implemented at a huge gas penalty. To give a rough estimate of curve performance, on AWS z1d.3xlarge a BN256 pairing can be computed in about 1ms and a curve multiplication 0.25/1.2ms (in $\mathbb{G}_1$ and $\mathbb{G}_2$ correspondingly); in BLS12-381 implementations timings are only slightly higher: a pairing costs about 1.3 ms and curve multiplication in 0.6/1.5ms (for $\mathbb{G}_1$ and $\mathbb{G}_2$ correspondingly) [gna, 2023, mir, 2023].

6.5.0.2 Consistency Proof

The consistency proof is constructed in KeyGen, updated in Update and Convert, and verified in VfKeyGen, VfHint and VfECommit. The consistency language uses witness of size $2d + 8$, and instance of size $2d + 4$, therefore being overall $O(d)$ in size; during updates, four of witness elements corresponding to zero values of $\alpha = r_\alpha = 0$ related elements, do not need to be communicated, but we will ignore this minor performance improvement and focus on the $O(d)$ terms instead. Our matrices — both $M(x)$ and update matrices $(T_{\mathsf{am}}, T_{\mathsf{wm}}, \ldots)$ — are quite sparse: $M(x)$ has $\approx 8d$ elements excluding constants ($\approx 6d$ if blinding is disabled), T_{am} has $\approx 3 \cdot d(d-1)$, and T_{wm} about $(3/2) \cdot d(d-1)$ elements. Performance of Π_{u} mostly depends on the number of elements in these matrices. The setup phase is $O(1)$ since we only need to generate a single $\mathbb{G}_2$ challenge element. In terms of space, the proof takes $|\mathsf{x}|\mathbb{G}_1 + |\mathsf{w}|\mathbb{G}_2 = 4d + 12$ elements. We estimate proving time to be $\approx 6d \cdot E_1 + 2d \cdot E_2$, where E_i stands for exponentiation in group $\mathbb{G}_i$ (here again we give constants for the asymptotically dominating term). Update time will be quadratic and dominated by the update matrices sizes: $\approx 3d(d-1) \cdot E_1 + (3/2)d(d-1) \cdot E_2$. We note however, that update time in Convert will be linear, because the update matrices will only introduce blinders α, r_α and randomize B_i without updating the commitment value $\hat{x}$ — thus the matrices will only contain linear number of non-zero elements. As for verification time, Π_{u} supports two approaches: naive and batched. With the naive approach, directly as described in [Couteau and Hartmann, 2020], the cost will be dominated by about $12dP$ (where P stands for pairings). However, using a well-known pairing batching optimisation, we can bring down the verification cost to $\approx (2d + 8) \cdot P + 10d \cdot E_1$. This is achieved by sampling random values ϕ_i for $i \in [l]$ on the verifier's side, and combining all the rows of the verification equation into a single one, where every row is taken with scalar multiple ϕ_i. Security of such a transformation is guaranteed by Schwartz-Zippel lemma as usual: if the final ϕ_i-linearly-combined equation holds, equations corresponding to the individual rows must hold unless with negligible probability due to ϕ_i being sampled randomly and independently. In practice, this is much more efficient, since not only pairings are much more expensive than multiplications, but also because we can use multi-scalar-multiplication optimisations to batch those further.

6.5.0.3 Non-Updatable Proofs

The key, trace, and escrow proofs are all created using standard Σ-protocol techniques, instantiated in $\mathbb{G}_1$. For all of them we assume Π to be FS-transformed, but also straightline-extractable, which we will assume is achieved by adding El-Gamal encryptions of corresponding witnesses (to a CRS-embedded public key). Without the extra encryptions, Π is quite similar in Π_u, except $\mathbb{G}_2$ exponentiations are now $\mathbb{F}_q$ multiplications, and $\mathbb{G}_1$ exponentiations are used instead of pairings; we will still achieve $(|\mathsf{x}| + |\mathsf{w}|)\mathbb{G}_1$ proof size, $\approx |M(x)|E_1 + |\mathsf{w}|E_\mathbb{F}$ proving time (where $|M(x)|$ stands for the number of non-zero items in the matrix and $E_\mathbb{F}$ are multiplications in $\mathbb{F}_q$), and $\approx |M(x)|$ multiplications in verification. For more details see [Maurer, 2009, Sec. 6.5]. Straightline extractability will add extra $|\mathsf{w}|$ elements into the witness (Paillier randomness), $2|\mathsf{w}|$ ciphertexts into the instance (Paillier ciphertexts), and extra $3|\mathsf{w}|$ exponentiations into $M(x)$ (each ciphertext is three exponentiations). The setup will only see the extra one Paillier modulus generated, which we will treat as a constant cost. In practice, such transformation will not change asymptotics, only increasing the constant factors by $\approx 2-3$, which is affordable because Π is generally efficient. The approach of [Couteau et al., 2021] can also be used for better asymptotic efficiency.

The key proof is very small and $O(1)$ for our calculations: it only contains 3 instance elements, 4 witness elements, and 5 exponentiations in $M(x)$ in total. It is created in KeyGen and verified in VfKeyGen.

Similarly, the trace proof is $O(1)$ with its 5 instance elements, 3 witness elements, and 5 exponentiations in total. The proof is created as part of Update and is verified in VfHistory.

The escrow proof is also compact, since even though it uses $\{A_i, D_i\}_{i=1}^d$, only their *products* $\prod A_i^{U_i}$ (similarly for D_i) enter $M(x)$, so these products can be treated like constants from the perspective of $\mathcal{L}_{\mathsf{e}}$. Therefore we have $|\mathsf{x}| = 5$, $|\mathsf{w}| = 6$, the proof is constant sized, and we will only treat its proving and verification time as $O(d)$ due to the need to construct the aforementioned $\prod A_i^{U_i}$ (which is efficient via MSM). The proof is constructed as part of $\mathsf{Convert}$ and verified of $\mathsf{VfECommit}$.

Table 6.1: Complexity of our ECS construction in terms of the number $\mathtt{exp}$ of group exponentiations (point multiplications) and the number $\mathtt{pair}$ of pairing operations. $\#\mathbb{G}$ indicates the number of new group elements returned as output of an algorithm. Constant d is defining the "explosion" range $[t, t+d-1]$. Value ι_{cur} stands for the current epoch (history length). The bottom "total" row represents the minimum complexity of ECS construction executed iteratively over ι updates.

Algorithm	pair	exp	$\#\mathbb{G}$
Setup	0	$O(1)$	$O(d^2)$
KeyGen	0	$O(d)$	$O(d)$
Update	0	$O(d^2)$	$O(d)$
Convert	0	$O(d)$	$O(d)$
Explode	0	$O(1)$	0
VfKeyGen	$O(d)$	$O(1)$	0
VfHint	$O(d)$	$O(1)$	0
VfHistory	0	$O(\iota_{\mathsf{cur}})$	0
VfECommit	$O(d)$	$O(d)$	0
Total (w. hints stored)	$O(\iota \cdot d)$	$O(\iota \cdot d^2)$	$O(d^2 + \iota \cdot d)$
Total (w/o. hints stored)	$O(\iota \cdot d)$	$O(\iota \cdot d^2)$	$O(d^2 + \iota)$

6.5.0.4 Asymptotic Summary

We summarize the asymptotic complexity of the different algorithms in Table 6.1. We assume that the public combinatorial values (binomial coefficients for $V_{i,j}$ and Stirling coefficients for U_i) are pre-computed, which requires total auxiliary storage of $2d^2$ elements (added into the cost for Setup). Given that most constants, as we discussed in the previous paragraphs, are quite low (for $\mathsf{UpdatePowers}$ it is 1, for updating the proofs it is ≈ 3), we roughly estimate that the conservative choice of d could be about 30-100, in order to achieve a latency in running Update of at most 1 second (given pairing timing of 0.25ms to 1ms). However, in practice many optimisations should be possible, such as parallelising the (row-independent) quadradic computations, and using efficient MSMs, so concrete performance could be much better. Nevertheless, the estimates clearly show that the system is practical, the NIZK overhead is comparably low, and given that hints are not aggregated due to updatability of Π_u, the space use is quite optimal w.r.t. what is absolutely necessary to produce an update ($\{A_i, B_i\}$).

6.6 Extensions and Applications

We now consider some possible extensions and use-cases of our exploding commitments and how we imagine they could be deployed.

6.6.1 External Proofs

The main scenario we describe sets d to a small value, which defines the range $[t, t+d-1]$, such that commitment "explodes" when the internal aggregated value $\hat{x}$ is within this range. For this example to fully work, an auxiliary mechanism *must* be employed to ensure that the commitment of the accumulated value, happening as part of Update, does *not* cause a overflow. In other words, we need an extra range proof on the update value x_ι, which we can conventiently integrate into the protocol which already exposes $\mathfrak{C}_i = G^{x_\iota}\mathfrak{H}^{\mathfrak{r}_i}$ for exactly this purpose.

There are several different approach to range proofs based on Pedersen commitments such as Bulletproofs [Bünz et al., 2018] or "adjusted" Pedersen commitments and square decomposition [Couteau et al., 2021]. These are efficient, with the latter only requiring a constant amount of exponentiations and group elements in the proof. However for our case there is a much simpler approach: committing to bits of x_ι, and using Π to prove that (1) $\mathfrak{C}$ contain bit-reconstructed values; and (2) commitments are actually to the bit values $\in \{0,1\}$. The latter can be done as follows: given $C = G^x H^r$ for H being chosen uniformly at random, note that condition $x \in \{0,1\}$ is equivalent to $(x-1)x = 0$, therefore it is enough to prove that $C^x = H^{r'}$ for some r'. This requires $O(\log(d))$ exponentiations and group elements, which is practically efficient for our choice of d.

With this in mind, every updater can only "adjust" the aggregated rating $\hat{x}$ by $x_\iota \in [0, d]$, which makes ECS a proper score aggregation system. It goes without saying that other more complicated predicates can be proven in a similar manner about

x_ι; commitment $\mathfrak{C}_i$ is used precisely for this kind of external intergration of ECS with other applications.

6.6.2 Extensions

Our scheme is presented for a limited class of predicates and features to keep our definitions and construction simple. However, it can be easily generalized to support arbitrary polynomial predicates and non-binary explosion values. Another interesting feature could be the distributed decryption of predicate escrows.

6.6.2.1 Arbitrary Polynomial Predicates

Our protocol targets the "range polynomial" $P_d(T, X)$, that is zero in $[t, t+d-1]$. Additional structure of P_d allows us to use a linear number of hints which is beneficial for the performance of our construction. However, the main idea of our protocol can be generalised to *any* polynomial predicate $P(T, X)$ — for this we will need use a *quadratic* number of hints, each encoding $\{x^i t^j\}_{i,j:i+j\leq d}$, which are updatable in a similar way our linear hints are. That is, assuming $P(T, X) = \sum_{i,j} C_{i,j} X^i T^j$, we can always construct the evaluation of the updated polynomial $P(T, X+Y) = \sum_{i,j} C_{i,j}(X+Y)^i T^j$ from the hints, if we can transform old hints $\{(x^i t^j)\}_{i,j}$ into the new ones $\{(x+y)^i t^j\}_{i,j}$. The latter is always possible since $(x+y)^i t^j = t^j \sum_{k=0}^{i} \binom{i}{k} y^{i-k} \times (t^j x^k)$ is a linear combination of the previous $(t^j x^k)$, which are known. Quadratic number of hints makes many algorithms much less efficient, and generally imposes much stricter upper bounds on d. It is easy to see however that in the "mixed" scenario, where $P(T, X)$ captures several disjoint ranges (e.g. $[t_1, t_1 + d_1]$ *and* $[t_2, t_2 + d_2]$), our construction works almost the same with two sets of linear hints, and thus the tradeoff is much softer. An example of using non-interval predicates could be designing P to encode a certain few excluded "exploding" points, for example representing a certain blacklist of public key hashes.

