DS-06-2017: Cybersecurity PPP: Cryptography

PRIViLEDGE
Privacy-Enhancing Cryptography in Distributed Ledgers

D2.4 – Revision of Privacy-Enhancing Cryptographic Primitives for Ledgers

Due date of deliverable: June 30th, 2021
Actual submission date: June 29th, 2021

Grant agreement number: 780477
Start date of project: 1 January 2018
Revision 1.0

Lead contractor: The University of Edinburgh (UEDIN)
Duration: 42 months

Project funded by the European Commission within the EU Framework Programme for Research and Innovation HORIZON 2020

Dissemination Level	
PU = Public, fully open	X
CO = Confidential, restricted under conditions set out in the Grant Agreement	
CI = Classified, information as referred to in Commission Decision 2001/844/EC	

D2.4

Revision of Privacy-Enhancing Cryptographic Primitives for Ledgers

Editor
Michele Ciampi, Markulf Kohlweiss, Mikhail Volkhov (UEDIN)

Contributors
Daniele Friolo, Ivan Visconti (UNISA)
Berry Schoenmakers, Toon Segers (TUE)
Janno Siim (UT)
Michele Ciampi, Markulf Kohlweiss, Mikhail Volkhov (UEDIN)

Reviewers
Sven Heiberg (SCCEIV)
Marko Vukolic (IBM)

June 29th, 2021
Revision 1.0

The work described in this document has been conducted within the project PRIViLEDGE, started in January 2018. This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 780477.

Executive Summary

This document presents revisions and additional improvements to the cryptographic primitives, relevant for distributed ledger technologies, that were presented earlier in the deliverable D2.3. The document contains both direct revisions of works presented earlier, such as improvements and small extensions, as well as more novel, standalone contributions, that nevertheless continue the previous line of work. The research that led to the technical contributions of the PRIViLEDGE partners has been motivated by D2.3, but also by other work packages, including toolkits. Concretely, this document contains analysis of blockchain protocols with hasty smart contract players, a new construction of non-interactive ZK argument without programming random oracles, a scheme to implement finite groups as oblivious data structures using MPC, and an analysis of security for trusted setup "ceremony" protocols for zk-SNARKs. All the mentioned contributions present or analyse basic cryptographic blocks that are commonly used in DLTs.

Contents

Chapter 1

Introduction

The research effort behind development of cryptographic primitives used in the DLT space shows a definite acceleration trend in line with popularity of DLTs. One factor that contributes to this is the activity of the blockchain industry, which is very sensitive to innovation, and incentivizes development of new cryptographic products. Another factor is that, arguably, the area as a whole is becoming more mature and professional, and thus the gap between academic experts and practitioners becomes more narrow. But paramount to all this is the intrinsic nature of such cryptographic primitives and the role of trust in the distributed protocols – these two main properties make these basic cryptographic blocks absolutely essential for most of the solutions in the area. Especially in the protocols where privacy is already a standard, each new feature or solution must account for it, and so it must rely on the developments and advances in the area of privacy-preserving cryptographic primitives.

In this document we revise the old primitives that were introduces in D2.3, but also introduce new solutions, that nevertheless continue the narrative of the previous deliverables. The following list of topics describes the outline of the document.

Hasty players in smart contracts. In Chapter 2 we revise our work on secure computation through smart contracts in forking blockchains already described in D2.3. Here we discuss some improvements and takeaways. A preliminary version of our work was included in D2.3, since then, we made some adjustments/improvements and the final version of the paper appeared in the proceedings of Financial Cryptography and Data Security 2021. This work influenced the design of the toolkit on ledger-oriented secure two/multi-party computation.

In this final version of our work, we add a threat model to show more clearly in which adversarial settings the protocols that we propose can be considered secure. Moreover, we informally propose an efficiency improvement to the (fair with penalties) compiler proposed in D2.3. Indeed, we can show how to avoid delays due to slow block confirmations by moving the deposit at the beginning of the protocol execution and adding some additional checks that the smart contract must perform. However, this requires a relaxation of the power of the assumed power of the adversary.

NIZKs without programming ROs. One of the limitations of non-interactive zero-knowledge protocols is that they require a setup to be generated honestly. In the blockchain setting, or in general, in a setting where there is not a central entity of trust, it might become tricky to generate such a setup. One common way to eliminate the setup is proving the security of the scheme in the programmable random oracle model. Unfortunately, exploiting the programmability of the random oracle in the security proof makes it difficult to prove the composability of the argument system in a setting where multiple instantiations of it are run in concurrency. The work of Canetti et al. [CJS14], for example, considers the natural scenario where multiple instantiations of different cryptographic protocols use the same random oracle. In such a scenario the requirement of programming the random oracle is unacceptable.

In Chapter 3, we show how to instantiate a non-interactive zero-knowledge protocol in the non-programmable random oracle, for the case where the simulator can run in quasi-polynomial time. This result extends the contribution related to the zero-knowledge toolkits.

Secure groups scheme and extended GCD protocol. In Chapter 4 we propose a scheme to implement finite groups as oblivious data structures. The oblivious operations are defined by a set of secure multiparty computation (MPC) protocols. Practical protocols are presented for the group of quadratic residues, elliptic curves groups and class groups of imaginary quadratic orders.

We introduce a practical protocol to calculate the extended gcd (xgcd) of two secret-shared integers adapting recent work by Bernstein and Yang [BY19a] from the p-adic setting to the finite field setting. This xgcd MPC protocol is a first and of independent interest. We apply it to implement the class group operation in MPC.

To demonstrate an application of secure groups, we extend a classical threshold cryptosystem to enable in- and output to a multiparty computation by communicating one ciphertext over an insecure channel. This is relevant in the context of PRIViLEDGE, because it permits parties to post to a bulletin board or blockchain an encrypted input, which can later be threshold decrypted by MPC parties for use in an MPC protocol.

Snarky ceremonies In Chapter 5 we present a framework and an analysis of so-called “SNARKy ceremonies”. Most zk-SNARKs require a trusted setup procedure, that is commonly done in practice using an MPC protocol, since trusting a single party for a setup is not a viable alternative in most applications. While in D2.3 we presented Sonic, an updatable and universal zk-SNARK, which tries to solve this trusted setup issue by significantly simplifying the SRS update procedure, here we focus on the more classical solution of running a distributed protocol *before* the SNARK is deployed.

The contribution analyses security of the Groth16 zk-SNARK [Gro16] in the framework that is less restrictive than the previous approaches to the problem. By doing this, we show that one such most commonly used protocol [BGM17] is indeed secure in our model. Additionally, we simplify the protocol by removing its dependence on the so-called random beacon – a sub-protocol that is usually very expensive in practice, as it is implemented using verifiable delay functions, which are by design hard to evaluate. The contribution provides a valuable insight into the modelling and practical security of the most commonly used ceremony protocol.

Additionally, implementation of the generic proof creation and proof verification that we specify in the contributions, have been implemented as part of the zero-knowledge toolkit, by the PRIViLEDGE partner GRNET. The intention is to provide an independent verification that will help to verify other ceremonies. While in theory every such MPC protocol must be verifiable post-execution, in practice this is still a task that takes certain non-trivial mental resources, since all ceremony implementations are different, and provide different verification methods. As a result, the trust in the ceremony correctness is slow to propagate, since not many users have enough motivation to validate a ceremony. Our intention is to simplify this process by providing and independent validator, thus simplifying the process of making sure a particular zk-SNARK setup is correct.

Chapter 2

Takeaways From Hasty Players in Smart Contracts with Forking Blockchains

In the previous version of our work, described in D2.3, we propose a general purpose compiler from any MPC protocol relying on point-to-point channels to an MPC protocol in which the ledger is used as a communication channel. The main feature of our protocol is that it retains efficiency even on forking blockchains, i.e. blockchains in which the last k blocks (k is also called 'chain consistency' parameter) can be different depending on the view of the miners. All those different views are branches of the chain. One of such branches will be then confirmed upon consensus (for example, by considering only the longest one). When a user issues transaction to a forking blockchain like Bitcoin or Ethereum, he has to wait that an additional number of blocks (matching with k) appears to the ledger to be sure that such transaction is indeed confirmed. This is usually done to avoid double-spending attacks.

In our work we have shown that even when the data field of such transactions contain only MPC messages, security of the underlying protocol can be compromised if the parties involved in such a protocol do not wait for message confirmation before issuing a new MPC message into the network. Instead, when a protocol is compiled with our construction, it retains security without such a constraint. More formally, we say that the compiled protocol is secure w.r.t. (ρ, σ)-hasty players[1], where ρ is the number of rounds of the compiled protocol and σ the number of messages that should be confirmed (in our case $\sigma = 0$). As an additional result, we proposed a fair with penalties extension. In this setting fairness guarantees are obtained by assuming the existence of a rational adversary whose choices are guided by monetary incentives. This extension is based on the work of [BK14]. As we will also recall in the section below, our extension requires confirmation of at least one block ($\sigma = 1$). As an important improvement, we show that we can obtain a fair with penalties blockchain-aided protocol without any message confirmation with little tweaks to the smart contract and the protocol. This holds under a reasonable relaxation of the adversarial power.

To describe more clearly the power of the adversary when running our protocols, we summarized the adversarial constraints in a separate section called "Threat model" which we describe below.

2.1 Threat Model

We assume that the blockchain adversary is computationally bounded, and when there is a fork in the blockchain, we pragmatically assume that the adversary has negligible impact on deciding which branch will be confirmed. Our generic compiler can be secure in the presence of hasty players w.r.t. dishonest majority when the protocol to be compiled is secure w.r.t. dishonest majority. We point out that if one would like to consider a very strong

[1]Note that now we are using the term "hasty" instead of "quick" , used in D2.3.

adversary with even 49% of the computational power of the network, then clearly our assumption does not hold. However, we stress that with such an adversary even the 6-block rule in Bitcoin does not make much sense. To guarantee that a delicate transaction (i.e., the coinbase transaction) is confirmed with a strong enough adversary, up to 144 blocks are necessary in Bitcoin [BW], meaning 1 day to communicate even a single protocol message. Therefore if one would like to consider such strong adversaries even a protocol requiring one confirmation might be impractical.

We will also consider adversaries mounting DoS attacks through aborts. In our context the adversary can mount this attack by causing an abort to the protocol by e.g. not playing anymore and, in our generic compiler, also by sending different messages on different branches, making honest players abort the execution. Such adversaries have the only purpose of penalizing honest players that will therefore waste time and transaction fees and perhaps restarting the protocol from scratch.

2.2 Improvements of the Generic Compiler with Fairness with Penalties

Firstly, let's informally recall how the generic compiler for blockchain-aided MPC without fairness guarantees works. The communication from P_i to P_j performed through point-to-point channels of the original MPC protocol is now emulated by making each player P_i post any message $m_{i,j}^{(r)}$ directed to P_j (for a specific round r) directly to the blockchain. To keep the message private for P_j, P_i will encrypt such message with P_j's public key. Now, P_j listens to the blockchain to receive the message $m_{i,j}^{(r)}$ (without waiting for the confirmation of the previously sent message $m_{j,i}^{(r-1)}$), and answers back with $m_{j,i}^{(r+1)}$. Now, if at any point of the execution P_i receives a message ${m'}_{i,j}^{(r)}$ that is different from $m_{i,j}^{(r)}$ received before, P_i aborts the protocol execution. To add fairness with penalties, let's consider now a protocol π' running with parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$ for a functionality f' that, given the output $y \leftarrow f(x_1, \ldots, x_n)$, where x_i is the input of player P_i, secret shares y into $(\sigma_1, \ldots, \sigma_n)$ (for a full threshold sharing scheme), generates a set of commitments $C = (\gamma_1, \ldots, \gamma_n)$ such that γ_i is the commitment of σ_i. Each player P_i obtains as an output the pair (C, σ_i)[2]. The fair protocol with penalties in the presence of hasty players can be obtained as follows: (i) We compile π' with our generic compiler, obtaining π'_{bc}. (ii) In our fair with penalties protocol π_{fair}, parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$ first engage in π'_{bc}. After π'_{bc} ends, each P_i obtains the output (C, σ_i). Now, each P_i has a limited time t_1 to send his tuple C to a smart contract together with a payment of some deposit (committing phase). (iii) If everyone sent the same tuple C, each player P_i has another time shift t_2 to send their share σ_i of γ_i to receive back their deposit. Else, if after t_2, $(\sigma_1, \ldots, \sigma_n)$ are posted to the smart contract[3]each P_i can reconstruct the output by using all collected shares. Else, players that have not opened their shares within t_2, will be penalized since their coins will remain frozen forever into the smart contract (opening phase).

The construction described above is secure against hasty players only when honest parties playing π_{fair} wait for confirmation of step (ii). The reason for requiring the confirmation of step (ii) is that otherwise the adversary can try to generate an abort during the execution of π'_{bc} after learning the output of the entire protocol π_{fair} on a different branch.

Efficiency improvement. We informally discuss how the construction described above can be modified to achieve security in the presence of $(\rho, 0)$-hasty players.

At the beginning, as in the original construction, P_1 publishes to the blockchain a smart contract containing the same functions needed to store the commitment C of the output shares needed during the committing phase and to check then their openings $(\sigma_1, \ldots, \sigma_n)$(the opening phase) . Moreover, we require that messages related to the MPC executions must be sent through this smart contract from all the participating players.

[2]P_i implicitly receives also a decommitment information of γ_i.

[3]the smart contract is published by a specific player, say P_1, before starting the protocol execution.

Moreover, the smart contract performs the following checks:

- **Opening check:** When a player does not open (or opens incorrectly) his committed value, he gets penalized by burning his deposit. This is inherited from the previous construction.
- **Fork attack check:** Whenever any player (say P_j) handles to the smart contract a message ${m'}_{i,j}^{(r)}$ for a specific MPC execution and round $r \in [\rho]$, extracted from a transaction signed by P_i, that is different from the message $m_{i,j}^{(r)}$ stored into the smart contract, the latter burns P_i's deposit. This check is needed to discourage a corrupted player P_i to provoke an abort of π'_{bc} because of a fork attack.
- **Missed message check:** When an honest player notices a message $m_{i,j}^{(r)}$ of a specific MPC execution and round r that is present in a branch but missing in another, he handles the transaction containing P_i's message $m_{i,j}^{(r)}$ to the smart contract in the branch where the message is missing. The smart contract will now store it as an MPC message for that execution. This check is needed when a corrupted player P_i double-spends the transaction containing an MPC message, therefore invalidating its appearance in some branch.

In our new construction, we require that each player send his deposit to the smart contract before starting the protocol execution and not during the committing phase as in the original construction. Recall that, in the original construction,if messages sent during the committing phase are not confirmed, a player can exploit forks to learn the output of the computation in one branch and abort the execution in another branch before handling his committed value (i.e. he aborts during π'_{bc}) without being penalized. This clearly violates fairness with penalties. We solve this issue with the missing message and fork attack checks. In fact, when a message is not appearing at all on a specific branch, honest parties will send to the smart contract all the messages of the cheating players that are missing in such a branch. With this tweak, there is no way for the adversary to learn the output in one branch and abort in another one by double spending the transaction containing the MPC message since this transaction will be sent by the honest players in any case. Now, The only way for the adversary to generate a protocol abort is by sending different messages in different branches, but this case is captured by the fork attack check.

Indeed, if there exists at least a branch where the protocol is not even started, the adversary, after learning the output in one branch, can still decide not to participate at all to the protocol in another branch by invalidating his deposit transaction. Our threat model does not capture this case. However we can reasonably relax it by assuming that it is very uncommon that an adversary can generate or find a branch where the protocol is not even started[4].

This argument applies especially to protocols with a number of rounds greater than the chain consistency parameter (e.g., In Ethereum, if a protocol requires 14 rounds, we can be sure that the first round of such a protocol is finalized by the time the committing phase starts).

DoS attacks. Note that in the original construction, deposits must be made at the end of step (ii) since adversaries trying to violate fairness can be spotted only during step (iii). Therefore an adversary can freely abort the execution before step (ii)[5]. This argument still holds when the deposit is moved at the beginning of the protocol. Even in this new construction, an adversary can still abort the protocol without compromising fairness with penalties by making an honest party abort by sending an incorrect message. Unfortunately, this event cannot be easily spotted by the smart contract itself, but only by taking as input a protocol achieving identifiable abort [IOZ14] that is publicly verifiable[6]. In this type of protocols a player cheating at any point of the execution

[4]In our first construction the need for confirming the committing phase is indeed more restrictive since it is always performed almost at the end of the protocol execution.

[5]Recall that before step (i), players run an un-fair MPC protocol π'_{bc}. At the end of such a protocol they do not learn the output, but only a share of the output that will be then committed and opened through the aid of a smart contract.

[6]An efficient construction can be found at [BOSS20].

can be successfully spotted. A modification exploiting the public verifiability of the underlying protocol can penalize aborting players even when the abort does not compromise fairness.

Chapter 3

Efficient NIZK Without Programming Random Oracles

Non-interactive zero-knowledge (NIZK) allows to prove the validity of an $\mathcal{NP}$-statement by sending just one message. In the real world, in order to obtain an efficient NIZK, the Fiat-Shamir (FS) transform is often used to convert an *efficient* constant-round public-coin honest-verifier zero-knowledge (public-coin HVZK) proof system into an efficient NIZK argument system. This approach is provable secure in the programmable random oracle and crucially require its programmability. The recent works of Lindell [TCC 2015] and Ciampi et al. [TCC 2016] proposed efficient NIZK with non-programmable random oracles along with a programmable common reference string. The last assumption, alone, is already sufficient to construct a NIZK.

In this work we show an *efficient* NIZK that relies on two assumptions that alone are insufficient to achieve NIZK (regardless of efficiency). More specifically we consider the notion of quasi-polynomial time simulation proposed by Pass in [EUROCRYPT 2003] and combine it with non-programmable random oracles. In particular, our construction combines non-interactive witness indistinguishable proofs with the concept of dense puzzles constructed by Baldimtsi et al. [ASIACRYPT 2016]. We then consider the Fischlin's transform [CRYPTO 2005] that yeld to a NIZK in the programmable random oracle with communication complexity lower compared to other works in the same setting. We prove that the Fischlin's transform retains its witness indistinguishability property in the *non*-programmable random oracle model, and that therefore can be used in our compiler to obtain a more efficient NIZK.