6.6.2.2 Non-binary Explosion Value

In some applications, *e.g.*, when users are anonymous, it is desirable for $\mathsf{Convert}_{\mathsf{pk}}$ to return not only a binary explosion value but also information about the user (*e.g.* their identity) to enable further investigations. The ECS scheme can be modified to realise this functionality. The core idea is that $\mathsf{Convert}_{\mathsf{pk}}$, given a piece of extra information y, instead of returning an encryption of $\beta P(x)$, will instead return $(\beta_1 P(x), \beta_2 P(x) + y)$ for random β_1, β_2, which in case $P(x) \neq 0$ produces two random points and in the case $P(x) = 0$ (*i.e.* if the commitment exploded) returns $(0, y)$. The extra information y then can be proven to be added correctly by integrating a pre-computed commitment $\mathfrak{C}_y$ to y (e.g. coming from an external identity scheme) into the escrow proof, which now will not only attest to the correctness of the evaluation w.r.t. (x, y) but also that y is coming from the designated commitment $\mathfrak{C}_y$.

Another simple but useful variation of this idea is to return y not based on $\mathfrak{C}_y$, but as a function of x. Consider two polynomials P_1, P_2, where polynomial P_1 is binary and defining "explosion predicate" as before, and $P_2(x) = y$ is defining what the result in case of explosion will be. Then $\mathsf{Convert}_{\mathsf{pk}}$ can return $(\beta_1 P_1(x), \beta_2 P_1(x) + P_2(x))$, revealing the non-binary y that depends on x. Since P_1 and P_2 are different predicates, they will have different set of hints; but assuming that $\deg P_2$ is low, the previous paragraph explained how we can extend the ECS scheme to support polynomial reconstructions for arbitrary P_2. Again, here the escrow proof must be modified to attest to the correct evaluation of the escrow.

6.6.2.3 Distributed Explosions

We observe that while we explicitly consider the KeyGen and $\mathsf{Explode}$ algorithms to be run by a single party, these steps could be distributed or thresholdized using standard techniques [Gennaro et al., 2007]. This would have the advantage of both separating the attack surface for the secret decryption key sk and of requiring explicit consensus of exploding/decryption by a quorum of different key share holders. That is, it would only make it possible for the auditors to discover exactly which update caused the explosion (by running $\mathsf{Explode}$ after each update) if sufficiently many of them agree that they should learn this.

6.6.3 Applications to Accountable Privacy Preserving Blockchains

We believe that exploding commitments can find multiple uses in ensuring accountability for privacy preserving blockchain applications beyond cryptocurrencies. The security properties of ECS and their flexibility as a building block in modular con-

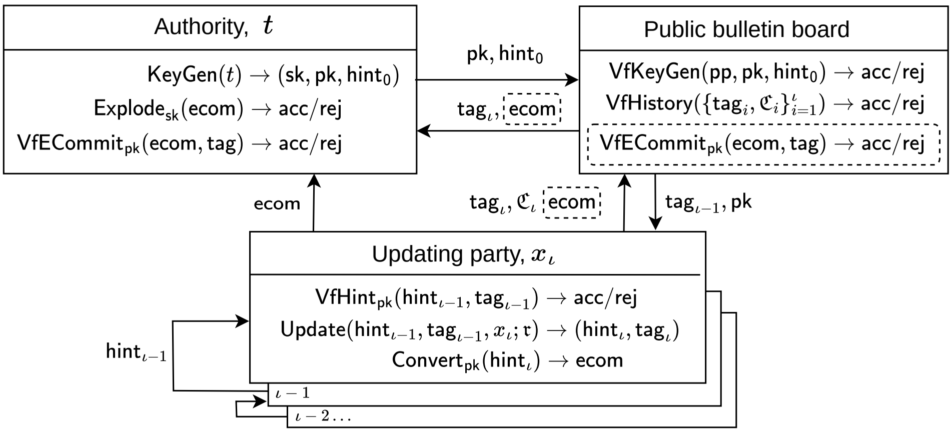

Figure 6.5: Illustration of a potential usage of exploding commitments in the blockchain setting. Dotted boxes illustrate optional communication and computation depending on the scenario.

structions makes them easy to integrate into various applications without undermining existing privacy properties.

In this setting, we assume access to a public append-only public ledger with support for a Turing complete scripting language (*e.g.* Ethereum or Cardano). We consider a model with an auditor who ensures setup of the system and who is able to decrypt the exploding commitment. With such a ledger, any auditor can run Setup and deploy a smart contract which embeds the verification functions: $\mathsf{VfKeyGen}, \mathsf{VfHint}, \mathsf{VfHistory}$ and $\mathsf{VfECommit}$. The auditor can then run KeyGen for a specific choice of exploding threshold t, and send the public key and hint_0 to the initial 0-commitment hint_0 to the smart contract, which validates these values using $\mathsf{VfKeyGen}$ and stores them if they are valid. External parties can now interact offchain, constructing updates by running the Update algorithm and sending the resulting hint to the next updating party, after validating the last hint using VfHint. When mandated by the application (potentially after every update), the updating party will publish its tag to the ledger, which will form a consistent history that will be validated by the smart contract using $\mathsf{VfHistory}$. When necessary, authority can request a party to $\mathsf{Convert}$ a certain hint corresponding to a point in history, obtain escrow ecom off-chain and check if the corresponding update has caused an explosion; this would also obviate any need for off-chain communication between authority and updating parties. Alternatively, ecom can be sent to the smart contract directly, which would be more expensive in terms of escrow validation, but can be used to e.g. allow authority to prove to a third party that a certain escrow exploded. We illustrate this in Fig. 6.5.

This approach is generic and allows for many application-specific variations. For example, it is possible the augment the smart contract logic to ensure that only a permissible set of parties are able to update the exploding commitments. Or, one could limit the quantity of updates any specific updating party is allowed to make; etc.

In case one wishes to simplify the smart contract and save gas, it is also possible to have the smart contract *only* store protocol elements but not run any of the verification algorithms, instead putting the responsibility for those on the auditor. Observe that each call to a smart contract is signed by the caller and stored on the ledger, linking it to the caller's public key. Thus, by augmenting the smart contract with Decentralized Identifiers [Sporny et al.,] it is possible to publicly audit which parties correctly follow the protocol.

6.6.4 Traditional AML

The exploding commitment scheme can be used in traditional AML by allowing banks to secretly communicate the user's suspiciousness score, or credit score, from the sending bank to the receiving bank. Concretely, by having the *sending* bank use the score of the *sending* account to compute a value with which the *recipient's* score should be increased. That is, an auditor generates keys (KeyGen) for an exploding commitment for each account in each bank. Then each bank checks that public keys using $\mathsf{VfKeyGen}$ and uses Update to add each account's *base* score to the corresponding hint_0 (initially containing 0). When an account holder makes a transaction, the *sending* bank updates the hint of the *receiving* account holder with an amount computed from the quantity of the transaction and the base-score of the sender. The receiving bank then runs VfHint and $\mathsf{VfHistory}$ to validate the update. At certain time-intervals each bank runs $\mathsf{Convert}$ to create the predicate escrow ecom to be shared with the auditor along with the tags that have been constructed as part of the updates since the last time the auditor did a check. The authority can then run $\mathsf{Explode}$ (along with $\mathsf{VfECommit}$) to check if an account should be flagged. We illustrate the overall flow of this in Fig. 6.6.

We observe that the scheme will allow the banks to get a much more accurate

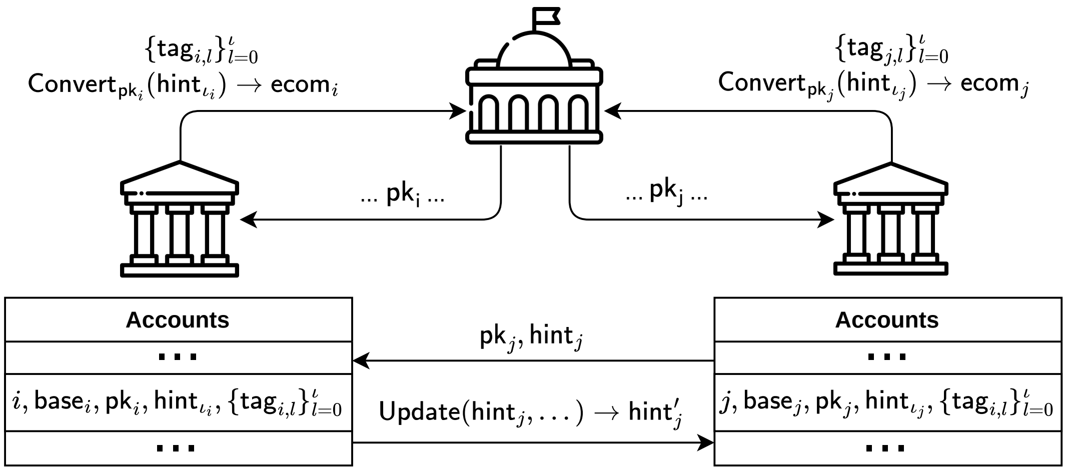

Figure 6.6: Illustration of the interaction between parties in the AML use-case.

suspiciousness score on each account without any bank leaking the base-score of their account holders. When manual inspection of an exploded account results in the bank needing to report the customer to the authorities, if it possible for the bank to also share all the hints used to compute the updates. This allows the authority to validate the entire history of transactions. Finally we also note that this can be enhanced using the idea of distributing the secret key in Sec. 6.6.2.3 to make the powers of the authority distributed.

6.6.5 Blockchain AML

The previously discussed approach to privacy preserving AML based on exploding commitments can be generalized to the blockchain setting. Most decentralized cryptocurrencies (*e.g.* Bitcoin, Ethereum and Cardano) allow anyone to perform transactions with no privacy guarantees, publicly revealing the transaction graph and transferred amounts. While this seemingly makes AML easy, it is cheap and easy to create new accounts, it is easy to perform layering though many accounts and various "mixing services", mixing tokens from many different sources [Pertsev et al., 2019]. Hence, although the transaction graph is public, AML is not simple. This issue is exacerbated by privacy preserving cryptocurrencies such as Zether [Bünz et al., 2020a], which is built on top of a smart contract system, or ZCash [Ben-Sasson et al., 2014a], Monero, and Dash, which incorporate privacy in their basic design. These schemes aim at hiding all transaction data, making AML even harder.

In order to reach a compromise between privacy and AML, several different idea have been proposed by authors in the recent years. Some authors [Sander and Ta-Shma, 1999, Barki and Gouget, 2020, Damgård et al., 2021] suggest an escrow system where anonymity and privacy can be broken if suspicious or illegal activities occur. Another approach is to specify a small spending limit per client which they can use every month for anonymous payments. After the client has made more transactions than covered by this budget, any future transactions can be traced [Wüst et al., 2019, Tomescu et al., 2022].