3.1 Introduction

A proof system allows an entity, called prover, to convince another entity, called verifier, about knowledge of some secret. Informally, a proof[1] system is *zero-knowledge* (ZK) if the prover successfully convinces the verifier without disclosing information about his secret. The notion of zero-knowledge, introduced by Goldwasser, Micali and Rackoff [GMR89], is also considered in the *non-interactive* scenario, where only the prover can speak and send just one message. This kind of proofs, introduced in [DMP87,BFM88,BDMP91], are called *Non-Interactive Zero-Knowledge (NIZK)* proofs. Unfortunately it is impossible to obtain a NIZK proof without setup assumptions and in order to overcome this impossibility result, Blum et al. [BDMP91] proposed the *Common Reference String* (CRS) Model. In this model it is assumed the existence of an honestly generated string (the exact shape of the CRS depends on the specific NIZK proof instantiation) that is given as input to both the prover and the verifier. Another setup that has been proposed in literature is the existence of registered public keys

[1]When discussing informally we will use the word proof to mean both an unconditionally sound proof and a computationally sound proof (i.e., an argument). Only in the more formal part of the contribution we will make a distinction between arguments and proofs.

in [BCNP04, DFN06, VV09, CG15].

In [FLS90, FLS99] it is showed how to obtain NIZK in the CRS model for any $\mathcal{NP}$-language in a setting where the same CRS can be reused to generate a polynomial number of proofs. Even though NIZK exists for all $\mathcal{NP}$, the candidate constructions are rather inefficient due to the $\mathcal{NP}$-reduction that needs to be computed before running the actual NIZK proof. One of the most popular approach used to obtain efficient NIZK proofs consists in taking an efficient interactive constant-round public-coin honest-verifier zero-knowledge (HVZK) proof system and making it non-interactive by replacing the role of the verifier with an hash function modelled as a random oracle [BR93] that takes as input the transcript computed so far and returns the message on the behalf of a verifier. This approach is the so called *Fiat-Shamir (FS) transform* [FS86]. To prove the security of such a transform, the ZK simulator needs to program the random oracle (i.e., the simulator decides how the RO answers to a query), in order to provide an accepting proof even though he does not know the witness (the secret) for the statement to be proved. Exploiting the programmability of the random oracle in the security proof makes difficult to prove the composability of the argument system in a setting where multiple instantiations of it are run in concurrency. The work of Canetti et al. [CJS14], for example, considers the natural scenario where multiple instantiations of different cryptographic protocols use the same random oracle. In such a scenario the requirement of programming the random oracle is clearly unacceptable.

Lindell in [Lin15] makes a step forward in order to avoid the programmability of the RO. More precisely, he provides a NIZK argument that can be proved secure assuming a non-programmable random oracle and a programmable CRS. In more details, the ZK of Lindell's protocol is proved without relying on the RO at all (though, the CRS needs to be programmed), and the soundness is proved *without* programming the RO. In a follow up work [CPSV16], the authors improve the construction of Lindell in terms of efficiency and generality under the same setup considered in [Lin15].

Another approach to go below the 3-round barrier for zero-knowledge showed in [GO94] is to allow the ZK simulator to run in quasi-polynomial time instead of expected polynomial time [Pas04b, Pas03b]. This notion, informally, implies that a malicious verifier can learn from the prover anything that can be learn by a quasi-polynomial time algorithm. As observed in [BP04] the simulator is never run by the applications that use the NIZK, therefore the notion of quasi-polynomial simulation can be sufficient for most applications of zero-knowledge, provided one is willing to make quantitatively stronger hardness assumptions. Moreover, in [Pas04b] it is showed that two rounds are necessary and sufficient for quasi-polynomial time simulatable arguments. Therefore, even tough the notion of ZK with quasi-polynomial time simulation allows to overcome some of the impossibility results with respect to standard ZK, the impossibility of obtaining NIZK argument holds also in this particular model.

Given the impossibility showed in [Pas04b] of obtaining NIZK with quasi-polynomial simulation, and the obvious impossibility of obtaining NIZK without setup assumptions in the NPRO model, one of the main question to answer is:

> *Is it possible to obtain an efficient NIZK argument with quasi-polynomial simulation in the non-programmable random oracle model?*

As a first contribution we answer affirmatively to this question providing a non-interactive zero-knowledge NIZK argument of knowledge that is secure in the non-programmable random oracle under the sole assumption that dense cryptographic puzzles exist. In more details, our protocol is proven to be *perfect* concurrent zero-knowledge (ZK) via quasi-polynomial simulation and argument of knowledge (AoK) with online extraction. Interestingly, even though we prove the ZK using quasi-polynomial simulation, the security of our NIZK argument does not rely on complexity-leveraging-type assumptions.

Our techniques. We start from any non-interactive witness-indistinguishable (WI) AoK in the NPRO model, and then we use it as a main building block together with dense cryptographic puzzles. Roughly speaking, a cryptographic puzzle is defined together with an hardness parameter g. So, if a randomly sampled puzzle is given to an adversary then she should not be able to find a solution with non-negligible probability in less than g steps. In this work we consider the notion of puzzle system of [BKZZ16] where the hardness of a puzzle holds as long as the puzzle is taken from a uniformly random distribution. In the work of Baldimtsi et al. [BKZZ16] it is also showed how to instantiate such a puzzle form standard number-theoretic assumptions such as discrete logarithm problem, and from NPROs.

In a nutshell, our NIZK combines a dense puzzle system PuzSys with a non-interactive WI argument of knowledge $\Pi^{\mathcal{WI}}$ as follows. The prover queries the random oracle with the statements x that he wants to prove thus obtaining a puzzle puz. Then the prover computes a non-interactive WI (NIWI) proof where he proves knowledge of either the witness for x or the solution of the puzzle. We observe that a malicious prover could fool the verifier by finding a solution of the puzzle and using it as a witness for the WI proof. To avoid this we just need to carefully choose the hardness factor of the puzzle in such a way that a malicious probabilistic polynomial time prover cannot solve it, but a quasi-polynomial time simulator can.

Theorem (informal). *Let Π be a sigma-protocol for the $\mathcal{NP}$ relation* Rel*, if dense cryptographic puzzles exist then there exists and efficient NIZK AoK with online extraction and straight-line quasi-polynomial time simulation in the NPRO model for* Rel.

We stress that our construction has a ZK straight-line simulator and an online AoK simulator. This yields to a protocol that can be easily composed concurrently with other cryptographic protocols [2].

To instantiate our construction we need to specify what NIWI AoK we could use. In [Pas03a] the authors propose a construction that, in combination with the OR-composition of Σ-protocol of [CDS94], would yield exactly to the tool we need.

However, in the work of Fischlin [Fis05] the authors construct a NIZK AoK with online extractor in the programmable random oracle that has has better efficiency compared to the protocol proposed in [Pas03a]. As an additional contribution of this work we prove that the Fischlin construction is a WIAoK with straight-line extraction in the *non*-programmable random oracle.

We also argue that our final construction is almost as efficient as Fischlin's construction and that can be instantiate from a large class of sigma-protocols (even larger than the class considered in [Fis05]). Indeed, in [Fis05] in order to prove the zero-knowledge property it is required that the first round of the sigma-protocol input has min-entropy superlogarithmic in the security parameter. In our approach we do not need to rely on this additional requirement.

3.2 Definitions and Tools

Preliminaries. We denote the security parameter by λ and for a finite set Q, $x \leftarrow Q$ the sampling of x from Q with uniform distribution. We use the abbreviation PPT that stays for probabilistic polynomial time. We use $\mathbb{N}$ to denote the set of all natural numbers and $\mathsf{poly}(\lambda)(\cdot)$ to indicate a generic polynomial function. A *polynomial-time $\mathcal{NP}$-relation* Rel (or *$\mathcal{NP}$-relation*, in short) is a subset of $\{0,1\}^* \times \{0,1\}^*$ such that membership of (x,w) in Rel can be decided in time polynomial in $|x|$. For $(x,w) \in \mathsf{Rel}$, we call x the *instance* and w a *witness* for x. For a polynomial-time relation Rel, we define the $\mathcal{NP}$-language L_{Rel} as $L_{\mathsf{Rel}} = \{x | \exists\ w : (x,w) \in \mathsf{Rel}\}$. Analogously, unless otherwise specified, for an $\mathcal{NP}$-language L we denote by $\mathsf{Rel_L}$ the corresponding polynomial-time relation (that is, $\mathsf{Rel_L}$ is such that $L = L_{\mathsf{Rel_L}}$). We also use $\hat{L}$ to denote the language that includes L and all well formed instances that are not in L. Let A and B be two interactive probabilistic algorithms. We denote by $\langle A(\alpha), B(\beta)\rangle(\gamma)$ the distribution of B's output after running on private input β with A using private input α, both running on common input γ. A *transcript* of $\langle A(\alpha), B(\beta)\rangle(\gamma)$ consists of the messages exchanged during

[2]Since our simulator does not run in expected polynomial time, it is not possible to prove results related to the Universal Composable (UC) setting [Can01]. We observe that the property of our protocol of being concurrently composable is still meaningful as showed in [BS05] where general concurrent composition with superpolynomial-time computation is considered.

an execution where A receives a private input α, B receives a private input β and both A and B receive a common input γ. Moreover, we will refer to the *view* of A (resp. B) as the messages it received during the execution of $\langle A(\alpha), B(\beta)\rangle(\gamma)$, along with its randomness and its input. A function $\nu(\cdot)$ from non-negative integers to reals is called negligible, if for every constant $c > 0$ and all sufficiently large $\lambda \in \mathbb{N}$ we have $\nu(\lambda) < \lambda^{-c}$.

3.2.1 Argument Systems

Here we recall the notions of completeness and online extraction provided in [Fis05]. A pair $\Pi = (\mathsf{P}, \mathsf{V})$ of probabilistic polynomial-time algorithms is called a non-interactive argument of knowledge for the $\mathcal{NP}$-relation $\mathsf{Rel_L}$ with an online extractor (in the non-programmable random oracle model) if the following holds.

Completeness. For any non-programmable random oracle $\mathcal{O}$, any $(x, w) \in \mathsf{Rel_L}$ and any $\pi \leftarrow \mathsf{P}^{\mathcal{O}}(x, w)$ we have

$$\Pr\big[\mathsf{V}^{\mathcal{O}}(x, \pi) = 1\big] = 1 - \nu(|x|).$$

Argument of Knowledge with Online Extractor. There exists a probabilistic polynomial-time algorithm Ext, the AoK online extractor, such that the following holds for any PPT algorithm $\mathcal{A}$. Let $\mathcal{O}$ be a non-programmable random oracle, $(x, \pi) \leftarrow \mathcal{A}^{\mathcal{O}}(\lambda)$ and $\mathcal{Q}_{\mathcal{O}}(\mathcal{A})$ be the sequence of queries of $\mathcal{A}$ to $\mathcal{O}$ and $\mathcal{O}$'s answers. Let $w \leftarrow \mathsf{Ext}(x, \pi, \mathcal{Q}_{\mathcal{O}}(\mathcal{A}))$. Then there exists a negligible function ν such that

$$\Pr\big[(x, w) \notin \mathsf{Rel} \text{ and } \mathsf{V}^{\mathcal{O}}(x, \pi) = 1\big] \leq \nu(\lambda).$$

Not to overburden the descriptions of protocols and simulators, we omit to specify that the parties have access to the non-programmable random oracle $\mathcal{O}$ when it is clear from the context.

Quasi-Polynomial Time Simulation. Since the verifier in an interactive argument is often modeled as a PPT machine, the classical zero-knowledge definition requires that the simulator runs also in (expected) polynomial time. In [Pas03b], the simulator is allowed to run in time $\lambda^{\mathtt{poly}(\mathtt{log}(\lambda))}$. Loosely speaking, we say that an interactive argument is $\lambda^{\mathtt{poly}(\mathtt{log}(\lambda))}$-perfectly simulatable if for any adversarial verifier there exists a simulator running in time $\lambda^{\mathtt{poly}(\mathtt{log}(\lambda))}$, where λ is the size of the statement being proved, whose output is identically distributed to the output of the adversarial verifier.

Definition 1 (straight-line $T(\lambda)$ simulatability, Def. 31 of [Pas04a]). Let $T(\lambda)$ be a class of functions that is closed under composition with any polynomial. We say that an interactive argument (proof) $(\mathcal{P}, \mathcal{V})$ for the language $L \in \mathcal{NP}$, with the witness relation $\mathsf{Rel_L}$, is straight-line $T(\lambda)$-simulatable if for every PPT machine $\mathcal{V}^\star$ there exists a probabilistic simulator S with running time bounded by $T(\lambda)$ such that the following two ensembles are computationally indistinguishable (when the distinguish gap is a function in $\lambda = |x|$)

- $\{(\langle \mathsf{P}(w), \mathcal{V}^\star(z)\rangle(x))\}_{z\in\{0,1\}^*, x\in L}$ for arbitrary w s.t. $(x, w) \in \mathsf{Rel_L}$
- $\{(\langle S, \mathcal{V}^\star(z)\rangle(x))\}_{z\in\{0,1\}^*, x\in L}$

The following theorem shows the importance of straight-line $\lambda^{\mathtt{poly}(\mathtt{log}(\lambda))}$-perfect simulatability by connecting it to concurrent composition of arguments.

Theorem 1 ([Pas04a]). *If the argument system* $\Pi = (\mathsf{P}, \mathsf{V})$ *is straight-line* $\lambda^{\mathtt{poly}(\mathtt{log}(\lambda))}$*-simulatable then it is also straight-line concurrent* $\lambda^{\mathtt{poly}(\mathtt{log}(\lambda))}$*-simulatable.*

We also consider the notion of *perfect straight-line simulation*. This is equal to the Definition 1 with the difference that the malicious adversary can be unbounded instead of being PPT and that the two ensembles (the simulated execution and the real execution) are identically distributed (See Definition 30 of [Pas04a]).

3.2.2 Cryptographic Puzzles

In [BKZZ16] the authors introduce a new class of prover verifier protocol called Proof of Work or Knowledge (PoWorK). Moreover, in order to formalize PoWorK, the authors give the notion of *puzzle system*. A puzzle system PuzSys is a tuple of algorithms $\mathsf{PuzSys} = (\xleftarrow{\$}, \mathsf{Solve}, \mathsf{Verify})$ that are defined in the following way. $\xleftarrow{\$}$ on input the security parameter 1^λ and the hardness factor h outputs a puzzle $\mathtt{puz}$; Solve on input the security parameter 1^λ, a hardness factor h and a puzzle instance $\mathtt{puz}$ outputs a potential solution $\mathtt{sol}$; Verify on input the security parameter 1^λ, a hardness factor h a puzzle instance $\mathtt{puz}$, and a potential solution $\mathtt{sol}$ outputs 0 or 1.

Moreover, while the algorithms $\xleftarrow{\$}$ and Verify are efficient, it is difficult to compute a solution for a sampled puzzle. More precisely, a puzzle system is g-hard if the Solve algorithm can not take less of g steps of computation. The authors of [BKZZ16] propose also a stronger notion of puzzle that enjoys the property of *dense samplability*. That is, the puzzles can be sampled by just generating random strings (i.e. the puzzle instances should be dense over $\{0,1\}^{\ell(h,\lambda)}$). In our work we consider the same notion of puzzle system with dense samplability of [BKZZ16]. We remark that the notion of dense samplable puzzles considered in [BKZZ16] is equipped with an additional efficient algorithm that generates a puzzle together with its solution, but we do not need this additional requirement in our work.

We denote the puzzle space as $\mathcal{PS}_\lambda$, the solution space as $\mathcal{SS}_\lambda$, and the hardness space as $\mathcal{HS}_\lambda$.

Definition 2. A Dense Samplable Puzzle (DSP) system $\mathsf{PuzSys} = (\xleftarrow{\$}, \mathsf{Solve}, \mathsf{Verify})$ enjoys the following properties.

Completeness. A puzzle system PuzSys is complete, if for every h in the hardness space $\mathcal{HS}_\lambda$:

$$\Pr\Big[\mathtt{puz} \leftarrow \xleftarrow{\$}(1^\lambda, h), \mathtt{sol} \leftarrow \mathsf{Solve}(1^\lambda, h, \mathtt{puz}) : \mathsf{Verify}(1^\lambda, h, \mathtt{puz}, \mathtt{sol}) = 0\Big] \leq \nu(\lambda)$$

The number of steps that Solve takes to run is monotonically decreasing in the hardness factor h and may exponentially depend on λ, while Verify and Solve run in time polynomial in λ.

g-Hardness. Let $\mathsf{Steps}_B(\cdot)$ be the number of steps (i.e. machine/operation cycles) executed by algorithm B. We say that a puzzle system PuzSys is g-hard for some function g, if for every adversary $\mathcal{A}$, for every auxiliary tape $z \in \{0,1\}^\star$ and for every $h \in \mathcal{HS}_\lambda$ then there exists a negligible function ν such that:

$$\begin{aligned}\mathrm{Prob}[\mathtt{puz} \leftarrow \xleftarrow{\$}(1^\lambda, h), \mathtt{sol} \leftarrow \mathcal{A}(1^\lambda, z, \mathtt{puz}) :&\\ \mathsf{Verify}(1^\lambda, h, \mathtt{puz}, \mathtt{sol}) = 1 \wedge&\\ \mathsf{Steps}_{\mathcal{A}}(z, 1^\lambda, h, \mathtt{puz}) \leq g(\mathsf{Steps}_{\mathsf{Solve}}(1^\lambda, h, \mathtt{puz}))] \leq \nu(\lambda)&\end{aligned}$$

Dense Puzzles. Given function ℓ of λ and h, there exists a negligible function ν such that

$$\Delta[\xleftarrow{\$}(1^\lambda, h), \mathsf{U}_{\ell(\lambda,h)})] \leq \nu(\lambda)$$

where $\mathsf{U}_{\ell(\lambda,h)}$ stands for the uniform distribution over $\{0,1\}^{\ell(\lambda,h)}$.

We observe that the properties of density and g-hardness imply that for every adversary $\mathcal{A}$, for every auxiliary tape $z \in \{0,1\}^\star$ and for every $h \in \mathcal{HS}_\lambda$ there exists a negligible function ν such that

$$\begin{aligned}\mathrm{Prob}[\mathtt{sol} \leftarrow \mathcal{A}(z, 1^\lambda, \eta) : \eta \leftarrow \{0,1\}^{\ell(\lambda,h)} \wedge \mathsf{Verify}(1^\lambda, h, \eta, \mathtt{sol}) = 1 \wedge&\\ \mathsf{Steps}_{\mathcal{A}}(z, 1^\lambda, h, \eta) \leq g(\mathsf{Steps}_{\mathsf{Solve}}(1^\lambda, h, \eta))] \leq \nu(\lambda).&\end{aligned}$$

3.2.3 Witness Indistinguishability.

To formalize the notion of WI we consider a game $\mathsf{ExpAWI}^b_{\Pi,\mathcal{A}}$ between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$ in which the instance x and two witnesses w_0 and w_1 for x are chosen by $\mathcal{A}$. The challenger, upon receiving (x, w_0, w_1) starts interacting with $\mathcal{A}$ accordingly to the prover procedure of Π using w_b as a witness. The adversary wins the game if she can guess which of the two witnesses was used by the challenger.

We now formally define the WI experiment $\mathsf{ExpAWI}^b_{\Pi,\mathcal{A}}(\lambda, \zeta)$. This experiment is parameterized by a protocol $\Pi = (\mathsf{P}, \mathsf{V})$ for an $\mathcal{NP}$-relation Rel and by a PPT adversary $\mathcal{A}$. The experiment has as input the security parameter λ and auxiliary information ζ for $\mathcal{A}$.

$\mathsf{ExpAWI}^b_{\Pi,\mathcal{A}}(\lambda, \zeta)$:

1. $\mathcal{A}$ picks an instance x, witnesses w_0 and w_1 such that $(x, w_0), (x, w_1) \in \mathsf{Rel}$, and sends (x, w_0, w_1) to $\mathcal{C}$.
2. $\mathcal{C}$ interacts with $\mathcal{A}$ as P would do using the witness w_b.
3. In the end of the interaction with the challenger $\mathcal{C}$, $\mathcal{A}$ outputs $b' \in \{0, 1\}$.

Definition 3 (Witness Indistinguishability)**.** A protocol Π is *WI* if for every PPT adversary $\mathcal{A}$ there exists a negligible function ν such that for any $\zeta \in \{0,1\}^*$ it holds that

$$\left|\Pr\left[\mathsf{ExpAWI}^0_{\Pi,\mathcal{A}}(\lambda, \zeta) = 1\right] - \Pr\left[\mathsf{ExpAWI}^1_{\Pi,\mathcal{A}}(\lambda, \zeta) = 1\right]\right| \leq \nu(\lambda).$$

We also consider the notion of *statistical* and *perfect* WI. In the case of statistical WI the definition is equal to the one proposed above with the difference that $\mathcal{A}$ it is not restricted to be PPT. The perfect WI notion is like the statistical one but with $\nu(\lambda) = 0$.

3.2.4 Sigma-Protocols

A Sigma-Protocol $\Pi = (\mathcal{P}, \mathcal{V})$ is a 3-round public-coin protocol. An execution of Π proceeds with the following 3 moves:

1. The prover $\mathcal{P}$ computes the first message using as input the instance to be proved x with the corresponding witness w, and outputs the first message a with an auxiliary information $\mathtt{aux}$ (we denote this action with $(a, \mathtt{aux}) \leftarrow \mathcal{P}(x, w)$).
2. The verifier $\mathcal{V}$ upon receiving a, compute and sends a random string $c \leftarrow \{0, 1\}^l$ with $l \in \mathbb{N}$.
3. $\mathcal{P}$ on input c and $\mathtt{aux}$ computes and sends z to $\mathcal{V}$ (we denote this action with $z \leftarrow \mathcal{P}(\mathtt{aux}, c)$).
4. $\mathcal{V}$, on input (x, a, c, z) outputs 1 to accept, 0 to reject (we denote this action with $\mathcal{V}(x, a, c, z) = b$ where $b \in \{0, 1\}$ denotes whether $\mathcal{V}$ accepts or not).

Definition 4 (Sigma-protocol [CDS94])**.** A 3-move protocol Π with challenge length $l \in \mathbb{N}$ is a *sigma-protocol* for a relation Rel if it enjoys the following properties:

1. **Completeness**. If $(x, w) \in \mathsf{Rel}$ then all honest 3-move transcripts for (x, w) are accepting.
2. **Special Soundness**. There exists an efficient algorithm $\mathsf{Extract}$ that, on input two accepting transcripts for x (a, c, z) and (a, c', z') with $c' \neq c$ outputs a witness w such that $(x, w) \in \mathsf{Rel}$.
3. **Special Honest-Verifier Zero Knowledge (SHVZK)**. There exists a PPT simulator algorithm Sim that takes as input $x \in L_{\mathsf{Rel}}$, security parameter 1^λ and $c \in \{0, 1\}^l$ and outputs an accepting transcript for x where c is the challenge (we denote this action with $(a, z) \leftarrow \mathsf{Sim}(x, c)$). Moreover, for all l-bit strings c, the distribution of the output of the simulator on input (x, c) is computationally indistinguishable from the

distribution of the 3-move honest transcript obtained when $\mathcal{V}$ sends c as challenge and $\mathcal{P}$ runs on common input x and any private input w such that $(x, w) \in \mathsf{Rel}$.

We say that Π is *statistical* when the two distributions are statistically close and perfect when the two distributions are identical.

Following [Fis05] we require an additional property with respect to the classical notion of sigma-protocol. That is, we require that the prover's third round is *quasi-unique*, i.e., it should be infeasible to find another valid third round to a proof (a, c, z), even if one knows the witness. As noted in [Fis05], this property holds for example if the third round z is uniquely determined by x, a, and c as for the protocols by Guillou-Quisquater [GQ88] and Schnorr [Sch89]. More in general, this property holds for the class of sigma-protocol for proving the knowledge of a preimage of a group homomorphism defined in [CD98, Mau15]. A formalisation of this property follows.

Definition 5. A (perfect/statistical) sigma-protocol $\Pi = (\mathcal{P}, \mathcal{V})$ has *quasi-unique third round* if for any probabilistic polynomial-time algorithm $\mathcal{A}$, for parameter λ and (x, a, c, z, z') we have that there exists a negligible function ν such that

$$\text{Prob}\left[\ \mathcal{V}(x, a, c, z) = \mathcal{V}(x, a, c, z') = 1 \text{ and } z \neq z'\ \right] \leq \nu(\lambda).$$

3.2.5 Or-Composition of Sigma-Protocols

In this section we recall the or-composition of sigma-protocols proposed in [CDS94]. Let $\Pi_0 = (\mathcal{P}_0, \mathcal{V}_0)$ be a sigma-protocol for the $\mathcal{NP}$-relation Rel_0 and $\Pi_1 = (\mathcal{P}_1, \mathcal{V}_1)$ be a sigma-protocol for the $\mathcal{NP}$-relation Rel_1. Moreover, let Sim_0 be the Special HVZK simulator for Π_0 and Sim_1 be the Special HVZK simulator for Π_1.

We consider the following 3-round public coin protocol $\Pi^{\mathtt{OR}} = (\mathcal{P}^{\mathtt{OR}}, \mathcal{V}^{\mathtt{OR}})$ where $\mathcal{P}^{\mathtt{OR}}$ and $\mathcal{V}^{\mathtt{OR}}$ has a common input (x_0, x_1) where $x_0 \in \hat{L}_0$ and $x_1 \in \hat{L}_1$. $\mathcal{P}^{\mathtt{OR}}$ has a private input w_b with $b \in \{0, 1\}$ and $(x_b, w_b) \in \mathsf{Rel}_b$.

1. The prover $\mathcal{P}^{\mathtt{OR}}$ picks $c_{1-b} \leftarrow \{0, 1\}^l$ and computes $(a_{1-b}, z_{1-b}) \leftarrow \mathsf{Sim}_{1-b}(x_{1-b}, c_{1-b})$ with $l \in \mathbb{N}$. Then $\mathcal{P}^{\mathtt{OR}}$ computes $(\mathtt{aux}, a_b) \leftarrow \mathcal{P}_b(x_b, w_b)$ and send (a_0, a_1) to $\mathcal{V}^{\mathtt{OR}}$.
2. The verifier $\mathcal{V}^{\mathtt{OR}}$ upon receiving (a_0, a_1), computes and sends a random string $c \leftarrow \{0, 1\}^l$.
3. $\mathcal{P}^{\mathtt{OR}}$, upon receiving c computes $c_b = c \oplus c_{1-b}$ and and $(z_b) \leftarrow \mathcal{P}_b(\mathtt{aux}, c_b)$ and sends (c_0, z_0, c_1, z_1) to $\mathcal{V}^{\mathtt{OR}}$.
4. $\mathcal{V}^{\mathtt{OR}}$, upon receiving (z_0, z_1) checks if $\mathcal{V}_0(x_0, a_0, c_0, z_0) = 1$ and $\mathcal{V}_1(x_1, a_1, c_1, z_1) = 1$ and $c = c_0 \oplus c_1$. If it is, then $\mathcal{V}^{\mathtt{OR}}$ outputs 1, 0 otherwise.

Theorem 2 ([CDS94, GMY06]). *Let Π_0 be a (perfect/statistical) sigma-protocol for the $\mathcal{NP}$-relation Rel_0 and Π_1 be a (perfect) sigma-protocol for the $\mathcal{NP}$-relation Rel_1 then $\Pi^{\mathtt{OR}}$ is a sigma-protocol that is WI for relation*

$$\mathsf{Rel}_{\mathtt{OR}} = \Big\{((x_0, x_1), w) : \big((x_0, w) \in \mathsf{Rel}_{L_0} \wedge x_1 \in L_1\big) \vee \big((x_1, w) \in \mathsf{Rel}_{L_1} \wedge x_0 \in L_0\big)\Big\}.$$

We recall the above theorem just for completeness even though in this chapter we use $\Pi^{\mathtt{OR}}$ in a non-blackbox way and we do not rely on its WI property directly. We just find convenient to use the prover and the verifier of $\Pi^{\mathtt{OR}}$ to shorten and make more clear the description of the protocols proposed in this chapter. Only in the security proof we make non-black box use of $\Pi^{\mathtt{OR}}$ in order to rely on the security of the underling sigma-protocols Π_0 and Π_1.

3.3 Fischlin's NIZK

In [Fis05] the authors provide a NIZK AoK Π^{Fishlin} where the AoK extractor relies only on the *observability* of the RO[3]. The main advantage of Fischlin's construction is that the AoK extractor does not need to rewind the malicious prover in order to extract the witness. Following [Fis05], we refer to such an extractor as *online extractor*. As remarked in [Fis05], the arguments of knowledge with online extractors are especially suitable for settings with concurrent executions. Indeed, Fischlin shows an example in which the NIZK with *standard* knowledge extractors cannot be used due to the rewinds that has to be done, and shows how overcome this limitation using online extractor. Also, Fischlin shows how to enhance the security of the Boneh et al. group signature scheme [BBS04] using a NIZK argument of knowledge with online extraction.

Our construction revisits Π^{Fishlin} by maintaining the AoK online extractor and completeness of Π^{Fishlin} but provides ZK with qualsi-polynomial time simulation. That is, we prove the ZK of our construction relying on quasi-polynomial time simulation whereas Π^{Fishlin} is proved to be ZK in the PRO. In this section we recall the NIZK construction proposed [Fis05] and then we show how to bootstrap Π^{Fishlin} to a NIZK with qualsi-polynomial time simulation. We refer the reader to Fig. 3.1 for the formal description of Π^{Fishlin}.

Let $\Pi = (\mathsf{P}, \mathsf{V})$ be a sigma-protocol with quasi-unique third round and challenge length $\ell = O(\log \lambda)$ for the $\mathcal{NP}$-relation $\mathsf{Rel_L}$. Define the parameters b, r, S, t (as functions of λ) such that $br = \omega(\log \lambda)$, $2^{t-b} = \omega(\log \lambda)$, $b, r, t = O(\log \lambda)$, $S = O(r)$ and $b \leq t \leq \ell$. Define the following non-interactive argument system $\Pi^{\mathsf{Fishlin}} = (\mathsf{P}^{\mathsf{Fischlin}}, \mathsf{V}^{\mathsf{Fischlin}})$ for relation $\mathsf{Rel_L}$ in the random oracle model, where the random oracle $\mathcal{O}$ maps to b bits.
Common input: security parameter λ, $\mathcal{NP}$-statement $x \in L$, the parameters b, r, S, t as defined above.
Input to $\mathcal{P}^{\mathcal{WI}}$: w s.t. $(x, w) \in \mathsf{Rel_L}$.
Proof. The prover $\mathsf{P}^{\mathsf{Fischlin}}$ executes the following steps.

1. On input (x, w) runs r times P (each time using fresh randomness) on input (x, w) thus obtaining $((\mathtt{aux}_1, a_1), (\mathtt{aux}_2, a_2), \dots, (\mathtt{aux}_r, \dots, a_r))$ Let $A = (a_1, a_2, \dots, a_r)$.
2. For $i = 1, \dots, r$
   Pick $c_i \in \{0,1\}^t$ such that $\mathcal{O}(x, A, i, c_i, z_i) = 0^b$ where z_i is the output of P on input $(\mathtt{aux}_i, c_i)$.
   If such a c_i does not exist, then pick the first one for which the output of the evaluation of $\mathcal{O}$ is minimal among all 2^t hash values.
3. Output $\pi = (\{a_i, c_i, z_i\}_{i=1,\dots,r})$.

Verification. The verifier $\mathsf{V}^{\mathsf{Fischlin}}$ on input x and $\pi = (\{a_i, c_i, z_i\}_{i=1,\dots,r})$ accepts if and only if for $i = 1, \dots, r$, $\mathsf{V}(x, a_i, c_i, z_i) = 1$ and $\Sigma_{i=1}^{r} \mathcal{O}(x, A, i, c_i, z_i) \leq S$.

Figure 3.1: Fischlin's Straight-line NIZK AoK

As remarked in [Fis05], this protocol has a small completeness error. For deterministic verifiers this error can be

[3] We observe that even though the authors of [Fis05] talks about Proof of Knowledge, they still need to polynomially bound the number of queries that an adversary can make to the random oracle. To avoid any ambiguity, in this work we consider only the notion of AoK since the malicious prover is implicitly bounded by the number of queries that can be made to the RO.

removed in principle by standard techniques, namely, by letting the prover check on behalf of the verifier that the proof is valid before outputting it; if not the prover simply sends the witness to the verifier. In practice, in case of this very unlikely event, the prover may just compute a proof from scratch. Here we recall Theorem 2 of [Fis05].

Theorem 2 of [Fis05]. Let Π be a sigma-protocol for the $\mathcal{NP}$-relation Rel with quasi-unique third round and where the first round of Π has min-entropy superlogarithmic in the security parameter λ, then Π^{Fishlin} is a non-interactive zero-knowledge proof of knowledge for the $\mathcal{NP}$-relation Rel (in the programmable random oracle model) with online extractor.

3.3.1 Efficiency

Following the analysis given in [Fis05], we set $b = 9$, $t = 12$, $r = 10$ and $S = 10$ thus obtaining an online-extractor that fails with probability at most $\mathcal{Q}2^{-72}$ where $\mathcal{Q}$ is the maximal number of hash queries (assuming that finding distinct responses is beyond feasibility). Then the total number of hash function evaluations is roughly $2^9 r$ and the number of executions of the underlying sigma-protocol Π is 10.

3.4 Our Results

3.4.1 Our NIZK AoK

In order to construct our NIZK with quasi-polynomial simulation for the $\mathcal{NP}$ relation $\mathsf{Rel_L}$ $\Pi^{\mathsf{NIZK}} = (\mathsf{P}^{\mathsf{NIZK}}, \mathsf{V}^{\mathsf{NIZK}})$, we make use of the following tools.

- A dense samplable puzzle system $\mathsf{PuzSys} = (\leftarrow\!\$\,, \mathsf{Solve}, \mathsf{Verify})$ such that for every hardness factor $h \in \mathcal{HS}_\lambda$ there exists a negligible function ν such that the following holds:
  1. $\Pr\big[\mathsf{puz} \leftarrow\ \leftarrow\!\$\,(1^\lambda, h) : g(\mathsf{Steps}_{\mathsf{Solve}}(1^\lambda, h, \mathsf{puz})) \le \lambda^{\log\lambda}\big] \le \nu(\lambda)$;
  2. the worst-case running time of $\mathsf{Solve}(1^\lambda, h, \cdot)$ is $\lambda^{\mathsf{poly}(\lambda)(\log\lambda)}$.[4]
- $\Pi^{\mathcal{WI}} = (\mathcal{P}^{\mathcal{WI}}, \mathcal{V}^{\mathcal{WI}})$: a non-interactive perfect/statistical WI AoK with online extractor for the $\mathcal{NP}$-relation $\mathsf{Rel}_{\mathcal{WI}} = \{((x, \mathsf{puz}), w) : (x, w) \in \mathsf{Rel_L} \text{ or } \mathsf{Verify}(1^\lambda, h, \mathsf{puz}, w) = 1\}$.

$\mathsf{P}^{\mathsf{NIZK}}$ and $\mathsf{V}^{\mathsf{NIZK}}$ have also access to a NPRO $\mathcal{O} : \{0,1\}^* \to \{0,1\}^{\ell(\lambda,h)}$ where $\ell(\lambda, h)$ is a function of the security and the hardness parameters of PuzSys. We need to relate the output length of the random oracle to the parameters of PuzSys because $\mathcal{O}$ is used to generate a puzzle in our construction. More details are given in the security proof.

Theorem 3. *If $\Pi^{\mathcal{WI}} = (\mathcal{P}^{\mathcal{WI}}, \mathcal{V}^{\mathcal{WI}})$ is a non-interactive perfect/statistical WI AoK with online extractor for the $\mathcal{NP}$-relation $\mathsf{Rel}_{\mathcal{WI}}$ that is secure in the NPRO model and* PuzSys *is a dense samplable puzzle system according to Definition 2, then Π^{NIZK} is a straight-line concurrent perfectly/statistically $\lambda^{\mathsf{poly}(\lambda)(\log\lambda)}$-simulatable argument of knowledge with online extraction in the NPRO model.*

Proof. **Completeness.** The completeness follows immediately from the completeness of $\Pi^{\mathcal{WI}}$ and PuzSys.

Quasi-polynomial time perfect/statistical simulation. Let $\mathsf{V}^{\mathsf{NIZK}}$ be an arbitrary verifier. We construct a simulator Sim that runs in $\lambda^{\mathsf{poly}(\lambda)(\log\lambda)}$ time, such that the distributions

$\Big\{(\langle \mathsf{P}^{\mathsf{NIZK}}(w), \mathsf{V}^{\mathsf{NIZK}\star}(z)\rangle(x))\Big\}_{z\in\{0,1\}^*, x\in L}$ for arbitrary w s.t. $(x, w) \in \mathsf{Rel_L}$ and

$\Big\{(\langle \mathsf{Sim}, \mathsf{V}^{\mathsf{NIZK}\star}(z)\rangle(x))\Big\}_{z\in\{0,1\}^*, x\in L}$

[4]This is the same puzzle used in Theorem 7 of [BKZZ16].

Common input: security parameter λ, $\mathcal{NP}$-statement $x \in L$.
Input to $\mathsf{P}^{\mathsf{NIZK}}$: w s.t. $(x, w) \in \mathsf{Rel_L}$.