We now discuss how exploding commitments could be used to achieve private spending limits on ledgers with Turing complete smart contracts supporting private transactions.

6.6.5.1 Private Spending Limit

In this setting, we assume that the underlying ledger supports private transactions and the goal of an authority is to be able to find out which (if any) users go over their private spending limit. This is achieved by having a smart contract store an exploding commitment hint associated with each user and validate that each private transaction is supported by a Update to the user's exploding commitment history consistent with the transaction amount, followed by a $\mathsf{Convert}$. If an update causes the commitment to explode, it leaks the user's real identity to the auditor via the extension discussed in Sec. 6.6.2 that allows for explosions to output specific messages.

With this in mind we describe the use-case step by step: An authority starts by setting up new public and private keys and an initial hint_0 using KeyGen for an exploding commitments scheme. Public key pk and hint_0 are going to be "shared" by *all* users with a private spending limit. Next we require each user to get their real-world identity y validated by the auditor and linked in a privacy preserving manner to their smart contract. This is done by having the user post a commitment $Y = G^y H^{r_y}$ to a Merkle tree in the smart contract and prove to the auditor that the committed message is indeed their identity y. Each user gets a copy of hint_0 which they will store, and use for their first Update. For each private transaction the user runs Update and $\mathsf{Convert}$ to make a new hint, tag and ecom which will open to y in case of an explosion. The latter is based on the extension of Sec. 6.6.2. They then construct a zero-knowledge proof that the value y that ecom will open to in case of an explosion corresponds to the value of a leaf commitment on a path in the smart contract's Merkle tree (without revealing the path). Finally the user also constructs a proof of equality between the value in the base commitment of the update and the value of the private transaction they wish to carry out. The smart contract now validates the proofs, the private transaction, and whether the updates are valid ($\mathsf{VfHistory}$, $\mathsf{VfECommit}$).

Chapter 7

Conclusion

This work unites several closely connected topics, best united by their reliance on or analysis of malleability, primarily in NIZKs. The overarching sentiment of the paper is that subtleness in malleability definitions and analysis is important both theoretically and practically.

1. In Chapter 3 we examined and analysed Groth16 SNARK, clarifying the somewhat confusing security notion that was unclear around its randomizability. This notion is necessary for its security modelling, and we suggest several ways to transform Groth16 for use in protocol that require black-box extraction; our findings make black-box transformations more efficient.

2. In Chapter 4 we focused on the ceremony of Groth16, and showed how by careful analysis we can prove Groth16 security *together* with the ceremony, simplifying the latter, which has not been done before. Our proof techniques can be of independent interest for malleability analysis of algebraic NIZKs.

3. In Chapter 5 we show that allowing partial malleability in zcash-like transactions can be used to build an atomic swap system, which is simple yet concretely practical. In addition to that, we formalise the notion of primate multi-assets, which started to gain practical traction, yet without a proper security analysis.

4. In Chapter 6 our main focus is on the unique malleability of CH20 NIZK, which was not yet used in practice. We formalize the notion and use it to create an efficient "exploding commitment" scheme, that finds applications

for many AML and similar accountability tasks.

Due to the diverse content of this work, each chapter naturally leads to its own questions regarding open questions and future work.

Starting from weak simulation extractability, one follow-up work by Faonio et al. [Faonio et al., 2023] that we briefly mentioned in the Section 2.6.2 deserves special attention. What it provides is in essence a generalisation of our approach with adding malleability into the SE notion. Its SE definition is extended to any predicate Φ, which captures not just randomizable, but arbitrary controlled-malleable NIZKs. This notion is initially used for the KZG [Kate et al., 2010] polynomial commitment scheme, but it is generally crucial to the contexts where both composability and argument malleability are present. In other words, in the situations where want to ensure that NIZKs for certain statements are sound in the presence of the simulator, yet cannot be mauled into proofs for statements that contradict the application logic. This concern can be seen as a generalised replay attack protection, where an alternative *statement* could be maliciously replayed instead.

At the same time, since one of our main motivations behind analysing weak SE of Groth16 was its UC applications, it is worth noting the ongoing and perspective research direction that intends to improve SNARK composability by adjusting the simulation paradigm instead. For example, [Kerber et al., 2021a] investigates the role of knowledge assumptions in UC directly, while [Ganesh et al., 2023] alternatively relies on the global random oracle to prove (witness)-succinct NIZK security fully within UC. Both approaches seem promising, and arguably complementary to the black-box transformations, which are a simpler and still relatively inexpensive way to integrate NIZKs into UC, albeit perhaps theoretically unsatisfactory due to their “erasure” of the argument’s succinctness.

As of setup ceremonies for SNARKs, our main observation is that the ecosystem at this point seemingly moved away from the monolitic SNARK approach towards more modular thinking. SNARKs used in the modern protocols are quite often either transparent, or based on a universal SRS. The approach of Sonic [Maller et al., 2019] and Plonk [Gabizon et al., 2019] was undoubtedly a great influence on the modular thinking, leading to the popularisation of the so-called interactive oracle proof (IOP) model, where SRS-related issues were abstracted away from the main argument into a polynomial commitment scheme. And even though many projects switched to the transparent IPA (generically, inner product argument) based commitment scheme [Bünz et al., 2021b], some solutions, notably Ethereum, still use the KZG commitment scheme, which is essentially based on a universal SRS. The main advantage of the universal SRS is that ceremonies for it are easier to implement, however I believe the same composability question can be raised about constructing a KZG commitment scheme securely within the bigger argument protocol.

On the topic of DeFi and atomic asset exchange, it seems reasonable to assume that no simple protocol can be practical without an ability to integrate with smart contract platforms, especially recent privacy oriented ones. Zswap can be seen as a simple yet effective extension that can be applied to many private cryptocurrencies. That being said, it still requires extending a layer one solution, while most existing popular exchanges thrive due to their simplicity and variety of delivered features. And although these solutions might be limited in their privacy settings, and private smart contracts do not allow atomic swaps per se, it seems to be a reasonable trade-off practically. This, however, does not mean that private atomic swaps are not a desired functionality — notably, the Anoma protocol[1] focuses on exactly this problem, by generalising transactions to transaction *intents*, allowing users to specify final conditions (similar to Zswap's "offer transaction" exchange conditions), and allow network nodes to satisfy them. The academic research on the topic of atomic and private intent resolution is not sufficiently active at this point, however, and it can be definitely viewed as a potentially fruitful open research question.

Finally, Chapter 6 is concerned with two big problems: the role of simple malleable NIZKs in cryptographic protocols, and the state of accountability and DeFi, with an emphasis of efficiency of simple and modular regulation-compliance primitives. Starting with the latter, in Section 6.6.2 we already mentioned several potential extensions of our excom scheme, such as support for arbitrary polynomials predicates or non-binary explosion values. We believe that our system design is quite flexible, and that it captures a non-trivial amount of potentially useful real-world functionality. However, in terms of regulation compliance in general, it stands to reason that a viable solution will most likely rely on generic, efficient private smart contract systems strengthened by interoperability between many smaller protocols.

The latter topic of the last chapter is updatable algebraic NIZKs, the primitive which

[1] `https://anoma.net/`

I believe has a lot of research potential. As it was already said, small malleable NIZK proofs are an indispensable tool for protocols which involve data update, and more particularly, homomorphic operations on some algebraic structures. They have literally no viable alternatives in this area, where recursive SNARKs are prohibitively expensive, and non-updatable SNARKs or regular algebraic NIZKs would incur performance overhead due to repeated consistency re-proving.

The relatively moderate interest in directly malleable NIZKs, especially within practical cryptography, can be perhaps explained in terms of the more salient short-term goals of the field that overfocuses on generic and powerful primitives, giving rise to versatile solutions that prioritise wide-scope innovation over excellence in a concrete narrowly-defined task. Non-linearity of cycles in both economy and cryptographic research being a given, I am confident that a change to the opposite trend can be expected. And together with the evidence of applicability of malleable primitives in this work and in adjacent areas, I think the notion will to see much more light in the coming decade, and I particularly hope that the field of zero-knowledge will converge towards standardized, reusable and interoperable solutions to benefit privacy needs of all the involved parties.

Bibliography

[awe, 2023] (2023). Awesome zkml repository. `https://github.com/worldcoin/awesome-zkml`. [Online; accessed: 24/08/2023].

[gna, 2023] (2023). Gnark benchmarks. `https://hackmd.io/@gnark/eccbench`. Accessed: 2023-07-25.

[mir, 2023] (2023). MIRACL benchmarks. `https://github.com/miracl/MIRACL/blob/master/docs/miracl-explained/benchmarks.md`. Accessed: 2023-07-25.

[Abdolmaleki et al., 2019] Abdolmaleki, B., Baghery, K., Lipmaa, H., Siim, J., and Zajac, M. (2019). UC-secure CRS generation for SNARKs. In Buchmann, J., Nitaj, A., and eddine Rachidi, T., editors, *AFRICACRYPT 19*, volume 11627 of *LNCS*, pages 99–117. Springer, Heidelberg.

[Abdolmaleki et al., 2017] Abdolmaleki, B., Baghery, K., Lipmaa, H., and Zajac, M. (2017). A subversion-resistant SNARK. In Takagi, T. and Peyrin, T., editors, *ASIACRYPT 2017, Part III*, volume 10626 of *LNCS*, pages 3–33. Springer, Heidelberg.

[Abdolmaleki et al., 2020] Abdolmaleki, B., Ramacher, S., and Slamanig, D. (2020). Lift-and-shift: Obtaining simulation extractable subversion and updatable SNARKs generically. In Ligatti, J., Ou, X., Katz, J., and Vigna, G., editors, *ACM CCS 2020*, pages 1987–2005. ACM Press.

[Acar and Nguyen, 2011] Acar, T. and Nguyen, L. (2011). Revocation for delegatable anonymous credentials. In Catalano, D., Fazio, N., Gennaro, R., and Nicolosi, A., editors, *PKC 2011*, volume 6571 of *LNCS*, pages 423–440. Springer, Heidelberg.

[Aggelakis et al., 2020] Aggelakis, A., Fauzi, P., Korfiatis, G., Louridas, P., Mergoupis-Anagnou, F., Siim, J., and Zajac, M. (2020). A non-interactive shuffle argument with low trust assumptions. In Jarecki, S., editor, *CT-RSA 2020*, volume 12006 of *LNCS*, pages 667–692. Springer, Heidelberg.

[Alonso and Joancomartí, 2018] Alonso, K. M. and Joancomartí, J. H. (2018). Monero - privacy in the blockchain. Cryptology ePrint Archive, Report 2018/535. `https://eprint.iacr.org/2018/535`.

[Ananth et al., 2017] Ananth, P., Cohen, A., and Jain, A. (2017). Cryptography with updates. In Coron, J.-S. and Nielsen, J. B., editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 445–472. Springer, Heidelberg.

[Ananth et al., 2019] Ananth, P., Deshpande, A., Kalai, Y. T., and Lysyanskaya, A. (2019). Fully homomorphic NIZK and NIWI proofs. In Hofheinz, D. and Rosen, A., editors, *TCC 2019, Part II*, volume 11892 of *LNCS*, pages 356–385. Springer, Heidelberg.