Proof. $\mathsf{P}^{\mathsf{NIZK}}$ computes the puzzle $\mathtt{puz}$ for PuzSys by querying the random oracle $\mathcal{O}$ on input x: $\mathtt{puz} \leftarrow \mathcal{O}(x)$. $\mathsf{P}^{\mathsf{NIZK}}$ defines $x_{\mathcal{WI}} = (x, \mathtt{puz})$, $w_{\mathcal{WI}} = w$ and runs $\mathcal{P}^{\mathcal{WI}}$ on input $(x_{\mathcal{WI}}, w_{\mathcal{WI}})$ thus obtaining $\pi_{\mathcal{WI}}$ which is sent to $\mathsf{V}^{\mathsf{NIZK}}$.
Verification. $\mathsf{V}^{\mathsf{NIZK}}$ queries $\mathcal{O}$ with x thus obtaining $\mathtt{puz}$ and defines $x_{\mathcal{WI}} = (x, \mathtt{puz})$. $\mathsf{V}^{\mathsf{NIZK}}$ now runs $\mathcal{V}^{\mathcal{WI}}$ on input $(x_{\mathcal{WI}}, \pi_{\mathcal{WI}})$ and outputs what $\mathcal{V}^{\mathcal{WI}}$ outputs.

Figure 3.2: Our NIZK AoK protocol

are perfectly/statistical indistinguishable.

The simulator Sim is described in Fig. 3.3. The only observations that are required to complete this part of the proof are that Sim can compute the solution of the puzzle $\mathtt{puz}$ in time $\lambda^{\mathsf{poly}(\lambda)(\log \lambda)}$ and that $\Pi^{\mathcal{WI}}$ is perfect/statistical witness-indistinguishable.

```
$\mathsf{Sim}(1^\lambda, x)$
    - Compute the puzzle $\mathtt{puz}$ for $\mathsf{PuzSys}$ by querying the random
      oracle $\mathcal{O}$ on input $x$: $\mathtt{puz} \leftarrow \mathcal{O}(x)$ and compute $\mathtt{sol} \leftarrow$
      $\mathsf{Solve}(1^\lambda, h, \mathtt{puz})$.
    - Define $x_{\mathcal{WI}} = (x, \mathtt{puz})$, $w_{\mathcal{WI}} = \mathtt{sol}$ and run $\mathcal{P}^{\mathcal{WI}}$ on input
      $(x_{\mathcal{WI}}, w_{\mathcal{WI}})$ thus obtaining $\pi_{\mathcal{WI}}$.
    - Send $\pi_{\mathcal{WI}}$ to $\mathsf{V}^{\mathsf{NIZK}\star}$ and output what $\mathsf{V}^{\mathsf{NIZK}\star}$ outputs.
```

Figure 3.3: Quasi-polynomial time simulator Sim.

Argument of knowledge. By assumption there exist a PPT AoK online extractor Ext for $\Pi^{\mathcal{WI}}$ such that the following holds for any PPT algorithm $\mathcal{P}^{\mathcal{WI}\star}$. Let $\mathcal{O}$ be a random oracle, $(x_{\mathcal{WI}}, \pi_{\mathcal{WI}}) \leftarrow \mathcal{P}^{\mathcal{WI}\star}(\lambda)$ and $\mathcal{Q}_{\mathcal{O}}(\mathcal{P}^{\mathcal{WI}\star})$ be the sequence of queries of $\mathcal{P}^{\mathcal{WI}\star}$ to $\mathcal{O}$ and $\mathcal{O}$'s answers. Let $w_{\mathcal{WI}} \leftarrow \mathsf{Ext}(x, \pi, \mathcal{Q}_{\mathcal{O}}(\mathcal{P}^{\mathcal{WI}\star}))$. Then there exists a negligible function ν such that

$$\Pr\left[(x_{\mathcal{WI}}, w_{\mathcal{WI}}) \notin \mathsf{Rel}_{\mathcal{WI}} \text{ and } \mathcal{V}^{\mathcal{WI}}(x_{\mathcal{WI}}, \pi_{\mathcal{WI}}) = 1\right] \leq \nu(\lambda).$$

The AoK PPT extractor $\mathsf{E}^{\mathsf{NIZK}}$ for Π^{NIZK} simply internally runs Ext and outputs what Ext would output. We now observe that $\mathsf{E}^{\mathsf{NIZK}}$ could fail only because the output of the internal extractor Ext is a solution $\mathtt{sol}$ for $\mathtt{puz} = \mathcal{O}(x)$.[5] Let us assume by contradiction that this happens. That is, let $\mathcal{O}$ be a random oracle, $(x, \pi_{\mathsf{NIZK}}) \leftarrow \mathsf{P}^{\mathsf{NIZK}\star}(\lambda)$, $\mathcal{Q}_{\mathcal{O}}(\mathsf{P}^{\mathsf{NIZK}\star})$ be the sequence of queries of $\mathsf{P}^{\mathsf{NIZK}\star}$ to $\mathcal{O}$ and $\mathcal{O}$'s answers and let $\eta \leftarrow \mathsf{Ext}(x, \pi, \mathcal{Q}_{\mathcal{O}}(\mathsf{P}^{\mathsf{NIZK}\star}))$, then

$$\Pr\left[\mathsf{Verify}(1^\lambda, h, \mathcal{O}(x), \eta) = 1 \text{ and } \mathsf{V}^{\mathsf{NIZK}}(x, \pi_{\mathsf{NIZK}}) = 1\right] = \delta(\lambda).$$

But this would contradict the g-hardness of PuzSys. Indeed the extractor $\mathsf{E}^{\mathsf{NIZK}}$ outputs in PPT a solution to a puzzle sampled from an uniform distribution (since $\mathcal{O}$ is modelled as a random oracle). □

[5] $\mathsf{E}^{\mathsf{NIZK}}$ could also output $\perp$, but in this case a reduction that contradicts the AoK online-extractor of $\Pi^{\mathcal{WI}}$ can be done.

We observe that even tough the in the above proof the ZK simulators needs to run in quasi-polynomial time we can still rely on the WI property of $\Pi^{\mathcal{WI}}$ since the WI of $\Pi^{\mathcal{WI}}$ holds against all powerful adversary. To prove the security of our NIZK in the case that $\Pi^{\mathcal{WI}}$ is not statistically/perfect WI we need to use complexity leveraging. That is, we need to assume that the security of $\Pi^{\mathcal{WI}}$ holds against an adversary that runs in time $\lambda^{\mathsf{poly}(\lambda)(\log \lambda)}$. Beside that, the overall proof stay the same.

3.4.2 On the WI of the Fischlin Protocol

We show how to use the Fischlin protocol to obtain an efficient non-interactive witness-indistinguishable (NIWI) argument of knowledge with straight-line extraction. The first attempt to obtain a NIWI AoK would be to construct a sigma-protocol Π^{OR} for a relation Rel_0 OR Rel_1 using the compiler of Sec. 3.2.5 by combining a sigma-protocol Π_0 for Rel_0 with a sigma-protocol Π_1 for Rel_1 where both Π_0 and Π_1 have unique third round. Then one might hope to use Π^{OR} as input of Fischlin's construction thus obtaining a NIWI AoK. Unfortunately this approach seems to not work because, even though Π_0 and Π_1 have unique third rounds, the resulting Π^{OR} does not have this properties anymore and therefore the Theorem 2 of Fischlin cannot be used. Indeed, it is easy to see that fixed the first and the second round (a, c) of Π^{OR} there are many possible third rounds since there are many possible ways to split the challenge c (see Sec. 3.2.5 for more details on the or-composition of [CDS94]). Therefore, as a first result of this section we prove that the Fischlin protocol is still an AoK with online extraction even though Π^{OR} does not have a quasi-unique third round. At high level the straight-line AoK property works also in this case because when the challenge messages c_0 and c_1 for the two first round messages a_0 and a_1 are fixed, then there exists a unique pair of messages (z_0, z_1) that makes the verifier $\mathcal{V}_0$ to accept (a_0, c_0, z_0) for the statement x_0 and $\mathcal{V}_1$ to accept (a_1, c_1, z_1) for the statement x_1. Therefore, even though Π^{OR} does not enjoys the unique third round property, it is sufficient that the values (z_0, z_1) are unique once that the first rounds and the challenges (c_0, c_1) are fixed.

The crucial technical point is that in the security proof both c_0 and c_1 are considered as being part of the second round of the sigma-protocol input to the Fischlin construction, instead of being part of the third round.

We also prove that in the case that the sigma-protocols Π_0 and Π_1 that composes Π^{OR} are not perfect, it is still possible to obtain a NIWI that is secure against PPT malicious verifier. In the WI proof, unfortunately, it is not clear how to reduce directly the security of the NIWI to the WI property of Π^{OR} and thereofre we rely on the Special HVZK property of the underling Π_0 and Π_1. For more details on this proof approach we refer the reader to the proof of Theorem 4.

We denote the instantiation of the Fischlin protocol that uses Π^{OR} as input with $\Pi^{\mathcal{WI}} = (\mathcal{P}^{\mathcal{WI}}, \mathcal{V}^{\mathcal{WI}})$ and propose it in Fig. 3.4. $\Pi^{\mathcal{WI}}$ is similar to the protocol proposed in Fig. 3.1, with the exception that it takes as input two sigma-protocols Π_0 and Π_1 and combines them using the [CDS94] compiler.

In this section we prove that the argument $\Pi^{\mathcal{WI}}$ proposed in Fig. 3.4 is WI and is an AoK with online extraction. More formally, we prove the following theorem.

Theorem 4. *Let Π_0 be a (perfect/statistical) sigma-protocol for the $\mathcal{NP}$-relation* Rel_0 *and Π_1 be a sigma-protocol for the $\mathcal{NP}$-relation* Rel_1 *such that both Π_0 and Π_1 have a quasi-unique third round, then $\Pi^{\mathcal{WI}}$ is (perfect/statistical) WI for the $\mathcal{NP}$-relation* $\mathsf{Rel}_{\mathsf{OR}}$ *and is an AoK with online extraction in the NPRO model.*

Proof. **Completeness.** The proof of completeness follows exactly the proof of Theorem 2 of [Fis05].

Witness-Indistinguishability. We prove that if the transform of [CDS94] showed in Sec. 3.2.5 on input Π_0 and Π_1 outputs a new sigma-protocol that is WI for the $\mathcal{NP}$-relation $\mathsf{Rel}_{\mathsf{OR}}$ then $\Pi^{\mathcal{WI}}$ is WI for $\mathsf{Rel}_{\mathsf{OR}}$ as well. We propose the proof for the case where Π_0 and Π_1 are sigma-protocols, the proof where Π_0 and Π_1 are perfect/statistical sigma-protocols follows by using the same arguments.

Let $\Pi_0 = (\mathcal{P}_0, \mathcal{V}_0)$ and $\Pi_1 = (\mathcal{P}_1, \mathcal{V}_1)$ be the sigma-protocols described above with challenge length $\ell = O(\log \lambda)$. Define the parameters b, r, S, t (as functions of λ) such that $br = \omega(\log \lambda)$, $2^{t-b} = \omega(\log \lambda)$, b, r, $t = O(\log \lambda)$, $S = O(r)$ and $b \leq t \leq \ell$. Define the following non-interactive argument system $\Pi^{\mathcal{WI}} = (\mathcal{P}^{\mathcal{WI}}, \mathcal{V}^{\mathcal{WI}})$ for relation $\mathsf{Rel}_{\mathtt{OR}}$ in the random oracle model, where the random oracle $\mathcal{O}$ maps to b bits.
Common input: security parameter λ, $\mathcal{NP}$-statement $x_0 \in L_0 \vee x_1 \in L_1$, the parameters b, r, S, t as defined above.
Input to $\mathcal{P}^{\mathcal{WI}}$: w_b s.t. $(x_b, w_b) \in \mathsf{Rel_b}$.
Proof. The prover $\mathcal{P}^{\mathcal{WI}}$ executes the following steps.

1. On input (x_b, w_b) run r times $\mathcal{P}_b$ (each time using fresh randomness) on input (x_b, w_b) thus obtaining $((\mathtt{aux}_1^b, a_1^b), (\mathtt{aux}_2^b, a_2^b), \ldots, (\mathtt{aux}_r^b, \ldots, a_r^b))$.
2. For $i = 1, \ldots, r$, pick $c_i^{1-b} \leftarrow \{0,1\}^t$ and compute $(a_i^{1-b}, z_i^{1-b}) \leftarrow \mathsf{Sim}(x_{1-b}, c_i^{1-b})$.
   Let $A = ((a_1^0, a_1^1), (a_2^0, a_2^1), \ldots, (a_r^0, a_r^1)$.
3. For $i = 1, \ldots, r$
   Pick $c_i \in \{0,1\}^t$ such that $\mathcal{O}(x_0, x_1, A, i, c_i, c_i^0, c_i^1, z_i^0, z_i^1) = 0^b$ where $c_i = c_i^0 \oplus c_i^1$ and $z_i^b \leftarrow \mathcal{P}_b(\mathtt{aux}_i, c_i^b)$.
   If such a c_i does not exist, then pick the first one for which the hash value is minimal among all 2^t hash values.
4. Output $\pi = (\{a_i^0, a_i^1, c_i^0, c_i^1, z_i^0, z_i^1\}_{i=1,\ldots,r})$.

Verification. The verifier $\mathcal{V}^{\mathcal{WI}}$ accepts if and only if for $i = 1, \ldots, r$: $\mathcal{V}_0(x_0, a_i^0, c_i^0, z_i^0) = 1$, $\mathcal{V}_1(x_1, a_i^1, c_i^1, z_i^1) = 1$, $c_i = c_i^0 \oplus c_i^1$ and $\Sigma_{i=1}^r \mathcal{O}(x_0, x_1, A, i, c_i, c_i^0, c_i^1, z_i^0, z_i^1) \leq S$.

Figure 3.4: $\Pi^{\mathcal{WI}}$: a NIWI AoK

We assume by contradiction that $\Pi^{\mathcal{WI}}$ is not WI and then we construct an adversary $\mathcal{A}^{\mathsf{SHVZK}}$ that breaks the Special HVZK of either Π_0 or Π_1. More formally, by contradiction we have that

$$\left|\Pr\left[\mathsf{ExpAWI}^0_{\Pi^{\mathcal{WI}},\mathcal{A}}(\lambda, \zeta) = 1\right] - \Pr\left[\mathsf{ExpAWI}^1_{\Pi^{\mathcal{WI}},\mathcal{A}}(\lambda, \zeta) = 1\right]\right| = \delta(\lambda)$$

for a non-negligible function δ.

Let b, r, S, t be the parameters defined in the description of $\Pi^{\mathcal{WI}}$ in Fig. 3.4, we now define the hybrid experiment $\mathcal{H}_i$ with $i \in \{1, \ldots, r\}$. The hybrid experiment $\mathcal{H}_i$ is formally described in Fig. 3.5 and takes as input the security parameter λ, the auxiliary information ζ for $\mathcal{A}$ and contains b, r, S and t.

We observe that $\mathsf{ExpAWI}^0_{\Pi^{\mathcal{WI}},\mathcal{A}}(\lambda, \zeta) = \mathcal{H}_0(\lambda, \zeta)$ and that $\mathsf{ExpAWI}^1_{\Pi^{\mathcal{WI}},\mathcal{A}}(\lambda, \zeta) = \mathcal{H}_r(\lambda, \zeta)$. Therefore, by contradiction, there must be a value $i \in \{1, \ldots r\}$ such that

$|\Pr[\mathcal{H}_{i-1}(\lambda, \zeta) = 1] - \Pr[\mathcal{H}_i(\lambda, \zeta) = 1]| = \delta(\lambda)$. In order to reach a contradiction we consider the additional intermediate hybrid experiment $\mathcal{H}_{\mathsf{int}}$ of Fig. 3.6. Informally, the difference between $\mathcal{H}_{i-1}$ and $\mathcal{H}_{\mathsf{int}}$ is that honest prover procedure $\mathcal{P}_1$ is used to compute (a_i^1, z_i^1) instead of the simulated one. Instead, the difference between $\mathcal{H}_{\mathsf{int}}$ and $\mathcal{H}_i$ is that in $\mathcal{H}_i$ the messages (a_i^0, z_i^0) are computed using the Special HVZK simulator of Π_0 instead of the honest prover procedure. We now prove the following lemma in order to complete the overall WI proof.

Upon receiving (x, w_0, w_1) from $\mathcal{A}$ execute the following steps.

1. Run i times $\mathcal{P}^{\mathsf{OR}}$ (each time using fresh randomness) on input (x_0, x_1, w_1) thus obtaining $((\mathtt{aux}^1_1, a^1_1), \dots, (\mathtt{aux}^1_i, a^1_i))$ and $((a^0_1, z^0_1), \dots, (a^0_i, z^0_i))$.
2. Run $r - i$ times $\mathcal{P}^{\mathsf{OR}}$ (each time using fresh randomness) on input (x_0, x_1, w_0) thus obtaining $((\mathtt{aux}^0_{i+1}, a^0_{i+1}), \dots, (\mathtt{aux}^0_r, a^0_r))$ and $((a^1_{i+1}, z^1_{i+1}), \dots, (a^1_r, z^1_r))$. Let $A = ((a^0_1, a^1_1), \dots, (a^0_r, a^1_r)$.
3. For $j = 1, \dots, i$
   Pick $c_j \in \{0,1\}^t$ such that $\mathcal{O}(x_0, x_1, A, j, c_j, c^0_j, c^1_j, z^0_j, z^1_j) = 0^b$ where $c_j = c^0_j \oplus c^1_j$ and $z^1_j \leftarrow \mathcal{P}_1(\mathtt{aux}^1_j, c^1_j)$.
   If such a c_j does not exist, then pick the first one for which the hash value is minimal among all 2^t hash values.
4. For $j = i + 1, \dots, r$
   Pick $c_j \in \{0,1\}^t$ such that $\mathcal{O}(x_0, x_1, A, j, c_j, c^0_j, c^1_j, z^0_j, z^1_j) = 0^b$ where $c_j = c^0_j \oplus c^1_j$ and $z^0_j \leftarrow \mathcal{P}_0(\mathtt{aux}^0_j, c^0_j)$.
   If such a c_j does not exist, then pick the first one for which the hash value is minimal among all 2^t hash values.
5. Send $\pi = (\{a^0_i, a^1_i, c^0_i, c^1_i, z^0_i, z^1_i\}_{i=1,\dots,r})$ to $\mathcal{A}$.
6. The adversary outputs $b' \in \{0,1\}$.

Figure 3.5: Hybrid experiment $\mathcal{H}_i$, with $i \in \{1, \dots, r\}$.