[Androulaki et al., 2020] Androulaki, E., Camenisch, J., Caro, A. D., Dubovitskaya, M., Elkhiyaoui, K., and Tackmann, B. (2020). Privacy-preserving auditable token payments in a permissioned blockchain system. In *AFT '20: 2nd ACM Conference on Advances in Financial Technologies, New York, NY, USA, October 21-23, 2020*, pages 255–267. ACM.

[Androulaki et al., 2013] Androulaki, E., Karame, G., Roeschlin, M., Scherer, T., and Capkun, S. (2013). Evaluating user privacy in Bitcoin. In Sadeghi, A.-R., editor, *FC 2013*, volume 7859 of *LNCS*, pages 34–51. Springer, Heidelberg.

[Aranha et al., 2022] Aranha, D. F., Hall-Andersen, M., Nitulescu, A., Pagnin, E., and Yakoubov, S. (2022). Count me in! Extendability for threshold ring signatures. In Hanaoka, G., Shikata, J., and Watanabe, Y., editors, *PKC 2022, Part II*, volume 13178 of *LNCS*, pages 379–406. Springer, Heidelberg.

[Arun et al., 2022] Arun, A., Ganesh, C., Lokam, S., Mopuri, T., and Sridhar, S. (2022). Dew: Transparent constant-sized zkSNARKs. Cryptology ePrint Archive, Report 2022/419. `https://eprint.iacr.org/2022/419`.

[Atapoor and Baghery, 2019] Atapoor, S. and Baghery, K. (2019). Simulation extractability in Groth's zk-SNARK. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pages 336–354. Springer.

[Attema and Cramer, 2020] Attema, T. and Cramer, R. (2020). Compressed Σ-protocol theory and practical application to plug & play secure algorithmics. In Micciancio, D. and Ristenpart, T., editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 513–543. Springer, Heidelberg.

[Badertscher et al., 2021] Badertscher, C., Matt, C., and Waldner, H. (2021). Policy-compliant signatures. In Nissim, K. and Waters, B., editors, *TCC 2021, Part III*, volume 13044 of *LNCS*, pages 350–381. Springer, Heidelberg.

[Baghery, 2019] Baghery, K. (2019). On the efficiency of privacy-preserving smart contract systems. In Buchmann, J., Nitaj, A., and eddine Rachidi, T., editors, *AFRICACRYPT 19*, volume 11627 of *LNCS*, pages 118–136. Springer, Heidelberg.

[Baghery et al., 2020] Baghery, K., Pindado, Z., and Ràfols, C. (2020). Simulation extractable versions of groth's zk-SNARK revisited. In Krenn, S., Shulman, H., and Vaudenay, S., editors, *CANS 20*, volume 12579 of *LNCS*, pages 453–461. Springer, Heidelberg.

[Baghery and Sedaghat, 2021] Baghery, K. and Sedaghat, M. (2021). Tiramisu: Black-box simulation extractable NIZKs in the updatable CRS model. In Conti, M., Stevens, M., and Krenn, S., editors, *CANS 21*, volume 13099 of *LNCS*, pages 531–551. Springer, Heidelberg.

[Barki and Gouget, 2020] Barki, A. and Gouget, A. (2020). Achieving privacy and accountability in traceable digital currency. Cryptology ePrint Archive, Report 2020/1565. `https://eprint.iacr.org/2020/1565`.

[Barreto et al., 2003] Barreto, P. S. L. M., Lynn, B., and Scott, M. (2003). Constructing elliptic curves with prescribed embedding degrees. In Cimato, S., Galdi, C., and Persiano, G., editors, *SCN 02*, volume 2576 of *LNCS*, pages 257–267. Springer, Heidelberg.

[Barreto and Naehrig, 2006] Barreto, P. S. L. M. and Naehrig, M. (2006). Pairing-friendly elliptic curves of prime order. In Preneel, B. and Tavares, S., editors, *SAC 2005*, volume 3897 of *LNCS*, pages 319–331. Springer, Heidelberg.

[Bartusek et al., 2023] Bartusek, J., Garg, S., Jain, A., and Policharla, G. (2023). End-to-end secure messaging with traceability only for illegal content. In Hazay, C. and Stam, M., editors, *Advances in Cryptology - EUROCRYPT 2023 - 42nd*

Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part V, volume 14008 of *Lecture Notes in Computer Science*, pages 35–66. Springer.

[Bauer et al., 2020] Bauer, B., Fuchsbauer, G., and Loss, J. (2020). A classification of computational assumptions in the algebraic group model. In Micciancio, D. and Ristenpart, T., editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 121–151. Springer, Heidelberg.

[Baum et al., 2021] Baum, C., David, B., and Frederiksen, T. K. (2021). P2DEX: Privacy-preserving decentralized cryptocurrency exchange. In Sako, K. and Tippenhauer, N. O., editors, *ACNS 21, Part I*, volume 12726 of *LNCS*, pages 163–194. Springer, Heidelberg.

[Baum et al., 2023] Baum, C., yu Chiang, J. H., David, B., and Frederiksen, T. K. (2023). SoK: Privacy-enhancing technologies in finance. Cryptology ePrint Archive, Report 2023/122. `https://eprint.iacr.org/2023/122`.

[Belenkiy et al., 2009] Belenkiy, M., Camenisch, J., Chase, M., Kohlweiss, M., Lysyanskaya, A., and Shacham, H. (2009). Randomizable proofs and delegatable anonymous credentials. In Halevi, S., editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 108–125. Springer, Heidelberg.

[Bellare et al., 2001] Bellare, M., Boldyreva, A., Desai, A., and Pointcheval, D. (2001). Key-privacy in public-key encryption. In Boyd, C., editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 566–582. Springer, Heidelberg.

[Bellare et al., 2016] Bellare, M., Fuchsbauer, G., and Scafuro, A. (2016). NIZKs with an untrusted CRS: Security in the face of parameter subversion. In Cheon, J. H. and Takagi, T., editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 777–804. Springer, Heidelberg.

[Bellare and Rogaway, 1993] Bellare, M. and Rogaway, P. (1993). Random oracles are practical: A paradigm for designing efficient protocols. In Denning, D. E., Pyle, R., Ganesan, R., Sandhu, R. S., and Ashby, V., editors, *ACM CCS 93*, pages 62–73. ACM Press.

[Ben-Sasson et al., 2014a] Ben-Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., and Virza, M. (2014a). Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE Computer Society Press.

[Ben-Sasson et al., 2013] Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., and Virza, M. (2013). SNARKs for C: Verifying program executions succinctly and in zero knowledge. In Canetti, R. and Garay, J. A., editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 90–108. Springer, Heidelberg.

[Ben-Sasson et al., 2015] Ben-Sasson, E., Chiesa, A., Green, M., Tromer, E., and Virza, M. (2015). Secure sampling of public parameters for succinct zero knowledge proofs. In *2015 IEEE Symposium on Security and Privacy*, pages 287–304. IEEE Computer Society Press.

[Ben-Sasson et al., 2014b] Ben-Sasson, E., Chiesa, A., Tromer, E., and Virza, M. (2014b). Scalable zero knowledge via cycles of elliptic curves. In Garay, J. A. and Gennaro, R., editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 276–294. Springer, Heidelberg.

[Ben-Sasson et al., 2014c] Ben-Sasson, E., Chiesa, A., Tromer, E., and Virza, M. (2014c). Succinct non-interactive zero knowledge for a von neumann architecture. In Fu, K. and Jung, J., editors, *USENIX Security 2014*, pages 781–796. USENIX Association.

[Benhamouda and Lin, 2020] Benhamouda, F. and Lin, H. (2020). Mr NISC: Multiparty reusable non-interactive secure computation. In Pass, R. and Pietrzak, K., editors, *TCC 2020, Part II*, volume 12551 of *LNCS*, pages 349–378. Springer, Heidelberg.

[Béres et al., 2021] Béres, F., Seres, I. A., Benczúr, A. A., and Quintyne-Collins, M. (2021). Blockchain is watching you: Profiling and deanonymizing ethereum users. In *2021 IEEE international conference on decentralized applications and infrastructures (DAPPS)*, pages 69–78. IEEE.

[Bitansky et al., 2013] Bitansky, N., Canetti, R., Chiesa, A., and Tromer, E. (2013). Recursive composition and bootstrapping for SNARKS and proof-carrying data. In Boneh, D., Roughgarden, T., and Feigenbaum, J., editors, *45th ACM STOC*, pages 111–120. ACM Press.

[Blake and Garefalakis, 2004] Blake, I. F. and Garefalakis, T. (2004). On the complexity of the discrete logarithm and diffie–hellman problems. *Journal of Com-*

plexity, 20(2-3):148–170.

[Blazy et al., 2011] Blazy, O., Fuchsbauer, G., Pointcheval, D., and Vergnaud, D. (2011). Signatures on randomizable ciphertexts. In Catalano, D., Fazio, N., Gennaro, R., and Nicolosi, A., editors, *PKC 2011*, volume 6571 of *LNCS*, pages 403–422. Springer, Heidelberg.

[Bogatov et al., 2021] Bogatov, D., De Caro, A., Elkhiyaoui, K., and Tackmann, B. (2021). Anonymous transactions with revocation and auditing in hyperledger fabric. In Conti, M., Stevens, M., and Krenn, S., editors, *CANS 21*, volume 13099 of *LNCS*, pages 435–459. Springer, Heidelberg.

[Boneh et al., 2018] Boneh, D., Bonneau, J., Bünz, B., and Fisch, B. (2018). Verifiable delay functions. In Shacham, H. and Boldyreva, A., editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 757–788. Springer, Heidelberg.

[Boneh and Venkatesan, 1998] Boneh, D. and Venkatesan, R. (1998). Breaking RSA may not be equivalent to factoring. In Nyberg, K., editor, *EUROCRYPT'98*, volume 1403 of *LNCS*, pages 59–71. Springer, Heidelberg.

[Bonneau et al., 2020a] Bonneau, J., Meckler, I., Rao, V., and Shapiro, E. (2020a). Coda: Decentralized cryptocurrency at scale. Cryptology ePrint Archive, Report 2020/352. `https://eprint.iacr.org/2020/352`.

[Bonneau et al., 2020b] Bonneau, J., Meckler, I., Rao, V., and Shapiro, E. (2020b). Mina: Decentralized cryptocurrency at scale. *New York Univ. O (1) Labs, New York, NY, USA, Whitepaper*, pages 1–47.

[Bonneau et al., 2014] Bonneau, J., Narayanan, A., Miller, A., Clark, J., Kroll, J. A., and Felten, E. W. (2014). Mixcoin: Anonymity for bitcoin with accountable mixes. In Christin, N. and Safavi-Naini, R., editors, *FC 2014*, volume 8437 of *LNCS*, pages 486–504. Springer, Heidelberg.

[Bowe et al., 2020] Bowe, S., Chiesa, A., Green, M., Miers, I., Mishra, P., and Wu, H. (2020). ZEXE: Enabling decentralized private computation. In *2020 IEEE Symposium on Security and Privacy*, pages 947–964. IEEE Computer Society Press.

[Bowe and Gabizon, 2018] Bowe, S. and Gabizon, A. (2018). Making groth's zk-SNARK simulation extractable in the random oracle model. Cryptology ePrint

Archive, Report 2018/187. https://eprint.iacr.org/2018/187.