Lemma 1. *There exists a negligible function ν such that*

$$|\Pr[\mathcal{H}_{i-1}(\lambda, \zeta) = 1] - \Pr[\mathcal{H}_{\mathsf{int}}(\lambda, \zeta) = 1]| \leq \nu(\lambda)$$

and

$$|\Pr[\mathcal{H}_{\mathsf{int}}(\lambda, \zeta) = 1] - \Pr[\mathcal{H}_i(\lambda, \zeta) = 1]| \leq \nu(\lambda)$$

Proof. By contradiction one of the above two conditions does not hold. Therefore we assume that there exists a non-negligible function δ such that[6]

$$|\Pr[\mathcal{H}_{i-1}(\lambda, \zeta) = 1] - \Pr[\mathcal{H}_{\mathsf{int}}(\lambda, \zeta) = 1]| = \delta(\lambda).$$

In this case we can construct an adversary $\mathcal{A}^{\mathsf{SHVZK}}$ that breaks the Special HVZK of Π_1. Let $\mathcal{C}^{\mathsf{SHVZK}}$ be the challenger of the Special HVZK security game, $\mathcal{A}^{\mathsf{SHVZK}}$ internally runs $\mathcal{A}$ and execute the following steps.

1. Upon receiving (x_0, x_1, w_0, w_1) from $\mathcal{A}$, pick $\mathsf{c} \leftarrow \{0,1\}^t$ and send (x_1, w_1, c) to $\mathcal{C}^{\mathsf{SHVZK}}$.
2. Upon receiving (a, z) from $\mathcal{C}^{\mathsf{SHVZK}}$, run $i - 1$ times $\mathcal{P}_1$ (each time using fresh randomness) on input (x_1, w_1) thus obtaining $((\mathtt{aux}^1_1, a^1_1), \dots, (\mathtt{aux}^1_{i-1}, a^1_{i-1}))$. For $j = 1, \dots, i - 1$ pick $c^0_j \leftarrow \{0,1\}^t$ and compute $(a^0_j, z^0_j) \leftarrow \mathsf{Sim}_0(x_0, c^0_j)$.
3. Run $\mathcal{P}_0$ in input (x_0, w_0) thus obtaining $(a^0_i, \mathtt{aux}^0_i)$, sets $a^1_i = \mathsf{a}$.
4. Run $r-i$ times $\mathcal{P}_0$ (each time using fresh randomness) on input (x_0, w_0) thus obtaining $((\mathtt{aux}^0_{i+1}, a^0_{i+1}), \dots, (\mathtt{aux}^0_r, a^0_r)$

[6] The proof for the other case follows using exactly the same arguments but in that case we break the Special HVZK of Π_0 instead of Π_1.

```
Upon receiving $(x, w_0, w_1)$ from $\mathcal{A}$ execute the following steps.
  1. Run $i-1$ times $\mathcal{P}_1$ (each time using fresh randomness) on input $(x_1, w_1)$ thus obtaining $((\mathsf{aux}^1_1, a^1_1), \ldots, (\mathsf{aux}^1_{i-1}, a^1_{i-1}))$.
  2. For $j = 1, \ldots, i-1$ pick $c^0_j \leftarrow \{0,1\}^t$ and compute $(a^0_j, z^0_j) \leftarrow \mathsf{Sim}_0(x_0, c^0_j)$.
  3. Run $\mathcal{P}_1$ in input $(x_1, w_1)$ thus obtaining $(a^1_i, \mathsf{aux}^1_i)$ and run $\mathcal{P}_0$ in input $(x_0, w_0)$ thus obtaining $(a^0_i, \mathsf{aux}^0_i)$.
  4. Run $r-i$ times $\mathcal{P}_0$ (each time using fresh randomness) on input $(x_0, w_0)$ thus obtaining $((\mathsf{aux}^0_{i+1}, a^0_{i+1}), \ldots, (\mathsf{aux}^0_r, a^0_r))$.
  5. For $j = i+1, \ldots, r$ pick $c^1_j \leftarrow \{0,1\}^t$ and compute $(a^1_j, z^1_j) \leftarrow \mathsf{Sim}_1(x_1, c^1_j)$.
     Let $A = ((a^0_1, a^1_1), (a^0_2, a^1_2), \ldots, (a^0_r, a^1_r)$.
  6. For $j = 1, \ldots, i-1$
       Pick $c_j \in \{0,1\}^t$ such that $\mathcal{O}(x_0, x_1, A, j, c_j, c^0_j, c^1_j, z^0_j, z^1_j) = 0^b$ where $c_j = c^0_j \oplus c^1_j$ and $z^1_j \leftarrow \mathcal{P}_1(\mathsf{aux}^1_j, c^1_j)$.
       If such a $c_j$ does not exist, then pick the first one for which the hash value is minimal among all $2^t$ hash values.
  7. Pick $c_i \in \{0,1\}^t$ such that $\mathcal{O}(x_0, x_1, A, i, c_i, c^0_i, c^1_i, z^0_i, z^1_i) = 0^b$ where $c^0_i \leftarrow \{0,1\}^t$, $c_i = c^0_i \oplus c^1_i$, $z^1_i \leftarrow \mathcal{P}_1(\mathsf{aux}^1_i, c^1_i)$ and $z^0_i \leftarrow \mathcal{P}_0(\mathsf{aux}^0_i, c^0_i)$
     If such a $c_i$ does not exist, then pick the first one for which the hash value is minimal among all $2^t$ hash values.
  8. For $j = i+1, \ldots, r$
       Pick $c_j \in \{0,1\}^t$ such that $\mathcal{O}(x_0, x_1, A, j, c_j, c^0_j, c^1_j, z^0_j, z^1_j) = 0^b$ where $c_j = c^0_j \oplus c^1_j$ and $z^0_j \leftarrow \mathcal{P}_0(\mathsf{aux}^0_j, c^0_j)$.
       If such a $c_j$ does not exist, then pick the first one for which the hash value is minimal among all $2^t$ hash values.
  9. Send $\pi = (\{a^0_i, a^1_i, c^0_i, c^1_i, z^0_i, z^1_i\}_{i=1,\ldots,r})$ to $\mathcal{A}$.
 10. The adversary outputs $b' \in \{0,1\}$.
```

Figure 3.6: Hybrid experiment $\mathcal{H}_{\mathsf{int}}$.

5. For $j = i+1, \ldots, r$ pick $c^1_j \leftarrow \{0,1\}^t$ and compute $(a^1_j, z^1_j) \leftarrow \mathsf{Sim}_1(x_1, c^1_j)$.

6. define $A = ((a^0_1, a^1_1), (a^0_2, a^1_2), \ldots, (a^0_r, a^1_r))$.

7. For $j = 1, \ldots, i-1$

   Pick $c_j \in \{0,1\}^t$ such that $\mathcal{O}(x_0, x_1, A, j, c_j, c^0_j, c^1_j, z^0_j, z^1_j) = 0^b$ where $c_j = c^0_j \oplus c^1_j$ and $z^1_j \leftarrow \mathcal{P}_1(\mathsf{aux}^1_j, c^1_j)$.

   If such a c_j does not exist, then pick the first one for which the hash value is minimal among all 2^t hash values.

8. Pick $c_i \in \{0,1\}^t$ such that $\mathcal{O}(x_0, x_1, A, i, c_i, c^0_i, c^1_i, z^0_i, z^1_i) = 0^b$ where $c_i = \mathsf{c} \oplus c^0_i$, $z^0_i \leftarrow \mathcal{P}_0(\mathsf{aux}^0_i, c^0_i)$ and $z^1_i = \mathsf{z}$.

   If such a c_i does not exist, then pick the first one for which the hash value is minimal among all 2^t hash values[7].

9. For $j = i+1, \ldots, r$

   Pick $c_j \in \{0,1\}^t$ such that $\mathcal{O}(x_0, x_1, A, j, c_j, c^0_j, c^1_j, z^0_j, z^1_j) = 0^b$ where $c_j = c^0_j \oplus c^1_j$ and $z^0_j \leftarrow \mathcal{P}_0(\mathtt{aux}^0_j, c^0_j)$.

   If such a c_j does not exist, then pick the first one for which the hash value is minimal among all 2^t hash values.

10. Send $\pi = (\{a^0_i, a^1_i, c^0_i, c^1_i, z^0_i, z^1_i\}_{i=1,\ldots,r})$ to $\mathcal{A}$.

11. Output what $\mathcal{A}$ outputs.

It should be easy to see that if the messages (a, z) have been computed by $\mathcal{C}^{\mathsf{SHVZK}}$ using the Special HVZK simulator Sim_1 then the output of $\mathcal{A}^{\mathsf{SHVZK}}$ corresponds to the output of $\mathcal{A}$ in $\mathcal{H}_{i-1}$, and to the output of $\mathcal{A}$ in $\mathcal{H}_{\mathsf{int}}$ if (a, z) are computed using $\mathcal{P}_1$ on input the witness w_1 for x_1. Therefore we have obtained an adversary $\mathcal{A}^{\mathsf{SHVZK}}$ such that breaks the security of the Special HVZK of Π_1 thus reaching a contradiction. $\square$

Argument of Knowledge with Online Extraction. Here we follow in large part the proof of Theorem 2 of [Fis05]. In order to complete this proof, we need to show a knowledge extractor $\mathsf{Ext}(x, \pi, \mathcal{Q}_{\mathcal{O}}(\mathcal{A}))$ that except with negligible probability over the choice of $\mathcal{O}$, is able to output a witness w_b such that $(x_b, w_b) \in \mathsf{Rel}_b$ for an accepted proof $\pi = (\{a^0_i, a^1_i, c^0_i, c^1_i, z^0_i, z^1_i\}_{i=1,\ldots,r})$ with respect to (x_0, x_1). Since Π^{OR} is special sound, then the algorithm Ext just needs to look into $\mathcal{Q}_{\mathcal{O}}(\mathcal{A})$ for a query $\mathcal{O}(x_0, x_1, A, i, c_i, c^0_i, c^1_i, z^0_i, z^1_i)$ and for another query $\mathcal{O}(x_0, x_1, A, i, \tilde{c}_i, \tilde{c}^0_i, \tilde{c}^1_i, \tilde{z}^0_i, \tilde{z}^1_i)$ such that $\mathcal{V}^{\mathsf{OR}}(x_0, x_1, a^0_i, a^1_i, \tilde{c}^0_i, \tilde{c}^1_i, \tilde{z}^0_i, \tilde{z}^1_i) = 1$ and $c_i \neq \tilde{c}_i$. Indeed, from the special soundness of Π^{OR}, this yields to an efficient extraction procedure of either the witness for x_0 or x_1. If there are not such queries then Ext simply outputs $\perp$. We now need to bound the probability that such two queries do not exist, but still $\mathcal{V}^{\mathcal{WI}}$ accepts π with probability. Consider the set of tuples (x_0, x_1, A) such that $\mathcal{A}$ queries $\mathcal{O}$ about $(x_0, x_1, A, i, c^0_i, c^1_i, z^0_i, z^1_i)$ for some $i, c^0_i, c^1_i, z^0_i, z^1_i$ and such that $\mathcal{V}^{\mathsf{OR}}(x_0, x_1, a^0_i, a^1_i, c^0_i, c^1_i, z^0_i, z^1_i) = 1$ (we can neglect tuples with invalid proofs). Let $Q = |\mathcal{Q}_{\mathcal{O}}(\mathcal{A})|$, then there are at most $Q+1$ of these tuples (x_0, x_1, A). Fix one of the tuples for the moment, say, (x_0, x_1, A). By contradiction, for this tuple and any i algorithm $\mathcal{A}$ never queries $\mathcal{O}$ about two values $(x_0, x_1, A, i, c_i, c^0_i, c^1_i, z^0_i, z^1_i)$, $(x_0, x_1, a^0_i, a^1_i, \tilde{c}^0_i, \tilde{c}^1_i, \tilde{z}^0_i, \tilde{z}^1_i)$ with $c_i \neq \tilde{c}_i$ wich $\mathcal{V}^{\mathsf{OR}}$ would accept (we note that if $c_i \neq \tilde{c}_i$ then either $c^0_i \neq \tilde{c}^1_i$ or $c^1_i \neq \tilde{c}^1_i$).

Similarly, we can assume that $\mathcal{A}$ never queries about $(x_0, x_1, A, i, c_i, c^0_i, c^1_i, z^0_i, z^1_i)$, $(x_0, x_1, a^0_i, a^1_i, \tilde{c}^0_i, \tilde{c}^1_i, \tilde{z}^0_i, \tilde{z}^1_i)$ with either $z^0_i \neq \tilde{z}^0_i$ or $z^1_i \neq \tilde{z}^1_i$, else this would contradict the property of unique responses of either Π_0 or Π_1. This allows us to assign a set of unique values $s_1, \ldots, s_r$ to (x_0, x_1, A) such that s_i equals $\mathcal{O}(x_0, x_1, A, i, c_i, c^0_i, c^1_i, z^0_i, z^1_i)$ if $\mathcal{A}$ queries about any such tuple[8]. Conclusively, the values $s_1, \ldots, s_r$ assigned to (x_0, x_1, A) are all random and independent. Given such an assignment we calculate the probability that the sum does not exceed the threshold value S. We consider $\binom{T+r-1}{r-1}$ combinations to represent a sum $T \leq S$ with r values $s_1, \ldots, s_r \geq 0$. There are $\sum_{T=0}^{S} \binom{T+r-1}{r-1} \leq (S+1) \cdot \binom{S+r-1}{r-1} \leq (S+1) \cdot \frac{e(S+r)}{r-1}^{r-1}$ possibilities. Since $S = O(r)$ we have $S + r \leq c(r-1)$ for some constant c, and the number of combinations is bounded above by $(S+1) \cdot 2^{\log(ec)r}$, which is polynomial in λ. Since $s_1, \ldots, s_r$ are random, the probability of obtaining such a sum $T \leq S$ is this number of combinations divided by 2^{br}. By the choice of parameters this is negligible. Finally, we extend the analysis from a fixed query to the set of the $Q+1$ possibilities (x_0, x_1, A). Since Q is polynomial the probability of finding a valid proof among this set for which the extractor also fails is bounded by $(Q+1)(S+1) \cdot 2^{(\log(ec)-b)r}$. This remains negligible. $\square$

[7]We observe that $\mathcal{A}^{\mathsf{SHVZK}}$ can iterate on all the possible c_j since he has the freedom to pick any possible c^0_j.

[8]From this point forward the proof follows exactly the same steps proposed in [Fis05], but for completeness we propose the complete proof.

How to instantiate $\Pi^{\mathcal{WI}}$.

Theorem 4 states that Π_0 and Π_1 can be instantiated with any (perfect/statistical) sigma-protocol with quasi-unique third round. As observed in [Fis05], and also in this work, there are many perfect sigma-protocols with quasi-unique third round such as Schnorr's protocol for discrete logarithm, the protocol for Diffie-Hellman (DH) tuples the protocol of [MP03] for proving knowledge of committed messages, and all sigma-protocols in the well known class proposed by Cramer in [CD98] and Maurer in [Mau15].

If we consider just sigma-protocol, then for our construction we could use the protocol proposed by Blum in [Blu86]. In this, the first round consists of a set of commitments, and in the third round only a subset of these commitments are opened by the prover. When there is only one witness for the statement being proved, for the binding of the underling commitment scheme, Blum's protocols has a quasi-unique third round.

3.5 Complexity Analysis of Π^{NIZK} when Instantiated with the Fischlin's Transform

Our construction consists of one execution of $\Pi^{\mathcal{WI}}$ and one evaluation of $\mathcal{O}$ on input x. Therefore, to give a concrete idea of the efficiency of our protocol we need to define the two sigma-protocols used as input of $\Pi^{\mathcal{WI}}$ (see Fig 3.4).

Therefore, we need a sigma-protocol $\Pi^{\mathtt{puz}}$ for the $\mathcal{NP}$-relation $\mathsf{Rel}_{\mathtt{puz}} = \{(\mathtt{puz}, \mathtt{sol}) : \mathsf{Verify}(1^\lambda, h, \mathtt{puz}, w) = 1)\}$ and a sigma-protocol Π^{L} for the $\mathcal{NP}$-relation $\mathsf{Rel}_{\mathsf{L}}$. This means that we need to use a dense samplable puzzle system that admits a sigma-protocol. In [BKZZ16] the authors propose a dense samplable puzzle system where the puzzle is an instance of the discrete logarithm problem. So we can use as $\Pi^{\mathtt{puz}}$ the well known Schnorr protocol [Sch89]. Following the analysis of Sec. 3.3.1, we obtain an online-extractor that fails with probability at most $\mathcal{Q}2^{-72}$ where the total number of hash function evaluations is at most $2^9 r + 1$ and the number of executions of $\Pi^{\mathtt{OR}}$ is $r = 10$, where $\Pi^{\mathtt{OR}}$ denotes the output of the or-composition proposed in Sec. 3.2.5 using $\Pi^{\mathtt{puz}}$ and Π^{L} as input.

To give a practical example, let us consider the complexity of Π^{NIZK} when $\mathsf{Rel}_{\mathsf{L}}$ is the discrete log $\mathcal{NP}$ relation

$$\{((\mathcal{G}, q, g, Y), y) : g^y = Y\}$$

where $\mathcal{G}$ is a group of prime-order q. We construct $\Pi_{\mathcal{WI}}$ using two instantiation of the Schnorr protocol thus obtaining a sigma-protocol that requires 3 exponentiation to be executed (see [CPS$^+$16] for a detailed analysis). So our protocol Π^{NIZK} requires 30 exponentiation and $2^9 r + 1 = 5121$ hash evaluations.

Chapter 4

Secure Groups and Their Applications in MPC-based Threshold Cryptography

Recall that a finite group is defined as a finite set $\mathbb{G}$ of group elements (including the identity element) together with an associative group operation $*$ and the corresponding inverse operation. We propose a scheme to implement finite groups as oblivious data structures, meaning that no information can be inferred about the values of the group elements after a sequence of operations. For a given group, the scheme defines the oblivious representation of, and oblivious operations on group elements. Operations include the group law, exponentiation and inversion, random sampling and encoding/decoding.

The oblivious operations are defined by a set of secure multiparty computation (MPC) protocols. Practical protocols are presented for the group of quadratic residues, elliptic curves groups and class groups of imaginary quadratic orders. We demonstrate these protocols in a standard setting for information theoretically secure MPC, tolerating a dishonest minority of passively corrupt parties.

We introduce a practical protocol to calculate the extended gcd (xgcd) of two secret-shared integers adapting recent work by Bernstein and Yang [BY19b] from the p-adic setting to the finite field setting. This xgcd MPC protocol is of independent interest. We apply it to implement the class group operation in MPC. To reduce positive definite binary quadratic forms, we present a protocol based on the binary reduction algorithm from [AF06].

To demonstrate an application of secure groups, we construct a protocol to convert ciphertexts to secret shares. This protocol extends a classical threshold cryptosystem by enabling in- and output to a multiparty computation by simply communicating one ciphertext.