[Bowe et al., 2017a] Bowe, S., Gabizon, A., and Green, M. D. (2017a). A multi-party protocol for constructing the public parameters of the pinocchio zk-SNARK. Cryptology ePrint Archive, Report 2017/602. https://eprint.iacr.org/2017/602.

[Bowe et al., 2017b] Bowe, S., Gabizon, A., and Miers, I. (2017b). Scalable multi-party computation for zk-SNARK parameters in the random beacon model. Cryptology ePrint Archive, Report 2017/1050. https://eprint.iacr.org/2017/1050.

[Bowe et al., 2019] Bowe, S., Grigg, J., and Hopwood, D. (2019). Halo: Recursive proof composition without a trusted setup. Cryptology ePrint Archive, Report 2019/1021. https://eprint.iacr.org/2019/1021.

[Bünz et al., 2020a] Bünz, B., Agrawal, S., Zamani, M., and Boneh, D. (2020a). Zether: Towards privacy in a smart contract world. In Bonneau, J. and Heninger, N., editors, *FC 2020*, volume 12059 of *LNCS*, pages 423–443. Springer, Heidelberg.

[Bünz et al., 2018] Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., and Maxwell, G. (2018). Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy*, pages 315–334. IEEE Computer Society Press.

[Bünz et al., 2020b] Bünz, B., Chiesa, A., Lin, W., Mishra, P., and Spooner, N. (2020b). Proof-carrying data without succinct arguments. Cryptology ePrint Archive, Report 2020/1618. https://eprint.iacr.org/2020/1618.

[Bünz et al., 2021a] Bünz, B., Chiesa, A., Lin, W., Mishra, P., and Spooner, N. (2021a). Proof-carrying data without succinct arguments. In Malkin, T. and Peikert, C., editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 681–710, Virtual Event. Springer, Heidelberg.

[Bünz et al., 2020c] Bünz, B., Chiesa, A., Mishra, P., and Spooner, N. (2020c). Recursive proof composition from accumulation schemes. In Pass, R. and Pietrzak, K., editors, *TCC 2020, Part II*, volume 12551 of *LNCS*, pages 1–18. Springer, Heidelberg.

[Bünz and Fisch, 2022] Bünz, B. and Fisch, B. (2022). Schwartz-zippel for multilinear polynomials mod N. Cryptology ePrint Archive, Report 2022/458. `https://eprint.iacr.org/2022/458`.

[Bünz et al., 2021b] Bünz, B., Maller, M., Mishra, P., Tyagi, N., and Vesely, P. (2021b). Proofs for inner pairing products and applications. In Tibouchi, M. and Wang, H., editors, *ASIACRYPT 2021, Part III*, volume 13092 of *LNCS*, pages 65–97. Springer, Heidelberg.

[Camenisch et al., 2017] Camenisch, J., Drijvers, M., and Dubovitskaya, M. (2017). Practical UC-secure delegatable credentials with attributes and their application to blockchain. In Thuraisingham, B. M., Evans, D., Malkin, T., and Xu, D., editors, *ACM CCS 2017*, pages 683–699. ACM Press.

[Camenisch et al., 1996] Camenisch, J., Maurer, U. M., and Stadler, M. (1996). Digital payment systems with passive anonymity-revoking trustees. In Bertino, E., Kurth, H., Martella, G., and Montolivo, E., editors, *ESORICS'96*, volume 1146 of *LNCS*, pages 33–43. Springer, Heidelberg.

[Campanelli et al., 2021] Campanelli, M., Engelmann, F., and Orlandi, C. (2021). Zero-knowledge for homomorphic key-value commitments with applications to privacy-preserving ledgers. Cryptology ePrint Archive, Report 2021/1678. `https://eprint.iacr.org/2021/1678`.

[Canetti, 2001] Canetti, R. (2001). Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press.

[Canetti et al., 2014] Canetti, R., Jain, A., and Scafuro, A. (2014). Practical UC security with a global random oracle. In Ahn, G.-J., Yung, M., and Li, N., editors, *ACM CCS 2014*, pages 597–608. ACM Press.

[Canetti et al., 2002] Canetti, R., Lindell, Y., Ostrovsky, R., and Sahai, A. (2002). Universally composable two-party and multi-party secure computation. In *34th ACM STOC*, pages 494–503. ACM Press.

[Cascudo et al., 2019] Cascudo, I., Damgård, I., David, B., Döttling, N., Dowsley, R., and Giacomelli, I. (2019). Efficient UC commitment extension with homomorphism for free (and applications). In Galbraith, S. D. and Moriai, S., editors, *ASIACRYPT 2019, Part II*, volume 11922 of *LNCS*, pages 606–635. Springer, Heidelberg.

[Catalano et al., 2022] Catalano, D., Fiore, D., and Tucker, I. (2022). Additive-homomorphic functional commitments and applications to homomorphic signatures. In Agrawal, S. and Lin, D., editors, *ASIACRYPT 2022, Part IV*, volume 13794 of *LNCS*, pages 159–188. Springer, Heidelberg.

[Chase et al., 2012] Chase, M., Kohlweiss, M., Lysyanskaya, A., and Meiklejohn, S. (2012). Malleable proof systems and applications. In Pointcheval, D. and Johansson, T., editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 281–300. Springer, Heidelberg.

[Chase et al., 2013a] Chase, M., Kohlweiss, M., Lysyanskaya, A., and Meiklejohn, S. (2013a). Succinct malleable NIZKs and an application to compact shuffles. In Sahai, A., editor, *TCC 2013*, volume 7785 of *LNCS*, pages 100–119. Springer, Heidelberg.

[Chase et al., 2013b] Chase, M., Kohlweiss, M., Lysyanskaya, A., and Meiklejohn, S. (2013b). Verifiable elections that scale for free. In Kurosawa, K. and Hanaoka, G., editors, *PKC 2013*, volume 7778 of *LNCS*, pages 479–496. Springer, Heidelberg.

[Chase et al., 2014] Chase, M., Kohlweiss, M., Lysyanskaya, A., and Meiklejohn, S. (2014). Malleable signatures: New definitions and delegatable anonymous credentials. In Datta, A. and Fournet, C., editors, *CSF 2014 Computer Security Foundations Symposium*, pages 199–213. IEEE Computer Society Press.

[Chase and Lysyanskaya, 2006] Chase, M. and Lysyanskaya, A. (2006). On signatures of knowledge. In Dwork, C., editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 78–96. Springer, Heidelberg.

[Chaum et al., 1988] Chaum, D., Crépeau, C., and Damgård, I. (1988). Multiparty unconditionally secure protocols (abstract) (informal contribution). In Pomerance, C., editor, *CRYPTO'87*, volume 293 of *LNCS*, page 462. Springer, Heidelberg.

[Chen et al., 2020] Chen, Y., Ma, X., Tang, C., and Au, M. H. (2020). PGC: Decentralized confidential payment system with auditability. In Chen, L., Li, N., Liang, K., and Schneider, S. A., editors, *ESORICS 2020, Part I*, volume 12308 of *LNCS*, pages 591–610. Springer, Heidelberg.

[Cheon, 2006] Cheon, J. H. (2006). Security analysis of the strong Diffie-Hellman problem. In Vaudenay, S., editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 1–11. Springer, Heidelberg.

[Cheon et al., 2017] Cheon, J. H., Kim, A., Kim, M., and Song, Y. S. (2017). Homomorphic encryption for arithmetic of approximate numbers. In Takagi, T. and Peyrin, T., editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 409–437. Springer, Heidelberg.

[Chiesa et al., 2020] Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, P., and Ward, N. P. (2020). Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In Canteaut, A. and Ishai, Y., editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 738–768. Springer, Heidelberg.

[Chiesa and Tromer, 2010] Chiesa, A. and Tromer, E. (2010). Proof-carrying data and hearsay arguments from signature cards. In Yao, A. C.-C., editor, *ICS 2010*, pages 310–331. Tsinghua University Press.

[Chu et al., 2020] Chu, S., Xia, Q., and Zhang, Z. (2020). Manta: Privacy preserving decentralized exchange. Cryptology ePrint Archive, Report 2020/1607. `https://ia.cr/2020/1607`.

[Connolly et al., 2022] Connolly, A., Lafourcade, P., and Perez-Kempner, O. (2022). Improved constructions of anonymous credentials from structure-preserving signatures on equivalence classes. In Hanaoka, G., Shikata, J., and Watanabe, Y., editors, *PKC 2022, Part I*, volume 13177 of *LNCS*, pages 409–438. Springer, Heidelberg.

[Couteau and Hartmann, 2020] Couteau, G. and Hartmann, D. (2020). Shorter non-interactive zero-knowledge arguments and ZAPs for algebraic languages. In Micciancio, D. and Ristenpart, T., editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 768–798. Springer, Heidelberg.

[Couteau et al., 2021] Couteau, G., Klooß, M., Lin, H., and Reichle, M. (2021). Efficient range proofs with transparent setup from bounded integer commitments. In Canteaut, A. and Standaert, F.-X., editors, *EUROCRYPT 2021, Part III*, volume 12698 of *LNCS*, pages 247–277. Springer, Heidelberg.

[Crites and Lysyanskaya, 2019] Crites, E. C. and Lysyanskaya, A. (2019). Delegatable anonymous credentials from mercurial signatures. In Matsui, M., editor, *CT-RSA 2019*, volume 11405 of *LNCS*, pages 535–555. Springer, Heidelberg.

[Damgård, 1992] Damgård, I. (1992). Towards practical public key systems secure against chosen ciphertext attacks. In Feigenbaum, J., editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 445–456. Springer, Heidelberg.

[Damgård et al., 2014] Damgård, I., David, B. M., Giacomelli, I., and Nielsen, J. B. (2014). Compact VSS and efficient homomorphic UC commitments. In Sarkar, P. and Iwata, T., editors, *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 213–232. Springer, Heidelberg.

[Damgård et al., 2021] Damgård, I., Ganesh, C., Khoshakhlagh, H., Orlandi, C., and Siniscalchi, L. (2021). Balancing privacy and accountability in blockchain identity management. In Paterson, K. G., editor, *CT-RSA 2021*, volume 12704 of *LNCS*, pages 552–576. Springer, Heidelberg.

[Damgard et al., 2008] Damgard, I., Geisler, M., and Kroigard, M. (2008). Homomorphic encryption and secure comparison. *International Journal of Applied Cryptography*, 1(1):22–31.

[Damgård et al., 2012] Damgård, I., Pastro, V., Smart, N. P., and Zakarias, S. (2012). Multiparty computation from somewhat homomorphic encryption. In Safavi-Naini, R. and Canetti, R., editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 643–662. Springer, Heidelberg.

[Danezis et al., 2014] Danezis, G., Fournet, C., Groth, J., and Kohlweiss, M. (2014). Square span programs with applications to succinct NIZK arguments. In Sarkar, P. and Iwata, T., editors, *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*, pages 532–550. Springer, Heidelberg.

[Danezis and Meiklejohn, 2016] Danezis, G. and Meiklejohn, S. (2016). Centrally banked cryptocurrencies. In *NDSS 2016*. The Internet Society.

[de Perthuis and Pointcheval, 2022] de Perthuis, P. and Pointcheval, D. (2022). Two-client inner-product functional encryption with an application to money-laundering detection. In Yin, H., Stavrou, A., Cremers, C., and Shi, E., editors, *ACM CCS 2022*, pages 725–737. ACM Press.