The secure groups extension to the MPC-framework MPyC [Sch18] is available from `https://github.com/toonsegers/sec_groups` under the MIT license. The *secure groups* includes the extended gcd protocol.

Roadmap. This contribution is organized as follows. Section 4.1 introduces preliminaries for secure multiparty computation. Section 4.2 presents a secure protocol for the extended gcd. Section 4.3 introduces the definition of a secure group. Section 4.4 presents generic constructions for secure groups. Section 4.5 presents generic protocols for the following tasks: conditional (if-else), random sampling, inversion and exponentiation. Section 4.6 presents specific constructions for groups used in cryptography, particularly quadratic residues, elliptic curve groups and class groups. Section 4.7 presents a threshold cryptosystem built from secure groups. Section 4.8 concludes our work by introducing our Python package for secure groups.

4.1 MPC Setting

Let $\mathrm{len}(x) = \lceil \log_2(|x| + 1) \rceil$ denote the bit length of integer x.

We consider an MPC setting with m parties tolerating a dishonest majority of up to t passively corrupt parties, $0 \le t < m/2$. The basic protocols for secure addition and multiplication over a finite field rely on Shamir secret sharing [BGW88, GRR98]. For our practical experiments we use the MPyC framework [Sch18].

Let $[\![a]\!]$ denote a Shamir secret sharing for any finite field element $a \in \mathbb{F}$. If necessary to specify the field's modulus q, denote $[\![a]\!]_q$ for $a \in \mathbb{F}_q$. The bit length of an integer q is denoted by ℓ_q. In the context of class groups and integer xgcd, $[\![a]\!]$ denotes a Shamir sharing of a bounded integer $a \in \mathbb{Z}$ with $|a| \le Q$, such that repeated integer multiplication does not flow over the field modulus, q. We will refer to the bit length of q/Q as headroom. A vector is denoted in bold, as well as a vector of shares, e.g., $[\![\mathbf{x}]\!]$. We typically suppress the modulus in the notation of a secret share.

It is required that $q > m$. Otherwise, use the smallest extension $\mathbb{F}_{q^d}$ such that $q^d > m$. E.g., for $\mathbb{F}_2$ used in permutation matrices, or for $\mathbb{F}_7$ in linear representation of the Rubik's cube group.

To instantiate secure groups of non-prime order, we also need additive sharings over $\mathbb{Z}$ as well as additive sharings over $\mathbb{Z}_n$ for arbitrary $n > 0$. We use $[\![a]\!]^+_{\mathbb{Z}}$ to denote an additive sharing of $a \in \mathbb{Z}$ and $[\![a]\!]^+_n$ to denote an additive sharing of a modulo n. The shares will be random integers $a_i \in_R \mathbb{Z}_n$ satisfying $\sum_i a_i = a$. We note that a Shamir sharing $[\![a]\!]$ can be converted to an additive sharing $[\![a]\!]^+_p$ by letting each party in a quorum Q, $|Q| \ge t + 1$, use $a_i = \lambda_{Q,i} a'_i$ as its additive share, where a'_i is its Shamir share.

Let $\mathcal{P}_i$ for $i \in \{1, ..., m\}$ denote MPC parties. Names of algorithms and protocols are in $\mathsf{sans\text{-}serif}$. Let $a \leftarrow \mathsf{open}([\![a]\!])$ denote the protocol for opening a secret share, $[\![r]\!] \leftarrow \mathsf{random}(\mathbb{F})$ the computation of a random Shamir share, $[\![\mathbf{x}]\!] = ([\![a_0]\!], ..., [\![a_{\ell-1}]\!]) \leftarrow \mathsf{bd}([\![a]\!])$ the bit-decomposition of one Shamir share into ℓ_a Shamir shares, $[\![b]\!] \leftarrow \mathsf{ltz}([\![x]\!])$ the less than zero comparison of an (integer) share and $[\![a]\!] \leftarrow \mathsf{norm}([\![\mathbf{x}]\!])$ the determination of the most (or least) significant bit equal to 1. See [Hoo12], [Tof07] and [DFK+06] for these and other common MPC protocols, which we will use as black-boxes.

A protocol that generates a vector of j secure field elements in $\mathbb{F} \cap \{0, 1\}$, $[\![\boldsymbol{r}]\!]$, is denoted by $[\![\boldsymbol{r}]\!] \leftarrow \mathsf{random\ bits}(\mathbb{F}, j)$.

Secure integer division $([\![q]\!], [\![r]\!]) \leftarrow \mathsf{div}([\![a]\!], [\![b]\!])$, such that $a = bq + r$, is also used as black box. Our implementations are based on the Newton-Raphson method [ACS02, CS10] and Taylor series [DNT12].

4.2 Safe and Secure Extended GCD

This section presents a new protocol for computing the extended gcd of secret-shared integers, which we will use to implement secure class groups of imaginary quadratic fields. The protocol is also of independent interest, as it is simple, quite practical, and its performance is superior to any of the known alternatives. Moreover, our result can also be used as an alternative to the constant-time extended gcd algorithms of Bernstein and Yang [BY19b] on which it is based, without the use of any 2-adic arithmetic.

As starting point we take the constant-time extended gcd algorithm $\mathsf{divsteps2}$ by Bernstein and Yang [BY19b, Figure 10.1]. Our protocol xgcd, see Protocol 1, retains the algorithmic flow of their $\mathsf{divsteps2}$ algorithm, which is controlled by the following step function [BY19b, Section 8]:

$$\mathsf{divstep}(\delta, f, g) = \begin{cases} (1 - \delta, g, (g - f)/2), & \text{if } \delta > 0 \text{ and } g \text{ is odd,} \\ (1 + \delta, f, (g + (g \bmod 2)f)/2), & \text{otherwise.} \end{cases} \tag{4.1}$$

Throughout, variable f is ensured to be an odd integer, and therefore the divisions by 2 are without remainder in both cases of the step function. Bernstein and Yang argue that this step function compares favorably with alternatives from the literature, e.g., the Brent–Kung step function [BK85] and the Stehlé–Zimmerman step

```
Algorithm 1 xgcd($n, a, b$)                                   $n, a, b \in \mathbb{N}$, $a$ odd
1: $\delta, f, g, u, v, q, r \leftarrow 1, a, b, 1, 0, 0, 1$      ▷ $f = ua + vb$, $g = qa + rb$
2: for $i = 1$ to $n$ do
3:     $g_0 \leftarrow g \bmod 2$
4:     if $\delta > 0$ and $g_0 = 1$ then
5:         $\delta, f, g, u, v, q, r \leftarrow -\delta, g, -f, q, r, -u, -v$
6:     if $g_0 = 1$ then
7:         $g, q, r \leftarrow g + f, q + u, r + v$
8:     if $r \bmod 2 = 1$ then
9:         $q, r \leftarrow q - b, r + a$
10:    $\delta, g, q, r \leftarrow \delta + 1, g/2, q/2, r/2$
11: if $f < 0$ then
12:    $f, u, v \leftarrow -f, -u, -v$
13: return $f, u, v$
```

function [SZ04]. The computational overhead of function divstep is small, and the required number of iterations n as a function of the bit lengths of the inputs a and b compares favorably with the alternatives. Concretely, with $(\delta_n, f_n, g_n) = \mathsf{divstep}^n(1, a, b)$ for odd a, Bernstein and Yang prove that $f_n = \gcd(a, b)$ and $g_n = 0$ holds for $n = \mathsf{iterations}(\max(\operatorname{len}(a), \operatorname{len}(b)))$, where

$$\mathsf{iterations}(d) = \begin{cases} \lfloor (49d + 80)/17 \rfloor, & \text{if } d < 46, \\ \lfloor (49d + 57)/17 \rfloor, & \text{otherwise.} \end{cases} \tag{4.2}$$

Hence, $\mathsf{iterations}(d) \approx 3d$.

The first major change compared to Bernstein and Yang's divsteps2 algorithm is that we entirely drop the use of truncation for f and g. In our MPC setting f and g are secret-shared values (over a prime field of large order) and therefore limiting the sizes of f and g is not useful. The second major change is that we will avoid the use of 2-adic arithmetic entirely, by ensuring that the Bezout coefficients will remain integral throughout all iterations. Concretely, this means that we will make sure that coefficients q and r are even before the division by 2 at the end of each iteration: if q and r are odd, we use $q - b$ and $r + a$ instead, which will then be even, because a is odd by assumption. We thus obtain Algorithm 1 as an alternative to Bernstein-Yang's constant-time algorithm.

We turn Algorithm 1 into a secure protocol operating on secret-shared values as follows. We drop variables q and u from the main loop, and instead set $u = (f - vb)/a$ at the end. Overall, this saves $3 * n$ secure multiplications for q and n secure multiplications for u that are otherwise needed to implement the if-then statements obliviously. See Protocol 1 for the result.

All operations on the remaining variables δ, g, f, v, r are done securely. The secure computations of $[\![g \bmod 2]\!]$ and $[\![r \bmod 2]\!]$ are done by using a secure protocol for computing the least significant bit. The secure computation of $[\![\delta > 0]\!]$ is the most expensive part of each iteration. However, by taking into account that δ is bounded above by $i + 1$, we can reduce the cost of this secure comparison.

For inputs a and b of bit length ℓ, the round complexity of Protocol 1 will be $O(\ell)$. Per round, the computational work is proportional to the work for $O(\log \ell)$ secure multiplications.

Theorem 5. *The value v output by xgcd satisfies $-\frac{3}{2}a \leq v \leq \frac{9}{2}a$.*

Proof. The for-loop in the xgcd algorithm can be divided into consecutive runs that start and end with a state in which $\delta = 1$ holds. The length of a run is defined as the number of loop-iterations. As explained in [BY19b,

Protocol 1 $\mathsf{xgcd}(n, [\![a]\!], [\![b]\!])$ $\qquad n, a, b \in \mathbb{N}$, a odd

```
1: $[\![\delta]\!], [\![f]\!], [\![g]\!], [\![v]\!], [\![r]\!] \leftarrow 1, [\![a]\!], [\![b]\!], 0, 1$
2: for $i = 1$ to $n$ do
3:     $[\![g_0]\!] \leftarrow [\![g \bmod 2]\!]$
4:     if $[\![\delta]\!] > 0$ and $[\![g_0]\!] = 1$ then
5:         $[\![\delta]\!], [\![f]\!], [\![g]\!], [\![v]\!], [\![r]\!] \leftarrow -[\![\delta]\!], [\![g]\!], -[\![f]\!], [\![r]\!], -[\![v]\!]$
6:     if $[\![g_0]\!] = 1$ then
7:         $[\![g]\!], [\![r]\!] \leftarrow [\![g]\!] + [\![f]\!], [\![r]\!] + [\![v]\!]$
8:     if $[\![r \bmod 2]\!] = 1$ then
9:         $[\![r]\!] \leftarrow [\![r]\!] + [\![a]\!]$
10:    $[\![\delta]\!], [\![g]\!], [\![r]\!] \leftarrow [\![\delta]\!] + 1, [\![g]\!]/2, [\![r]\!]/2$
11: if $[\![f]\!] < 0$ then
12:     $[\![f]\!], [\![v]\!] \leftarrow -[\![f]\!], -[\![v]\!]$
13: $[\![u]\!] \leftarrow ([\![f]\!] - [\![v]\!] * [\![b]\!]/[\![a]\!]$
14: return $[\![f]\!], [\![u]\!], [\![v]\!]$        $\triangleright f = ua + vb = \gcd(a, b)$
```

Appendix G], each run is of length $2e$ for some $e \geq 1$. The last run ends in a state with $\delta = 1$ and $g = 0$, after which the value of v does not change anymore.

We claim that $-\frac{3}{2}a \leq v, r \leq \frac{9}{2}a$ is an invariant that holds between these runs.

Clearly, the initial values $v, r = 0, 1$ satisfy these bounds.

Next, let v_0, r_0 denote the values of v, r at the start of a run of length $2e$, satisfying the bounds $-\frac{3}{2}a \leq v_0, r_0 \leq \frac{9}{2}a$. Let v', r' denote the least possible values for v, r at the end of the run, and let v'', r'' denote the largest possible values for v, r at the of the run.

Define $\phi(r, a, N) = 2^{-N}(r + (-1 + 2^N)a)$. Then we have:

$$v' \geq \phi(r_0, 0, e-1) = r_0/2^{e-1} \geq -\tfrac{3}{2}a/2^{e-1} \geq -\tfrac{3}{2}a,$$

and

$$r' \geq \phi(\phi(-v_0, v', 1), 0, e) = (v' - v_0)/2^{e+1} \geq (-\tfrac{3}{2}a - \tfrac{9}{2}a)/2^{e+1} = 6a/2^{e+1} \geq -\tfrac{3}{2}a.$$

Similarly, we have:

$$\begin{aligned} v'' &\leq \phi(r_0, a, e-1), \\ &= (r_0 + (2^{e-1} - 1)a)/2^{e-1} \\ &\leq (\tfrac{9}{2} + 2^{e-1} - 1)a/2^{e-1} \\ &\leq (\tfrac{7}{2} + 2^{e-1})a/2^{e-1} \\ &\leq (\tfrac{7}{2} + 1)a \\ &= \tfrac{9}{2}a, \end{aligned}$$

and

$$\begin{aligned} r'' &= \phi(-v_0, v'' + a, e+1) \\ &\leq \phi(-v_0, \tfrac{11}{2}a, e+1) \\ &= (-v_0 + (2^{e+1} - 1)\tfrac{11}{2}a)/2^{e+1} \\ &\leq (\tfrac{3}{2}a + (2^{e+1} - 1)\tfrac{11}{2}a)/2^{e+1} \\ &= ((\tfrac{11}{2}2^{e+1} - 4)a)/2^{e+1} \\ &= (\tfrac{11}{2} - 4/2^{e+1})a \\ &\leq (\tfrac{11}{2} - 1)a \\ &= \tfrac{9}{2}a. \end{aligned}$$

□

The extended gcd protocol can be easily adapted for computing the modular inverse of b modulo a, assuming $\gcd(a, b) = 1$. NOTE: common case a is an odd prime and $0 < b < a$.

Also, the protocol can be stripped to obtain an efficient protocol for computing $\gcd(a, b)$ securely.

4.3 Definition of Secure Groups

Let $(\mathbb{G}, *)$ denote an arbitrary finite group, written multiplicatively. To define secure group schemes we will use $[\![a]\!]_{\mathbb{G}}$ to denote a secure representation of a group element $a \in \mathbb{G}$. In general, such a secure representation $[\![a]\!]_{\mathbb{G}}$ will be constructed from one or more secret-shared finite field elements. Also, we may compose secure representations, e.g., we may put $[\![a]\!]_{\mathbb{G}} = ([\![a']\!]_{\mathbb{G}'}, [\![a'']\!]_{\mathbb{G}''})$ as secure representation of $a = (a', a'') \in \mathbb{G}$ for a direct product group $\mathbb{G} = \mathbb{G}' \times \mathbb{G}''$.

A secure group scheme should allow us to apply the group operation $*$ to given $[\![a]\!]_{\mathbb{G}}$ and $[\![b]\!]_{\mathbb{G}}$, and obtain $[\![a * b]\!]_{\mathbb{G}}$ as a result. We will refer to this as a secure application of the group operation, or as a secure group operation, for short. Similarly, a secure group scheme may allow us to perform a secure inversion, which lets us obtain $[\![a^{-1}]\!]_{\mathbb{G}}$ for a given $[\![a]\!]_{\mathbb{G}}$. Another common task is to generate a random sample $[\![a]\!]_{\mathbb{G}}$ with $a \in_R \mathbb{G}$, hence to obtain a secure representation of a group element a drawn from the uniform distribution on $\mathbb{G}$.

To cover a representative set of tasks, we define secure groups as follows.

Definition 6. A **secure group scheme** for $(\mathbb{G}, *)$ comprises protocols for the following tasks, where $a, b \in \mathbb{G}$.

Group operation. Given $[\![a]\!]_{\mathbb{G}}$ and $[\![b]\!]_{\mathbb{G}}$, compute $[\![a * b]\!]_{\mathbb{G}}$.

Inversion. Given $[\![a]\!]_{\mathbb{G}}$, compute $[\![a^{-1}]\!]_{\mathbb{G}}$.

Equality test. Given $[\![a]\!]_{\mathbb{G}}$ and $[\![b]\!]_{\mathbb{G}}$, compute $[\![a = b]\!]$.

Conditional. Given $[\![a]\!]_{\mathbb{G}}$, $[\![b]\!]_{\mathbb{G}}$ and $[\![x]\!]$ with $x \in \{0, 1\}$, compute $[\![a^x b^{1-x}]\!]_{\mathbb{G}}$.

Exponentiation. Given $[\![a]\!]_{\mathbb{G}}$ and $[\![x]\!]$ with $x \in \mathbb{Z}$, compute $[\![a^x]\!]_{\mathbb{G}}$.

Random sampling. Compute $[\![a]\!]_{\mathbb{G}}$ with $a \in_R \mathbb{G}$ (or, close to uniform).

En/decoding. For a set S and an injective map $\sigma : S \to \mathbb{G}$:

- *Encoding.* Given $[\![s]\!]$, compute $[\![\sigma(s)]\!]_{\mathbb{G}}$.
- *Decoding.* Given $[\![a]\!]_{\mathbb{G}}$ with $a \in \sigma(S)$, compute $[\![\sigma^{-1}(a)]\!]$.

By default, all inputs and outputs to these protocols are secret-shared. In addition, the scheme comprises protocols for all these tasks where some of the inputs and outputs are public and/or private.

By definition, a secure group scheme thus includes an ordinary group scheme where all protocols operate on public values. Also note that there may be multiple encoding/decodings for a group $\mathbb{G}$, each defined on a specific set S.

Examples of important variants of the default protocols of a secure group scheme are the following cases for secure exponentiation, which will be covered in Section 4.5.4: (i) given public a and secret $[\![x]\!]$ compute secret $[\![a^x]\!]_{\mathbb{G}}$, (ii) given secret $[\![a]\!]_{\mathbb{G}}$ and public x compute secret $[\![a^x]\!]_{\mathbb{G}}$. Similarly, variants of the trivial secure encoding/decoding protocols with $S = \mathbb{G}$ and σ the identity map on $\mathbb{G}$ will allow us to support private input and output of group elements: (i) given private input a, compute secret $[\![a]\!]_{\mathbb{G}}$, and (ii) given secret $[\![a]\!]_{\mathbb{G}}$, compute private output a. The private inputs and outputs may even belong to external parties, not taking part in the MPC protocol.