[De Santis et al., 2001] De Santis, A., Di Crescenzo, G., Ostrovsky, R., Persiano, G., and Sahai, A. (2001). Robust non-interactive zero knowledge. In Kilian, J., editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 566–598. Springer, Heidelberg.

[Delignat-Lavaud et al., 2016] Delignat-Lavaud, A., Fournet, C., Kohlweiss, M., and Parno, B. (2016). Cinderella: Turning shabby X.509 certificates into elegant anonymous credentials with the magic of verifiable computation. In *2016 IEEE Symposium on Security and Privacy*, pages 235–254. IEEE Computer Society Press.

[Dent, 2002] Dent, A. W. (2002). Adapting the weaknesses of the random oracle model to the generic group model. In Zheng, Y., editor, *ASIACRYPT 2002*, volume 2501 of *LNCS*, pages 100–109. Springer, Heidelberg.

[Desfontaines and Pejó, 2020] Desfontaines, D. and Pejó, B. (2020). SoK: Differential privacies. *PoPETs*, 2020(2):288–313.

[Deshpande and Herlihy, 2020] Deshpande, A. and Herlihy, M. (2020). Privacy-preserving cross-chain atomic swaps. In Bernhard, M., Bracciali, A., Camp, L. J., Matsuo, S., Maurushat, A., Rønne, P. B., and Sala, M., editors, *FC 2020 Workshops*, volume 12063 of *LNCS*, pages 540–549. Springer, Heidelberg.

[Ding et al., 2019] Ding, D., Li, K., Jia, L., Li, Z., Li, J., and Sun, Y. (2019). Privacy protection for blockchains with account and multi-asset model. *China Communications*, 16(6):69–79.

[Dingledine et al., 2004] Dingledine, R., Mathewson, N., Syverson, P. F., et al. (2004). Tor: The second-generation onion router. In *USENIX security symposium*, volume 4, pages 303–320.

[Dodis et al., 2010] Dodis, Y., Haralambiev, K., López-Alt, A., and Wichs, D. (2010). Cryptography against continuous memory attacks. In *51st FOCS*, pages 511–520. IEEE Computer Society Press.

[ElGamal, 1984] ElGamal, T. (1984). A public key cryptosystem and a signature scheme based on discrete logarithms. In Blakley, G. R. and Chaum, D., editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 10–18. Springer, Heidelberg.

[ElGamal, 1985] ElGamal, T. (1985). A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472.

[Engelmann et al., 2021] Engelmann, F., Müller, L., Peter, A., Kargl, F., and Bösch, C. (2021). SwapCT: Swap confidential transactions for privacy-preserving multi-token exchanges. *PoPETs*, 2021(4):270–290.

[Faonio et al., 2023] Faonio, A., Fiore, D., Kohlweiss, M., Russo, L., and Zajac, M. (2023). From polynomial iop and commitments to non-malleable zksnarks. *Cryptology ePrint Archive*.

[Faust et al., 2012] Faust, S., Kohlweiss, M., Marson, G. A., and Venturi, D. (2012). On the non-malleability of the Fiat-Shamir transform. In Galbraith, S. D. and Nandi, M., editors, *INDOCRYPT 2012*, volume 7668 of *LNCS*, pages 60–79. Springer, Heidelberg.

[Fauzi et al., 2017] Fauzi, P., Lipmaa, H., Siim, J., and Zajac, M. (2017). An efficient pairing-based shuffle argument. In Takagi, T. and Peyrin, T., editors, *ASIACRYPT 2017, Part II*, volume 10625 of *LNCS*, pages 97–127. Springer, Heidelberg.

[Fauzi et al., 2019] Fauzi, P., Meiklejohn, S., Mercer, R., and Orlandi, C. (2019). Quisquis: A new design for anonymous cryptocurrencies. In Galbraith, S. D. and Moriai, S., editors, *ASIACRYPT 2019, Part I*, volume 11921 of *LNCS*, pages 649–678. Springer, Heidelberg.

[Fiat and Shamir, 1987] Fiat, A. and Shamir, A. (1987). How to prove yourself: Practical solutions to identification and signature problems. In Odlyzko, A. M., editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg.

[Fischlin, 2005] Fischlin, M. (2005). Communication-efficient non-interactive proofs of knowledge with online extractors. In Shoup, V., editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 152–168. Springer, Heidelberg.

[Frankle et al., 2018] Frankle, J., Park, S., Shaar, D., Goldwasser, S., and Weitzner, D. J. (2018). Practical accountability of secret processes. In Enck, W. and Felt, A. P., editors, *USENIX Security 2018*, pages 657–674. USENIX Association.

[Fuchsbauer, 2011] Fuchsbauer, G. (2011). Commuting signatures and verifiable encryption. In Paterson, K. G., editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 224–245. Springer, Heidelberg.

[Fuchsbauer, 2018] Fuchsbauer, G. (2018). Subversion-zero-knowledge SNARKs. In Abdalla, M. and Dahab, R., editors, *PKC 2018, Part I*, volume 10769 of *LNCS*, pages 315–347. Springer, Heidelberg.

[Fuchsbauer et al., 2018] Fuchsbauer, G., Kiltz, E., and Loss, J. (2018). The algebraic group model and its applications. In Shacham, H. and Boldyreva, A., editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 33–62. Springer, Heidelberg.

[Fuchsbauer et al., 2019] Fuchsbauer, G., Orrù, M., and Seurin, Y. (2019). Aggregate cash systems: A cryptographic investigation of Mimblewimble. In Ishai, Y. and Rijmen, V., editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 657–689. Springer, Heidelberg.

[Fuchsbauer et al., 2020] Fuchsbauer, G., Plouviez, A., and Seurin, Y. (2020). Blind schnorr signatures and signed ElGamal encryption in the algebraic group model. In Canteaut, A. and Ishai, Y., editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 63–95. Springer, Heidelberg.

[Gabizon, 2019] Gabizon, A. (2019). On the security of the BCTV pinocchio zk-SNARK variant. Cryptology ePrint Archive, Report 2019/119. `https://eprint.iacr.org/2019/119`.

[Gabizon et al., 2019] Gabizon, A., Williamson, Z. J., and Ciobotaru, O. (2019). PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953. `https://eprint.iacr.org/2019/953`.

[Galbraith et al., 2006] Galbraith, S., Paterson, K., and Smart, N. (2006). Pairings for cryptographers. Cryptology ePrint Archive, Report 2006/165. `https://eprint.iacr.org/2006/165`.

[Gama et al., 2020] Gama, N., Georgieva, M., Jetchev, D., McCarthy, K., Odersky, J., Petric, A., and Tang, A. S. (2020). Detecting money laundering activities via secure multi-party computation for structural similarities in flow networks. In *Real World Cryptography*.

[Ganesh et al., 2022a] Ganesh, C., Khoshakhlagh, H., Kohlweiss, M., Nitulescu, A., and Zając, M. (2022a). What makes fiat–shamir zksnarks (updatable srs) simulation extractable? In *International Conference on Security and Cryptography for Networks*, pages 735–760. Springer.

[Ganesh et al., 2022b] Ganesh, C., Kondi, Y., Orlandi, C., Pancholi, M., Takahashi, A., and Tschudi, D. (2022b). Witness-succinct universally-composable SNARKs. Cryptology ePrint Archive, Report 2022/1618. `https://eprint.iacr.org/2022/1618`.

[Ganesh et al., 2023] Ganesh, C., Kondi, Y., Orlandi, C., Pancholi, M., Takahashi, A., and Tschudi, D. (2023). Witness-succinct universally-composable SNARKs. In Hazay, C. and Stam, M., editors, *EUROCRYPT 2023, Part II*, volume 14005 of *LNCS*, pages 315–346. Springer, Heidelberg.

[Gao et al., 2019] Gao, Z., Xu, L., Kasichainula, K., Chen, L., Carbunar, B., and Shi, W. (2019). Private and atomic exchange of assets over zero knowledge based payment ledger. *arXiv preprint arXiv:1909.06535*.

[Garay et al., 2007] Garay, J. A., Schoenmakers, B., and Villegas, J. (2007). Practical and secure solutions for integer comparison. In Okamoto, T. and Wang, X., editors, *PKC 2007*, volume 4450 of *LNCS*, pages 330–342. Springer, Heidelberg.

[Gennaro et al., 2013] Gennaro, R., Gentry, C., Parno, B., and Raykova, M. (2013). Quadratic span programs and succinct NIZKs without PCPs. In Johansson, T. and Nguyen, P. Q., editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 626–645. Springer, Heidelberg.

[Gennaro et al., 2007] Gennaro, R., Jarecki, S., Krawczyk, H., and Rabin, T. (2007). Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology*, 20(1):51–83.

[Gentry, 2009] Gentry, C. (2009). Fully homomorphic encryption using ideal lattices. In Mitzenmacher, M., editor, *41st ACM STOC*, pages 169–178. ACM Press.

[Gentry and Wichs, 2011] Gentry, C. and Wichs, D. (2011). Separating succinct non-interactive arguments from all falsifiable assumptions. In Fortnow, L. and Vadhan, S. P., editors, *43rd ACM STOC*, pages 99–108. ACM Press.

[Ghadafi et al., 2010] Ghadafi, E., Smart, N. P., and Warinschi, B. (2010). Groth-Sahai proofs revisited. In Nguyen, P. Q. and Pointcheval, D., editors, *PKC 2010*, volume 6056 of *LNCS*, pages 177–192. Springer, Heidelberg.

[Gilad et al., 2017] Gilad, Y., Hemo, R., Micali, S., Vlachos, G., and Zeldovich, N. (2017). Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th symposium on operating systems principles*, pages 51–68.

[Goldreich, 2001] Goldreich, O. (2001). *Foundations of Cryptography: Basic Tools*, volume 1. Cambridge University Press, Cambridge, UK.

[Goldreich et al., 1987] Goldreich, O., Micali, S., and Wigderson, A. (1987). How to play any mental game or A completeness theorem for protocols with honest majority. In Aho, A., editor, *19th ACM STOC*, pages 218–229. ACM Press.

[Goldwasser and Kalai, 2016] Goldwasser, S. and Kalai, Y. T. (2016). Cryptographic assumptions: A position paper. In Kushilevitz, E. and Malkin, T., editors, *TCC 2016-A, Part I*, volume 9562 of *LNCS*, pages 505–522. Springer, Heidelberg.

[Goldwasser et al., 1985] Goldwasser, S., Micali, S., and Rackoff, C. (1985). The knowledge complexity of interactive proof-systems (extended abstract). In *17th ACM STOC*, pages 291–304. ACM Press.

[Goldwasser et al., 1988] Goldwasser, S., Micali, S., and Rivest, R. L. (1988). A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308.

[Goldwasser and Park, 2018] Goldwasser, S. and Park, S. (2018). Public accountability vs. secret laws: Can they coexist? Cryptology ePrint Archive, Report 2018/664. `https://eprint.iacr.org/2018/664`.

[Goodman, 2014] Goodman, L. (2014). Tezos: A self-amending crypto-ledger position paper. *Aug*, 3:2014.

[Grassi et al., 2019] Grassi, L., Kales, D., Khovratovich, D., Roy, A., Rechberger, C., and Schofnegger, M. (2019). Starkad and Poseidon: New hash functions for zero knowledge proof systems. Cryptology ePrint Archive, Report 2019/458. `https://eprint.iacr.org/2019/458`.