We have included a representative range of tasks in our definition of secure groups. Even though some of the tasks are redundant, they are included nevertheless because these tasks cannot necessarily be implemented *without loss* *of efficiency* in terms of the other tasks. For instance, secure random sampling can be implemented by letting each party $\mathcal{P}_i$ generate a random group element $a_i \in_R \mathbb{G}$ as private input, for $i = 0, \dots, t$, and then computing the product $[\![a_0]\!]_{\mathbb{G}} \cdots [\![a_t]\!]_{\mathbb{G}}$ securely. Depending on the implementation, this may be done much more efficiently for particular groups.

A class of tasks that we have not (yet) incorporated in the definition of secure groups are higher-order versions of basic tasks such exponentiation. An important example is multi-exponentiation $[\![\prod_i a_i^{x_i}]\!]_{\mathbb{G}}$. Similarly, the n-ary group operation $[\![\prod_{i=1}^n a_i]\!]_{\mathbb{G}}$ may be included as a basic task, for which there may be more efficient solutions compared to $n-1$ applications of the secure group operation.

4.4 Generic Constructions of Secure Groups

4.4.1 Secure Groups From Table Lookup

For our first generic construction of secure group schemes for arbitrary groups $\mathbb{G}$, we apply secure table lookup to the multiplication table of $\mathbb{G}$. This construction works best for relatively small groups because the performance is polynomial in $n = |\mathbb{G}|$.

Let $\sigma_0 : [0, n) \to \mathbb{G}$ be an arbitrary encoding for $\mathbb{G}$ such that $\sigma_0(0) = 1$ is the identity element of $\mathbb{G}$. The multiplication table for $\mathbb{G}$ is then represented by a public matrix $X \in [0, n)^{n \times n}$ with $X_{i,j} = \sigma_0^{-1}(\sigma_0(i) * \sigma_0(j))$, for all $i, j \in [0, n)$.

For the secure representation of any $a \in \mathbb{G}$, we set $[\![a]\!]_{\mathbb{G}} = [\![\sigma_0^{-1}(a)]\!]_p$ for a fixed prime $p \geq n$. Thus, each group element is represented by a secret-shared integer in the range $[0, n)$, and in particular, we have $[\![1]\!]_{\mathbb{G}} = [\![0]\!]_p$. To implement the group operation securely we take $[\![a * b]\!]_{\mathbb{G}} = [\![u]\!]_p^T X [\![v]\!]_p$, where $[\![u]\!]_p$ and $[\![v]\!]_p$ are secret-shared unit vectors of length n corresponding to $[\![a]\!]_{\mathbb{G}}$ and $[\![b]\!]_{\mathbb{G}}$, respectively. A unit vector has one entry equal to 1, and all its other entries equal to 0. There are efficient protocols for converting $[\![i]\!]_p$, $0 \leq i < n$, to the corresponding unit vector $[\![e_i]\!]_p$ and back.

The secure equality test between $[\![a]\!]_{\mathbb{G}} = [\![i]\!]_p$ and $[\![b]\!]_{\mathbb{G}} = [\![j]\!]_p$ reduces to a secure equality test $[\![i = j]\!]_p$.

The secure conditional can also be implemented efficiently, given $[\![x]\!]_p$ with $x \in \{0, 1\}$. Given $[\![a]\!]_{\mathbb{G}} = [\![i]\!]_p$ and $[\![b]\!]_{\mathbb{G}} = [\![j]\!]_p$, we have $[\![a^x b^{1-x}]\!]_{\mathbb{G}} = [\![x]\!]_p([\![i]\!]_p - [\![j]\!]_p) + [\![j]\!]_p$, hence the result is obtained with a single secure multiplication modulo p.

To generate a random group element, we generate a uniform random integer $[\![r]\!]_p$ with $r \in_R [0, n)$. This requires about $\log_2 n$ secure random bits (modulo p).

For the remaining tasks we can use the generic protocols presented later in this contribution. If desired, these protocols can also be optimized.

4.4.2 Secure Groups From Faithful Linear Representations

Our second generic construction of secure group schemes builds on the existence of faithful linear representations for arbitrary groups $\mathbb{G}$. A **(faithful) linear representation of** $\mathbb{G}$ is an (injective) homomorphism $\rho : \mathbb{G} \to GL_d(\mathbb{F}_q)$ for some finite field $\mathbb{F}_q$. The general existence of such linear representations can be inferred from Cayley's theorem, which states that every group $\mathbb{G}$ is isomorphic to a subgroup of the symmetric group acting on $\mathbb{G}$; clearly, every permutation can be represented by a unique permutation matrix over $\mathbb{F}_2$.

The basic idea is to use encoding $\sigma_\rho(A) = \rho^{-1}(A)$, defined for any $A \in \rho(\mathbb{G}) \subseteq GL_d(\mathbb{F}_q)$. Hence, we set $[\![a]\!]_{\mathbb{G}} = [\![\rho(a)]\!]_q$ such that group element a is represented by a secret-shared $d \times d$ matrix $A = \rho(a)$ with entries in $\mathbb{F}_q$. The secure group operation for $[\![a]\!]_{\mathbb{G}}$ and $[\![b]\!]_{\mathbb{G}}$ is implemented as a secure matrix product $[\![\rho(a)]\!]_q[\![\rho(b)]\!]_q$, which can be computed efficiently by performing d^2 secure dot products in parallel, using one round of communication. The secure inverse of $[\![a]\!]_{\mathbb{G}}$ can be computed efficiently by generating a matrix $[\![R]\!]_q$ with $R \in_R GL_d(\mathbb{F}_q)$

and opening $[\![\rho(a)]\!]_q[\![R]\!]_q$, inverting this matrix in the clear to obtain $\rho(a^{-1})R^{-1}$, and multiplying this with $[\![R]\!]_q$ to obtain the result. Here, R can be any matrix in $GL_d(\mathbb{F}_q)$, hence R does not have to lie in $\rho(\mathbb{G})$.

The secure equality test reduces to securely testing if $[\![\rho(a)-\rho(b)]\!]_q$ is the all-zero matrix. The secure conditional is implemented efficiently, given $[\![x]\!]_q$ with $x \in \{0,1\}$, by setting $[\![a^x b^{1-x}]\!]_{\mathbb{G}} = [\![x]\!]_q([\![\rho(a)]\!]_q - [\![\rho(b)]\!]_q) + [\![\rho(b)]\!]_q$, hence the result is obtained with a single secure multiplication of a scalar with a matrix over F_q.

The representation theory of finite groups [Ser77] helps us to find not just any linear representation, but rather (faithful) linear representations ρ of low degree d, preferably over a small finite field $\mathbb{F}_q$. As a simple example, we consider the representation of an arbitrary cyclic group of prime order p. These groups are all isomorphic to $(\mathbb{Z}_p, +)$, the group of integers modulo p with addition. To obtain a linear representation of degree 2 for this group, one takes

$$\rho : \mathbb{Z}_p \to GL_2(\mathbb{F}_p), \qquad i \mapsto \begin{pmatrix} 1 & i \\ 0 & 1 \end{pmatrix}.$$

The essential property is that $\rho(i)\rho(j) = \rho(i+j)$ for all $i, j \in \mathbb{Z}_p$. However, a linear representation of degree 1 is also possible: in fact, for a cyclic group of any order n, $n \geq 1$, let g be an element of order n in $\mathbb{F}_q^*$ for some prime power q. Then we simply take $\rho(i) = g^i$ for $i \in \mathbb{Z}_n$.

As a more advanced example, we illustrate the use of a linear representation of *minimum* degree for the well-known Rubik's Cube group. The Rubik's Cube group $\mathbb{G}$ is a subgroup of S_{48} generated by the six permutations corresponding to a clockwise turn of each side (keeping the center pieces at rest). The smallest faithful linear representation of the Rubik's Cube group turns out to be of degree 20 [HM16].

Concretely, linear representation $\rho : \mathbb{G} \to GL_{20}(\mathbb{F}_7)$ can be used, where $\rho(a)$ for $a \in \mathbb{G}$ corresponds with a generalized permutation matrix that encodes all 20 movable cubies as positions (12 edge and 8 corner cubies). Each position encodes an edge flip or a corner twist as elements in $\mathbb{F}_7$ of multiplicative order 2 and 3, respectively. We can define $\rho(a)$ as a block diagonal matrix over $\mathbb{F}_7$ with two blocks: a 12×12 generalized permutation matrix with its nonzero entries in $\{-1, 1\}$ and an 8×8 generalized permutation matrix with its nonzero entries in $\{2, 4, 1\}$. Moreover, the following conditions should be satisfied: for each block, the product of its nonzero entries is equal to 1 (modulo 7), and the permutations corresponding to the two blocks are of equal parity (i.e., both permutations are even, or both are odd). This results in $12!\, 2^{12}\, 8!\, 3^8/2/3/2 = 2^{10}3^7 8!12!$ possible representations, matching the order of the Rubik's Cube group.

Using secure random sampling for the Rubik's Cube group, we can thus replace the trusted shuffler in a Rubik's Cube competition.

4.5 Generic Protocols for Secure Groups

In Section 4.4 we have shown several representations for secure groups, and we have discussed protocols for implementing the tasks listed in Definition 6. In general, the implementation of basic tasks such as the group operation itself and en/decoding will strongly depend on the representation of the secure group. For other tasks, however, we may look for generic implementations.

This section presents generic protocols for the following four tasks from Definition 6: conditional (if-else), random sampling, inverse, and exponentiation.

4.5.1 Secure Conditional

It is usually best to implement the secure conditional directly in terms of the underlying representation for a secure group, as we have shown in Sections 4.4.1 and 4.4.2. Alternatively, the secure conditional can be evaluated in terms of secure exponentiation in either of these two generic ways, if applicable: as $[\![a]\!]_{\mathbb{G}}^{[\![x]\!]} * [\![b]\!]_{\mathbb{G}}^{1-[\![x]\!]}$, or as $([\![a]\!]_{\mathbb{G}}/[\![b]\!]_{\mathbb{G}})^{[\![x]\!]} * [\![b]\!]_{\mathbb{G}}$, where $x \in \{0,1\}$. This way it suffices to implement the basic operation $[\![a]\!]_{\mathbb{G}}^{[\![x]\!]}$ with

$x \in \{0, 1\}$. And if base a is publicly known, it even suffices to implement $a^{[\![x]\!]}$ (with Protocol 7 as a good option to implement this operation).

In pseudocode, the secure conditional will be denoted as $\mathsf{if\text{-}else}([\![x]\!], [\![a]\!]_{\mathbb{G}}, [\![b]\!]_{\mathbb{G}})$, with the obvious variations if a and/or b are publicly known. The condition $[\![x]\!]$ should always be secret.

4.5.2 Secure Random Sampling

The task of secure random sampling from a group $\mathbb{G}$ amounts to generating $[\![a]\!]_{\mathbb{G}}$ with $a \in_R \mathbb{G}$ (or, close to uniform), see Definition 6. A generic protocol is obtained by letting party $\mathcal{P}_i$ generate a random group element $a_i \in_R \mathbb{G}$ privately, and then use the secure group operation to form $[\![a]\!]_{\mathbb{G}} = \prod_i [\![a_i]\!]_{\mathbb{G}}$ for a sufficiently large subset of the parties. Alternatively, with knowledge of the structure of $\mathbb{G}$, we may try to generate $[\![a]\!]_{\mathbb{G}}$ in a more direct way. For example, if $\mathbb{G} = \langle g \rangle$ is a cyclic group of order n, a generic solution is to let the parties generate $[\![x]\!]$ with $x \in_R \mathbb{Z}_n$, and then set $[\![a]\!]_{\mathbb{G}} = g^{[\![x]\!]}$. This approach can be extended directly to abelian groups, given a generating set.

The problem of efficient sampling from nonabelian groups with a (close to) uniform distribution is much harder and has been studied extensively in the literature. We will follow Dixon [Dix08] to present the state of the art.

Specifically, Dixon focuses on generating ε-uniformly distributed random elements, meaning that each group element has probability $(1 \pm \varepsilon)/|\mathbb{G}|$ to appear as output. The notion of a *random cube* is an important building block:

Definition 7 (Random Cube [Dix08])**.** For $a_1, \ldots, a_j \in \mathbb{G}$, we define probability distribution $\mathit{Cube}(a_1, \ldots, a_j) = \{a_1^{\epsilon_1} \cdots a_j^{\epsilon_j} : \epsilon_1, \ldots, \epsilon_j \in_R \{0, 1\}\}$.

Given a generating set $\mathcal{G}$ of $\mathbb{G}$, the following theorem shows that we can construct a random cube of length proportional to $\log |\mathbb{G}|$ that is $1/4$-uniform with a given probability. The type of algorithm for generating the sequence of cubes is attributed to Cooperman.

Theorem 6 ([Dix08, Theorem 1])**.** *Let $\mathcal{G} = \{g_1, \ldots, g_d\}$ be a generating set of $\mathbb{G}$. Let $W_j = \mathit{Cube}(g_j^{-1}, \ldots, g_1^{-1}, g_1, \ldots, g_j)$ be a sequence of cubes, where for $j > d$, g_j is chosen at random from W_{j-1}. Then for each $\delta > 0$, there is a constant K_δ, independent of d or $\mathbb{G}$, such that with probability at least $1 - \delta$, distribution W_j is $1/4$-uniform when $j \geq d + K_\delta \log |\mathbb{G}|$.*

The number of group operations to construct the random element generator from Theorem 6 is proportional to $\log^2 |\mathbb{G}|$ and the average cost to produce successive random elements is proportional to $\log |\mathbb{G}|$.

Constant K_δ in Theorem 6 may still make the implementation impractical. However, the following theorem states that we can reduce the cube length if we start from a distribution W that is close to the uniform distribution U on $\mathbb{G}$ w.r.t. to the statistical (or, variational) distance $\Delta[W; U] = \frac{1}{2} \sum_{a \in \mathbb{G}} |W(a) - U(a)| = \max_{A \subset \mathbb{G}} |W(A) - U(A)|$.

Theorem 7 ([Dix08, Theorem 3(c)])**.** *Let U be the uniform distribution on $\mathbb{G}$ and suppose W is a distribution such that $\Delta[W; U] \leq \varepsilon < 1$. Let $a_0, \ldots, a_{j-1} \in \mathbb{G}$ be chosen independently according to distribution W. If $Z_j = \mathit{Cube}(a_0, \ldots, a_{j-1})$, then with probability at least $1 - 2^{-h}$, Z_j is 2^{-k}-uniform when*

$$j \geq \frac{h + 2k + 2 \log_2 |\mathbb{G}|}{1 - \log_2(1 + \varepsilon)}. \tag{4.3}$$

If W is ε-uniform, we have $\Delta[W; U] \leq \varepsilon/2$ for the uniform distribution U.

As an example we apply these results to random sampling from the Rubik's Cube group $\mathbb{G}$. Assume that we have generated a $1/4$-uniform cube W using Theorem 6. In practice, this means that we apply a statistical test to see if W is sufficiently large. Once this pre-processing step is completed we can apply Theorem 7 with $\varepsilon = 1/8$, to

Protocol 2 random($\mathbb{G}, x$) random cube $Z_j(\boldsymbol{g})$ for $\mathbb{G}$

```
1: $[\![\mathbf{r}]\!] \leftarrow$ random-bits($j$)
2: $[\![g_i^{xr_i}]\!]_{\mathbb{G}} \leftarrow$ if-else($[\![r_i]\!], g_i^x, 1_{\mathbb{G}}$), for $i = 0, \dots, j-1$
3: $[\![a^x]\!]_{\mathbb{G}} \leftarrow \prod_i [\![g_i^{xr_i}]\!]_{\mathbb{G}}$
4: return $[\![a^x]\!]_{\mathbb{G}}$        $\triangleright$ random $\mathbb{G}$-element to the power $x$, $x \in \mathbb{Z}$
```

Protocol 3 inverse($[\![a]\!]_{\mathbb{G}}$)

```
1: $[\![r]\!]_{\mathbb{G}} \leftarrow$ random($\mathbb{G}$)
2: $a * r \leftarrow [\![a]\!]_{\mathbb{G}} * [\![r]\!]_{\mathbb{G}}$
3: $[\![a^{-1}]\!]_{\mathbb{G}} \leftarrow [\![r]\!]_{\mathbb{G}} * (a * r)^{-1}$
4: return $[\![a^{-1}]\!]_{\mathbb{G}}$
```

generate, with probability at least $1 - 2^{-10}$, a new 2^{-k}-uniform cube Z_j of length $j \approx 0.83(10 + 2k + 2 \cdot 65.23)$ group elements. With $k = 20$, we then need a cube of 150 group elements.[1]

Protocol 2 is a general protocol for secure random sampling from a group $\mathbb{G}$. The protocol uses a given random cube Z_j of length j, which is assumed to be sufficiently close to uniform random. The protocol thus requires j secure random bits and $j - 1$ secure group operations. The result is raised to the power x, where $x = 1$ is intended as the default case, and this can be extended to several powers.

4.5.3 Secure Inverse

The sampling protocols from Section 4.5.2 allow us to invert secure group elements for groups with known or unknown order. Protocol 3 implements this functionality following the classical approach by [BIB89].

4.5.4 Secure Exponentiation

We start this section with a generally applicable protocol for secure exponentiation $[\![a^x]\!]_{\mathbb{G}}$ based on the binary representation of $[\![x]\!]$, see Protocol 4. The computational complexity is dominated by about 2ℓ group operations and ℓ calls to lsb. For the round complexity we see that the ℓ iterations of the loop are done sequentially. The protocol can be optimized in lots of ways. For instance, it is more efficient to compute the bits of $[\![x]\!]$ all at once, including the sign bit, using a standard solution.

For abelian groups the linear dependency on ℓ for the round complexity can be removed, as we show in Protocol 5. The improvement is that the ℓ iterations of the loop can now be done in parallel. Using standard techniques for constant rounds protocols (see, e.g., [BIB89, DFK+06]) the overall round complexity can be made independent of ℓ.

These protocols can be optimized if either a or x is public. For instance, if a is public, the list of powers $a, a^2, a^4, \dots$ can be computed locally, and if x is public, one can use techniques such as addition chains.

However, if one of the two inputs a or x is public, we can obtain efficient protocols by securely randomizing the other input. Protocol 6 solves the case that x is public, assuming that the group is abelian. The protocol uses secure random sampling from $\mathbb{G}$ to obtain both a random group element and its $(-x)$th power.

Protocol 7 solves the case that a is public. The protocol random-pair(a) outputs a random exponent $[\![r]\!]$ together with $[\![a^{-r}]\!]_{\mathbb{G}}$ for public input $a \in \mathbb{G}$. For public output a^x, we can also use public a^{-r} in the first step of the

[1]The minimum number of operations to scramble the Rubik's cube corresponds to the formal notion of the mixing time of the Rubik's cube group, which to our knowledge is an open problem. [CH19] proved that for the n-by-n puzzle, a generalization of the 15-puzzle, the mixing time is bounded by $n^4 \log n$. This is asymptotically larger than the minimum number of moves to solve the puzzle, $5n^3$ [Par95]. Exact mixing time for the 3x3x3 Rubik's cube is unknown. The number of quarter-turn operations to solve the Rubik's cube is 26.