[Green et al., 2021] Green, M., Kaptchuk, G., and Laer, G. V. (2021). Abuse resistant law enforcement access systems. In Canteaut, A. and Standaert, F.-X., editors, *EUROCRYPT 2021, Part III*, volume 12698 of *LNCS*, pages 553–583. Springer, Heidelberg.

[Groth, 2006] Groth, J. (2006). Simulation-sound NIZK proofs for a practical language and constant size group signatures. In Lai, X. and Chen, K., editors, *ASIACRYPT 2006*, volume 4284 of *LNCS*, pages 444–459. Springer, Heidelberg.

[Groth, 2010] Groth, J. (2010). Short pairing-based non-interactive zero-knowledge arguments. In Abe, M., editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 321–340. Springer, Heidelberg.

[Groth, 2016] Groth, J. (2016). On the size of pairing-based non-interactive arguments. In Fischlin, M. and Coron, J.-S., editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Heidelberg.

[Groth et al., 2018] Groth, J., Kohlweiss, M., Maller, M., Meiklejohn, S., and Miers, I. (2018). Updatable and universal common reference strings with applications to zk-SNARKs. In Shacham, H. and Boldyreva, A., editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 698–728. Springer, Heidelberg.

[Groth and Maller, 2017] Groth, J. and Maller, M. (2017). Snarky signatures: Minimal signatures of knowledge from simulation-extractable SNARKs. In Katz, J. and Shacham, H., editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 581–612. Springer, Heidelberg.

[Groth et al., 2006] Groth, J., Ostrovsky, R., and Sahai, A. (2006). Perfect non-interactive zero knowledge for NP. In Vaudenay, S., editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 339–358. Springer, Heidelberg.

[Groth and Sahai, 2008] Groth, J. and Sahai, A. (2008). Efficient non-interactive proof systems for bilinear groups. In Smart, N. P., editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 415–432. Springer, Heidelberg.

[Han et al., 2020] Han, R., Yu, J., and Lin, H. (2020). RandChain: Decentralised randomness beacon from sequential proof-of-work. Cryptology ePrint Archive, Report 2020/1033. `https://eprint.iacr.org/2020/1033`.

[Hanke et al., 2018] Hanke, T., Movahedi, M., and Williams, D. (2018). Dfinity technology overview series, consensus system. *arXiv preprint arXiv:1805.04548*. https://arxiv.org/abs/1805.04548.

[Hanser and Slamanig, 2014] Hanser, C. and Slamanig, D. (2014). Structure-preserving signatures on equivalence classes and their application to anonymous credentials. In Sarkar, P. and Iwata, T., editors, *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*, pages 491–511. Springer, Heidelberg.

[Hébant et al., 2020] Hébant, C., Phan, D. H., and Pointcheval, D. (2020). Linearly-homomorphic signatures and scalable mix-nets. In Kiayias, A., Kohlweiss, M., Wallden, P., and Zikas, V., editors, *PKC 2020, Part II*, volume 12111 of *LNCS*, pages 597–627. Springer, Heidelberg.

[Heilman et al., 2017] Heilman, E., Alshenibr, L., Baldimtsi, F., Scafuro, A., and Goldberg, S. (2017). TumbleBit: An untrusted bitcoin-compatible anonymous payment hub. In *NDSS 2017*. The Internet Society.

[Hopwood et al., 2022] Hopwood, D., Bowe, S., Hornby, T., and Wilcox, N. (2022). Zcash protocol specification, version 2022.3.8. https://github.com/zcash/zips/blob/master/protocol/protocol.pdf. [Online; accessed: 24/08/2023].

[Kate et al., 2010] Kate, A., Zaverucha, G. M., and Goldberg, I. (2010). Constant-size commitments to polynomials and their applications. In Abe, M., editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 177–194. Springer, Heidelberg.

[Kerber et al., 2021a] Kerber, T., Kiayias, A., and Kohlweiss, M. (2021a). Composition with knowledge assumptions. In Malkin, T. and Peikert, C., editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 364–393, Virtual Event. Springer, Heidelberg.

[Kerber et al., 2021b] Kerber, T., Kiayias, A., and Kohlweiss, M. (2021b). KACHINA - foundations of private smart contracts. In Küsters, R. and Naumann, D., editors, *CSF 2021 Computer Security Foundations Symposium*, pages 1–16. IEEE Computer Society Press.

[Kerber et al., 2019] Kerber, T., Kiayias, A., Kohlweiss, M., and Zikas, V. (2019). Ouroboros crypsinous: Privacy-preserving proof-of-stake. In *2019 IEEE Symposium on Security and Privacy*, pages 157–174. IEEE Computer Society Press.

[Khalili et al., 2019] Khalili, M., Slamanig, D., and Dakhilalian, M. (2019). Structure-preserving signatures on equivalence classes from standard assumptions. In Galbraith, S. D. and Moriai, S., editors, *ASIACRYPT 2019, Part III*, volume 11923 of *LNCS*, pages 63–93. Springer, Heidelberg.

[Kiayias et al., 2022] Kiayias, A., Kohlweiss, M., and Sarencheh, A. (2022). PEReDi: Privacy-enhanced, regulated and distributed central bank digital currencies. In Yin, H., Stavrou, A., Cremers, C., and Shi, E., editors, *ACM CCS 2022*, pages 1739–1752. ACM Press.

[Kiayias et al., 2017] Kiayias, A., Russell, A., David, B., and Oliynykov, R. (2017). Ouroboros: A provably secure proof-of-stake blockchain protocol. In Katz, J. and Shacham, H., editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 357–388. Springer, Heidelberg.

[Kilian, 1992] Kilian, J. (1992). A note on efficient zero-knowledge proofs and arguments (extended abstract). In *24th ACM STOC*, pages 723–732. ACM Press.

[Kim et al., 2019] Kim, J., Lee, J., and Oh, H. (2019). Updatable CRS simulation-extractable zk-SNARKs with a single verification. Cryptology ePrint Archive, Report 2019/586. `https://eprint.iacr.org/2019/586`.

[Kim and Jeong, 2017] Kim, T. and Jeong, J. (2017). Extended tower number field sieve with application to finite fields of arbitrary composite extension degree. In Fehr, S., editor, *PKC 2017, Part I*, volume 10174 of *LNCS*, pages 388–408. Springer, Heidelberg.

[Koblitz, 1987] Koblitz, N. (1987). Elliptic curve cryptosystems. *Mathematics of computation*, 48(177):203–209.

[Koblitz and Menezes, 2015] Koblitz, N. and Menezes, A. (2015). The random oracle model: A twenty-year retrospective. Cryptology ePrint Archive, Report 2015/140. `https://eprint.iacr.org/2015/140`.

[Kohlweiss et al., 2023] Kohlweiss, M., Lysyanskaya, A., and Nguyen, A. (2023). Privacy-preserving blueprints. In Hazay, C. and Stam, M., editors, *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part II*, volume 14005 of *Lecture Notes in Computer Science*, pages 594–625. Springer.

[Kohlweiss and Zając, 2021] Kohlweiss, M. and Zając, M. (2021). On simulation-extractability of universal zkSNARKs. Cryptology ePrint Archive, Report 2021/511. `https://eprint.iacr.org/2021/511`.

[Konvalina, 2000] Konvalina, J. (2000). A unified interpretation of the binomial coefficients, the stirling numbers, and the gaussian coefficients. *The American Mathematical Monthly*, 107(10):901–910.

[Kosba et al., 2015] Kosba, A., Zhao, Z., Miller, A., Qian, Y., Chan, H., Papamanthou, C., Pass, R., shelat, a., and Shi, E. (2015). C∅c∅: A framework for building composable zero-knowledge proofs. Cryptology ePrint Archive, Report 2015/1093. `https://eprint.iacr.org/2015/1093`.

[Kosba et al., 2016] Kosba, A. E., Miller, A., Shi, E., Wen, Z., and Papamanthou, C. (2016). Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *2016 IEEE Symposium on Security and Privacy*, pages 839–858. IEEE Computer Society Press.

[Kosba et al., 2020] Kosba, A. E., Papadopoulos, D., Papamanthou, C., and Song, D. (2020). MIRAGE: Succinct arguments for randomized algorithms with applications to universal zk-SNARKs. In Capkun, S. and Roesner, F., editors, *USENIX Security 2020*, pages 2129–2146. USENIX Association.

[Kothapalli et al., 2021] Kothapalli, A., Setty, S., and Tzialla, I. (2021). Nova: Recursive zero-knowledge arguments from folding schemes. Cryptology ePrint Archive, Report 2021/370. `https://eprint.iacr.org/2021/370`.

[Kothapalli et al., 2022] Kothapalli, A., Setty, S., and Tzialla, I. (2022). Nova: Recursive zero-knowledge arguments from folding schemes. In Dodis, Y. and Shrimpton, T., editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 359–388. Springer, Heidelberg.

[Kozaki et al., 2007] Kozaki, S., Kutsuma, T., and Matsuo, K. (2007). Remarks on cheon's algorithms for pairing-related problems. In *International Conference on Pairing-Based Cryptography*, pages 302–316. Springer.

[Kurosawa, 2002] Kurosawa, K. (2002). Multi-recipient public-key encryption with shortened ciphertext. In Naccache, D. and Paillier, P., editors, *PKC 2002*, volume 2274 of *LNCS*, pages 48–63. Springer, Heidelberg.

[Kwon and Buchman, 2019] Kwon, J. and Buchman, E. (2019). Cosmos whitepaper. *A Netw. Distrib. Ledgers*, 27.

[Lai et al., 2019] Lai, R. W. F., Ronge, V., Ruffing, T., Schröder, D., Thyagarajan, S. A. K., and Wang, J. (2019). Omniring: Scaling private payments without trusted setup. In Cavallaro, L., Kinder, J., Wang, X., and Katz, J., editors, *ACM CCS 2019*, pages 31–48. ACM Press.

[Lee, 2021] Lee, J. (2021). Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. In Nissim, K. and Waters, B., editors, *TCC 2021, Part II*, volume 13043 of *LNCS*, pages 1–34. Springer, Heidelberg.

[Lee et al., 2019] Lee, J., Choi, J., Kim, J., and Oh, H. (2019). SAVER: Snark-friendly, additively-homomorphic, and verifiable encryption and decryption with rerandomization. Cryptology ePrint Archive, Report 2019/1270. `https://eprint.iacr.org/2019/1270`.

[Liang et al., 2022] Liang, M., Karantaidou, I., Baldimtsi, F., Gordon, S. D., and Varia, M. (2022). (ϵ, δ)-indistinguishable mixing for cryptocurrencies. *PoPETs*, 2022(1):49–74.

[Lindbergh, 2011] Lindbergh, A. M. (2011). *Gift from the Sea*. Pantheon.

[Lipmaa, 2012] Lipmaa, H. (2012). Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In Cramer, R., editor, *TCC 2012*, volume 7194 of *LNCS*, pages 169–189. Springer, Heidelberg.

[Lipmaa, 2019] Lipmaa, H. (2019). Simulation-extractable SNARKs revisited. Cryptology ePrint Archive, Report 2019/612. `https://eprint.iacr.org/2019/612`.

[Lipp et al., 2021] Lipp, M., Kogler, A., Oswald, D. F., Schwarz, M., Easdon, C., Canella, C., and Gruss, D. (2021). PLATYPUS: Software-based power side-channel attacks on x86. In *2021 IEEE Symposium on Security and Privacy*, pages 355–371. IEEE Computer Society Press.

[Maller et al., 2019] Maller, M., Bowe, S., Kohlweiss, M., and Meiklejohn, S. (2019). Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In Cavallaro, L., Kinder, J., Wang, X., and Katz, J., editors, *ACM CCS 2019*, pages 2111–2128. ACM Press.

[Maurer, 2009] Maurer, U. M. (2009). Unifying zero-knowledge proofs of knowledge. In Preneel, B., editor, *AFRICACRYPT 09*, volume 5580 of *LNCS*, pages 272–286. Springer, Heidelberg.

[Maxwell, 2013] Maxwell, G. (2013). Coinjoin: Bitcoin privacy for the real world. `https://bitcointalk.org/?topic=279249`.

[Meiklejohn and Mercer, 2018] Meiklejohn, S. and Mercer, R. (2018). Möbius: Trustless tumbling for transaction privacy. *PoPETs*, 2018(2):105–121.

[Meiklejohn et al., 2016] Meiklejohn, S., Pomarole, M., Jordan, G., Levchenko, K., McCoy, D., Voelker, G. M., and Savage, S. (2016). A fistful of bitcoins: characterizing payments among men with no names. *Commun. ACM*, 59(4):86–93.

[Miers et al., 2013] Miers, I., Garman, C., Green, M., and Rubin, A. D. (2013). Zerocoin: Anonymous distributed E-cash from Bitcoin. In *2013 IEEE Symposium on Security and Privacy*, pages 397–411. IEEE Computer Society Press.

[Moshkovitz, 2010] Moshkovitz, D. (2010). An alternative proof of the schwartz-zippel lemma. In *Electron. Colloquium Comput. Complex.*, volume 17, page 96.

[Muralidhara and Sen, 2007] Muralidhara, V. N. and Sen, S. (2007). A result on the distribution of quadratic residues with applications to elliptic curve cryptography. In Srinathan, K., Rangan, C. P., and Yung, M., editors, *INDOCRYPT 2007*, volume 4859 of *LNCS*, pages 48–57. Springer, Heidelberg.

[Nakamoto, 2008] Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. *Decentralized business review*.

[Narula et al., 2018] Narula, N., Vasquez, W., and Virza, M. (2018). zkledger: Privacy-preserving auditing for distributed ledgers. In Banerjee, S. and Seshan, S., editors, *15th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2018, Renton, WA, USA, April 9-11, 2018*, pages 65–80. USENIX Association.

[Naveh and Tromer, 2016] Naveh, A. and Tromer, E. (2016). PhotoProof: Cryptographic image authentication for any set of permissible transformations. In *2016*

IEEE Symposium on Security and Privacy, pages 255–271. IEEE Computer Society Press.

[Noether et al., 2016] Noether, S., Mackenzie, A., et al. (2016). Ring confidential transactions. *Ledger*, 1:1–18.

[Oren, 1987] Oren, Y. (1987). On the cunning power of cheating verifiers: Some observations about zero knowledge proofs (extended abstract). In *28th FOCS*, pages 462–471. IEEE Computer Society Press.

[Paillier and Vergnaud, 2005] Paillier, P. and Vergnaud, D. (2005). Discrete-log-based signatures may not be equivalent to discrete log. In Roy, B. K., editor, *ASIACRYPT 2005*, volume 3788 of *LNCS*, pages 1–20. Springer, Heidelberg.

[Parno et al., 2013] Parno, B., Howell, J., Gentry, C., and Raykova, M. (2013). Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy*, pages 238–252. IEEE Computer Society Press.

[Pedersen, 1992] Pedersen, T. P. (1992). Non-interactive and information-theoretic secure verifiable secret sharing. In Feigenbaum, J., editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 129–140. Springer, Heidelberg.

[Pertsev et al., 2019] Pertsev, A., Semenov, R., and Storm, R. (2019). Tornado Cash Privacy Solution, version 1.4. `https://web.archive.org/web/20211026053443/https://tornado.cash/audits/TornadoCash_whitepaper_v1.4.pdf`.

[Pietrzak, 2019] Pietrzak, K. (2019). Simple verifiable delay functions. In Blum, A., editor, *ITCS 2019*, volume 124, pages 60:1–60:15. LIPIcs.

[Poelstra et al., 2019] Poelstra, A., Back, A., Friedenbach, M., Maxwell, G., and Wuille, P. (2019). Confidential assets. In Zohar, A., Eyal, I., Teague, V., Clark, J., Bracciali, A., Pintore, F., and Sala, M., editors, *FC 2018 Workshops*, volume 10958 of *LNCS*, pages 43–63. Springer, Heidelberg.

[PUB, 1993] PUB, N. F. (1993). Digital signature standard.

[Rackoff and Simon, 1992] Rackoff, C. and Simon, D. R. (1992). Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In Feigenbaum, J., editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 433–444. Springer, Heidelberg.

[Reid and Harrigan, 2011] Reid, F. and Harrigan, M. (2011). An analysis of anonymity in the bitcoin system. In *PASSAT/SocialCom 2011, Privacy, Security, Risk and Trust (PASSAT), 2011 IEEE Third International Conference on and 2011 IEEE Third International Conference on Social Computing (SocialCom), Boston, MA, USA, 9-11 Oct., 2011*, pages 1318–1326. IEEE Computer Society.

[Reuter and Truman, 2004] Reuter, P. and Truman, E. M. (2004). *Chasing Dirty Money: The Fight Against Money Laundering*. Peterson Institute for International Economics.

[Ron and Shamir, 2013] Ron, D. and Shamir, A. (2013). Quantitative analysis of the full Bitcoin transaction graph. In Sadeghi, A.-R., editor, *FC 2013*, volume 7859 of *LNCS*, pages 6–24. Springer, Heidelberg.

[Ruffing and Moreno-Sanchez, 2017] Ruffing, T. and Moreno-Sanchez, P. (2017). ValueShuffle: Mixing confidential transactions for comprehensive transaction privacy in bitcoin. In Brenner, M., Rohloff, K., Bonneau, J., Miller, A., Ryan, P. Y. A., Teague, V., Bracciali, A., Sala, M., Pintore, F., and Jakobsson, M., editors, *FC 2017 Workshops*, volume 10323 of *LNCS*, pages 133–154. Springer, Heidelberg.

[Ruffing et al., 2014] Ruffing, T., Moreno-Sanchez, P., and Kate, A. (2014). CoinShuffle: Practical decentralized coin mixing for bitcoin. In Kutylowski, M. and Vaidya, J., editors, *ESORICS 2014, Part II*, volume 8713 of *LNCS*, pages 345–364. Springer, Heidelberg.

[Sahai, 1999] Sahai, A. (1999). Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *40th FOCS*, pages 543–553. IEEE Computer Society Press.

[Sander and Ta-Shma, 1999] Sander, T. and Ta-Shma, A. (1999). Flow control: A new approach for anonymity control in electronic cash systems. In Franklin, M., editor, *FC'99*, volume 1648 of *LNCS*, pages 46–61. Springer, Heidelberg.

[Scafuro, 2019] Scafuro, A. (2019). Break-glass encryption. In Lin, D. and Sako, K., editors, *PKC 2019, Part II*, volume 11443 of *LNCS*, pages 34–62. Springer, Heidelberg.

[Schnorr, 1990] Schnorr, C.-P. (1990). Efficient identification and signatures for smart cards. In Brassard, G., editor, *CRYPTO'89*, volume 435 of *LNCS*, pages

239–252. Springer, Heidelberg.

[Schnorr, 1991] Schnorr, C.-P. (1991). Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174.

[Shoup, 1997] Shoup, V. (1997). Lower bounds for discrete logarithms and related problems. In Fumy, W., editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 256–266. Springer, Heidelberg.

[Shoup, 2004] Shoup, V. (2004). Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332. `https://eprint.iacr.org/2004/332`.

[Sporny et al.,] Sporny, M., Longley, D., Sabadello, M., Reed, D., Steele, O., and Allen, C. Decentralized identifiers (DIDs) v1.0 - core architecture, data model, and representations. `https://w3c-ccg.github.io/did-spec/`. Accessed: 2023-05-18.

[Steffen et al., 2022] Steffen, S., Bichsel, B., Baumgartner, R., and Vechev, M. T. (2022). ZeeStar: Private smart contracts by homomorphic encryption and zero-knowledge proofs. In *2022 IEEE Symposium on Security and Privacy*, pages 179–197. IEEE Computer Society Press.

[Steffen et al., 2019] Steffen, S., Bichsel, B., Gersbach, M., Melchior, N., Tsankov, P., and Vechev, M. T. (2019). zkay: Specifying and enforcing data privacy in smart contracts. In Cavallaro, L., Kinder, J., Wang, X., and Katz, J., editors, *ACM CCS 2019*, pages 1759–1776. ACM Press.

[Tibouchi, 2016] Tibouchi, M. (2016). CRYPTO and CHES 2016, santa barbara, CA, USA. `https://ellipticnews.wordpress.com/2016/09/02/crypto-and-ches-2016-santa-barbara-ca-usa/`. Accessed: 2023-05-16.

[Tomescu et al., 2022] Tomescu, A., Bhat, A., Applebaum, B., Abraham, I., Gueta, G., Pinkas, B., and Yanai, A. (2022). UTT: Decentralized ecash with accountable privacy. Cryptology ePrint Archive, Report 2022/452. `https://eprint.iacr.org/2022/452`.

[Valiant, 2008] Valiant, P. (2008). Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In Canetti, R., editor, *TCC 2008*, volume 4948 of *LNCS*, pages 1–18. Springer, Heidelberg.

[Werner et al., 2021] Werner, S. M., Perez, D., Gudgeon, L., Klages-Mundt, A., Harz, D., and Knottenbelt, W. J. (2021). Sok: Decentralized finance (defi). *arXiv preprint arXiv:2101.08778*.

[Wesolowski, 2019] Wesolowski, B. (2019). Efficient verifiable delay functions. In Ishai, Y. and Rijmen, V., editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 379–407. Springer, Heidelberg.

[Wood, 2016] Wood, G. (2016). Polkadot: Vision for a heterogeneous multi-chain framework. *White paper*, 21(2327):4662.

[Wood et al., 2014] Wood, G. et al. (2014). Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32.

[Wüst et al., 2019] Wüst, K., Kostiainen, K., Capkun, V., and Capkun, S. (2019). PRCash: Fast, private and regulated transactions for digital currencies. In Goldberg, I. and Moore, T., editors, *FC 2019*, volume 11598 of *LNCS*, pages 158–178. Springer, Heidelberg.

[Xu et al., 2021] Xu, J., Vavryk, N., Paruch, K., and Cousaert, S. (2021). Sok: Decentralized exchanges (dex) with automated market maker (amm) protocols. *arXiv preprint arXiv:2103.12732*.

[Yi et al., 2019] Yi, Z., Ye, H., Dai, P., Tongcheng, S., and Gelfer, V. (2019). Confidential assets on MimbleWimble. Cryptology ePrint Archive, Report 2019/1435. `https://eprint.iacr.org/2019/1435`.