Protocol 4 exponentiation($[\![a]\!]_{\mathbb{G}}, [\![x]\!]$)

```
1: $[\![b]\!]_{\mathbb{G}}, [\![y]\!] \leftarrow$ if-else($[\![x < 0]\!]$, (inverse($[\![a]\!]_{\mathbb{G}}$), $-[\![x]\!]$), ($[\![a]\!]_{\mathbb{G}}, [\![x]\!]$))    ▷ skip if $x \geq 0$ ensured
2: $[\![c]\!]_{\mathbb{G}} \leftarrow [\![1]\!]_{\mathbb{G}}$
3: for $i = 0$ to $\ell - 1$ do    ▷ $\mathrm{len}(x) \leq \ell$ assumed
4:     $[\![e_i]\!] \leftarrow$ lsb($[\![y]\!]$)
5:     $[\![c]\!]_{\mathbb{G}} \leftarrow$ if-else($[\![e_i]\!], [\![b]\!]_{\mathbb{G}} * [\![c]\!]_{\mathbb{G}}, [\![c]\!]_{\mathbb{G}}$)
6:     $[\![y]\!] \leftarrow ([\![y]\!] - [\![e_i]\!])/2$
7:     $[\![b]\!]_{\mathbb{G}} \leftarrow [\![b]\!]_{\mathbb{G}}^2$
8: return $[\![c]\!]_{\mathbb{G}}$
```

Protocol 5 exponentiation($[\![a]\!]_{\mathbb{G}}, [\![x]\!]$) $\mathbb{G}$ abelian

```
1: $[\![x_0]\!], \ldots, [\![x_{\ell-1}]\!] \leftarrow$ bits($[\![x]\!]$)    ▷ $\mathrm{len}(x) + 1 \leq \ell$ assumed for two's complement
2: $[\![r]\!]_{\mathbb{G}}, [\![r^{-1}]\!]_{\mathbb{G}}, \ldots, [\![r^{-2^{\ell-1}}]\!]_{\mathbb{G}} \leftarrow$ random($\mathbb{G}, 1, -1, \ldots, -2^{\ell-1}$)
3: $b \leftarrow [\![a]\!]_{\mathbb{G}} * [\![r]\!]_{\mathbb{G}}$    ▷ $b = a * r$
4: for $i = 0$ to $\ell - 1$ do
5:     $[\![c_i]\!]_{\mathbb{G}} \leftarrow$ if-else($[\![x_i]\!], b^{2^i} * [\![r^{-2^i}]\!]_{\mathbb{G}}, 1_{\mathbb{G}}$)    ▷ in parallel
6: return $[\![c_0]\!]_{\mathbb{G}} * \cdots * [\![c_{\ell-2}]\!]_{\mathbb{G}} * [\![c_{\ell-1}]\!]_{\mathbb{G}}^{-1}$
```

protocol. Finally, in the special case that a is an element of large order, parties may directly use their Shamir secret sharea, and perform Lagrange interpolation in the exponent to raise a to the power $[\![x]\!]$. The security of the protocol then relies on the discrete log assumption.

4.6 Specific Constructions of Secure Groups

4.6.1 Secure Quadratic Residue Groups

The first specific type of groups that we consider is $QR_p = (\mathbb{Z}_p^*)^2$, the group of quadratic residues modulo an odd prime p. Quadratic residue groups, and more generally subgroups of $\mathbb{F}_q^*$, are used widely in cryptography, where $n = |QR_p| = (p-1)/2$ is chosen sufficiently large to ensure that the discrete log problem is hard. Often, n is assumed to be prime as well.

Secure group schemes for $\mathbb{F}_q^*$ and all its subgroups are easily obtained using $[\![a]\!]_{\mathbb{G}} = [\![a]\!]_q$ as secure representation for $a \in \mathbb{F}_q^*$. Protocols for all tasks listed in Definition 6 can be obtained with standard techniques, except for the task of secure en/decoding. To enable integer-valued inputs and outputs for applications of secure groups, we often need encoding functions $\sigma\colon S \to \mathbb{G}$ with $S \subset \mathbb{Z}$. Efficient en/decoding is hard for arbitrary subgroups of $\mathbb{F}_q^*$, but for QR_p efficient solutions are possible.

Many encodings (or, embeddings) for $\mathbb{G} = QR_p$ have been proposed in the cryptographic literature. For our purposes, we consider the following four encodings for $\mathbb{G}$:

$$\begin{aligned}
&\sigma_1\colon \{1,\ldots,n\} \to \mathbb{G}, && s \mapsto s^2 \bmod p \\
&\sigma_2\colon \{1,\ldots,n\} \to \mathbb{G}, && s \mapsto \begin{cases} s, & \text{if } s \in \mathbb{G} \\ p - s, & \text{if } s \notin \mathbb{G} \end{cases} \\
&\sigma_3\colon \{0,\ldots,\lfloor p/k \rfloor - 1\} \to \mathbb{G}, && s \mapsto \min(\mathbb{G} \cap \{ks + i\colon 0 \leq i < k\}) \\
&\sigma_4\colon \{1,\ldots,n\} \to \mathbb{G}, && s \mapsto H(s, \arg\min_i\{i \geq 1\colon H(s,i) \in \mathbb{G}\}),
\end{aligned}$$

where $1 < k < p$ and H is a cryptographic hash function with codomain $\mathbb{Z}_p^*$.

Encoding σ_1 is the natural encoding for $\mathbb{G}$. Since squaring is a 2-to-1 mapping on $\mathbb{Z}_p^*$ as $s^2 = (-s)^2$, it follows that σ_1 is a bijective encoding on $\{1,\ldots,n\}$. Decoding of $a \in \mathbb{G}$ amounts to taking the unique modular square

Protocol 6 exponentiation($[\![a]\!]_{\mathbb{G}}, x$) public exponent x, $\mathbb{G}$ abelian

```
1: $[\![r]\!]_{\mathbb{G}}, [\![r^{-x}]\!]_{\mathbb{G}} \leftarrow \mathsf{random}(\mathbb{G}, 1, -x)$
2: $a * r \leftarrow [\![a]\!]_{\mathbb{G}} * [\![r]\!]_{\mathbb{G}}$
3: $[\![a^x]\!]_{\mathbb{G}} \leftarrow (a * r)^x * [\![r^{-x}]\!]_{\mathbb{G}}$
4: **return** $[\![a^x]\!]_{\mathbb{G}}$
```

Protocol 7 exponentiation($a, [\![x]\!]$) public base a

```
1: $[\![r]\!], [\![a^{-r}]\!]_{\mathbb{G}} \leftarrow \mathsf{random\text{-}pair}(a)$
2: $x + r \leftarrow [\![x]\!] + [\![r]\!]$                ▷ $x + r$ should be sufficiently random
3: $[\![a^x]\!]_{\mathbb{G}} \leftarrow a^{x+r}[\![a^{-r}]\!]_{\mathbb{G}}$
4: **return** $[\![a^x]\!]_{\mathbb{G}}$
```

root of $a \leq n$.

For $p \equiv 3 \pmod 4$, σ_2 is another bijective encoding for $\mathbb{G}$, generalizing the encoding defined for safe primes p in [CS03, Section 4.2, Example 2]. We see that $\sigma_2(s) = p - s$ for $s \notin \mathbb{G}$ is indeed a quadratic residue modulo p because $p - s \equiv (-1)s \pmod p$ is the product of two quadratic nonresidues. Decoding of $a \in \mathbb{G}$ amounts to $\sigma_2^{-1}(a) = a$ if $a < p/2$ and $\sigma_2^{-1}(a) = p - a$ otherwise.

Encoding σ_3 resembles an encoding introduced by Koblitz in the context of elliptic curve cryptosystems [Kob87, Section 3.2]. The value of $\sigma_3(s)$ is well-defined as long as each interval $\{ks + i\colon 0 \leq i < k\}$ contains a quadratic residue. This can be ensured by picking k sufficiently large as a function of p. The classical result by Burgess [Bur63] implies that $k = \lceil \sqrt{p} \rceil$ ensures successful encoding for sufficiently large p (see also [Hum03]). Under the extended Riemann Hypothesis, Ankeny [Ank52] proved that the least quadratic nonresidue for prime p equals $O(\log^2 p)$.

In practice, however, we may set parameter $k = \lceil \log_2 p \rceil$ or a small multiple thereof.[2] In general, encoding σ_3 is not bijective. Decoding of any a in the range of σ_3 is simple as $\sigma_3^{-1}(a) = \lfloor a/k \rfloor$.

Finally, encoding σ_4 corresponds to a hash function used in certain elliptic curve signature schemes [BLS04, Section 3.2]. Since $\Pr[H(s,i) \in \mathbb{G}] = \frac{1}{2}$ in the random oracle model, computing $\sigma_4(s)$ requires two hashes and two Legendre symbols on average. The collision-resistance of H implies that the encoding will be “computationally injective” in the sense that it is infeasible to find $s \neq s'$ for which $\sigma_4(s) = \sigma_4(s')$. Due the one-wayness of H, however, decoding of any a in the range σ_4 amounts to an exhaustive search.

For our purposes, we are interested in secure computation of these encodings and decodings. We briefly compare the performance for the four encodings. A secure encoding $[\![\sigma_1(s)]\!]$ amounts to a secure squaring of $[\![s]\!]$, which is very efficient. Secure decoding $[\![\sigma_1^{-1}(a)]\!]$ requires taking the modular square root of a quadratic residue $[\![a]\!]$. This can be done efficiently by multiplying $[\![a]\!]$ with a uniformly random square $[\![r]\!]^2$, opening the result ar^2, taking a square root $a^{1/2}r$ (in the clear) and dividing this by $[\![r]\!]$. Finally, we need one secure comparison to make sure that the result is in $\{1, \ldots, n\}$.

Similarly, a secure encoding $[\![\sigma_2(s)]\!]$ amounts to securely evaluating the Legendre symbol $[\![(s \mid p)]\!]$. Since s is known to be nonzero, we simply multiply $[\![s]\!]$ with a uniformly random square $[\![r]\!]^2$ and also with $[\![1 - 2b]\!] = [\![(-1)^b]\!]$ for a uniformly random bit b, opening the result $sr^2(-1)^b$, for which we compute the Legendre symbol

[2]Probabilistic heuristics for small primes ($<$ 20 bits), suggest that the greatest number of consecutive quadratic nonresidues in the average-case is fit by $\log_2(p)$. Buell and Hudson [BH84] computed the lengths of the longest sequences of consecutive residues and nonresidues. By numerical analysis the authors find that the longest sequence of residues and nonresidues for a given prime p are fit very closely by $\log_2(p) + \delta$ with δ slightly larger than 1, which suggests a choice of k for an average-case selection of modulus p. A small data set from [Hum03] using smaller primes (`http://www.math.caltech.edu/people/hummel.html`) shows sequences of length $< 2(\log(p) + \delta)$ in worst-case. Heuristics for larger primes (say $\geq$ 256-bit) are computationally heavy and beyond the scope of this work.

Protocol 8 $\mathsf{add}([\![P_1]\!]_{\mathbb{G}}, [\![P_2]\!]_{\mathbb{G}})$

```
1: $([\![x_1]\!], [\![y_1]\!], [\![t_1]\!], [\![z_1]\!]) \leftarrow [\![P_1]\!]_{\mathbb{G}}$
2: $([\![x_2]\!], [\![y_2]\!], [\![t_2]\!], [\![z_2]\!]) \leftarrow [\![P_2]\!]_{\mathbb{G}}$
3: $[\![r_1]\!], [\![r_2]\!], [\![r_3]\!], [\![r_4]\!] \leftarrow [\![y_1]\!] - [\![x_1]\!], [\![y_2]\!] + [\![x_2]\!], [\![y_1]\!] + [\![x_1]\!], [\![y_2]\!] - [\![x_2]\!]$
4: $[\![r_1]\!], [\![r_2]\!], [\![r_3]\!], [\![r_4]\!] \leftarrow [\![r_1]\!][\![r_2]\!], [\![r_3]\!][\![r_4]\!], 2[\![z_1]\!][\![t_2]\!], 2[\![t_1]\!][\![z_2]\!]$    ▷ 4M
5: $[\![r_1]\!], [\![r_2]\!], [\![r_3]\!], [\![r_4]\!] \leftarrow [\![r_4]\!] + [\![r_3]\!], [\![r_2]\!] - [\![r_1]\!], [\![r_2]\!] + [\![r_1]\!], [\![r_4]\!] - [\![r_3]\!]$
6: $[\![x_3]\!], [\![y_3]\!], [\![t_3]\!], [\![z_3]\!] \leftarrow [\![r_1]\!][\![r_2]\!], [\![r_3]\!][\![r_4]\!], [\![r_1]\!][\![r_4]\!], [\![r_2]\!][\![r_3]\!]$    ▷ 4M
7: $[\![P_3]\!]_{\mathbb{G}} \leftarrow ([\![x_3]\!], [\![y_3]\!], [\![t_3]\!], [\![z_3]\!])$
8: **return** $[\![P_3]\!]_{\mathbb{G}}$
```

$z = (sr^2(-1)^b \mid p)$ in the clear. Then $[\![(s \mid p)]\!] = z[\![(-1)^b]\!]$. Secure decoding boils down to a secure comparison with public $(p-1)/2$, which is quite efficient.

A secure encoding $[\![\sigma_3(s)]\!]$ amounts to the secure evaluation of k Legendre symbols $[\![(ks + i \mid p)]\!]$ and finding the first $+1$ among these. With k of the order $\log_2 p$, this represents a considerable amount of work. However, secure decoding $[\![\sigma_3^{-1}(a)]\!] = [\![\lfloor a/k \rfloor]\!]$ is much more efficient, especially if k is a power of two.

Finally, σ_4 is not practical to compute obliviously. To obliviously select the smallest i such that $[\![(H(s,i) \mid p)]\!] = +1$ requires computing $[\![H(s,i)]\!]$ for $0 \leq i < |\mathbb{G}|$, which is not practical. Also, σ_4 does not allow efficient decoding due to the one-wayness of H. However, this encoding is still useful to map private inputs $s \in \{1, \ldots, n\}$ to (unique) secret-shared group elements $[\![\sigma_4(s)]\!]_{\mathbb{G}}$.

4.6.2 Secure Elliptic Curve Groups

Let $E(\mathbb{F}_q)$ denote the finite group of points on an elliptic curve group E over $\mathbb{F}_q$. For cryptographic purposes we often use a subgroup $\mathbb{G} \subseteq E(\mathbb{F}_q)$ of large prime order, such that the discrete log problem and the (decisional) Diffie-Hellman problem are hard in $\mathbb{G}$.

As secure representation of an affine point $P = (x, y) \in \mathbb{G}$ we take $[\![P]\!]_{\mathbb{G}} = ([\![x]\!], [\![y]\!])$, where it is left understood that x and y are secret-shared over $\mathbb{F}_q$. For efficient implementation of the secure group operation it is advantageous to use *complete* formulas, that is, group law formulas without any exceptions. Complete formulas are known for groups of prime order Weierstrass curves [RCB16] and Edwards curves [BL07, HWCD08].

As an example, the complete formula for the group operation $P_1 + P_2$ on a twisted Edwards curve is given by

$$\left(\frac{x_1 y_2 + y_1 x_2}{1 + d x_1 y_1 x_2 y_2}, \frac{y_1 y_2 - a x_1 x_2}{1 - d x_1 y_1 x_2 y_2} \right),$$

where $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ are arbitrary points on the curve. In particular, one can see that $(0, 1)$ acts as the identity element. This way, it is straightforward to compute $[\![P_1 + P_2]\!]_{\mathbb{G}}$ given $[\![P_1]\!]_{\mathbb{G}}$ and $[\![P_2]\!]_{\mathbb{G}}$. The number of secure operations over $\mathbb{F}_q$ is low, like at most 7 secure multiplications (and 2 secure inversions). The multiplicative depth is however at least 3.

Using twisted Edwards curves with extended coordinates (and $a = -1$), the result from Hisil et al. [HWCD08, Section 4.4] yields a multiplicative depth of 2 for secure point addition, performing four multiplications in parallel. Protocol 8 is based on their complete formula.

We illustrate en/decoding for secure elliptic curve groups with an example similar to encoding σ_3 of Section 4.6.1.

Consider a twisted Edwards curve $E(\mathbb{F}_p)$ with curve equation $E : ax^2 + y^2 = 1 + dx^2y^2$ for given $a, d \in \mathbb{F}_p$. For encoding input s given parameter k, we repeatedly set $x = sk + i$ for varying i and test if x corresponds to a valid point $(x, y) \in E(\mathbb{F}_p)$, which amounts to testing if $u = (1 - ax^2)/(1 - dx^2)$ is a quadratic residue modulo p. Then we define $\sigma(s) = (x, y)$, where y is a quare root of u modulo p.

Algorithm 2 compose(f_1, f_2) f_1, f_2 primitive, positive definite

```
    Input: $f_1 = (a_1, b_1, c_1)$ and $f_2 = (a_2, b_2, c_2)$ with $a_1 \leq a_2$
 1: $s \leftarrow (b_1 + b_2)/2$
 2: $d', x_1, y_1 \leftarrow \mathsf{xgcd}(a_1, a_2)$          ▷ $x_1 a_1 + y_1 a_2 = d' = \mathsf{gcd}(a_1, a_2)$
 3: $d, x_2, y_2 \leftarrow \mathsf{xgcd}(s, d')$             ▷ $x_2 s + y_2 d' = \mathsf{gcd}(s, d')$
 4: $v_1, v_2 \leftarrow a_1/d, a_2/d$
 5: $r \leftarrow (y_1 y_2 (s - b_2) - x_2 c_2) \bmod v_1$    ▷ integer division
 6: $a_3 \leftarrow v_1 v_2$
 7: $b_3 \leftarrow b_2 + 2 v_2 r$
 8: $c_3 \leftarrow (b_3^2 - \Delta)/(4a_3)$
 9: $f_3 = (a_3, b_3, c_3)$
10: **return** reduce($f_3$)
```

Algorithm 3 square(f) f primitive, positive definite

```
    Input: $f = (a, b, c)$
 1: $x_2 \leftarrow b^{-1} \bmod a$        ▷ modular inverse
 2: $r \leftarrow (x_2 c) \bmod a$         ▷ integer division
 3: $a_2 \leftarrow a^2$
 4: $b_2 \leftarrow b - 2ar$
 5: $c_2 \leftarrow (b_2^2 - \Delta)/(4a_2)$
 6: $f_2 = (a_2, b_2, c_2)$
 7: **return** reduce($f_2$)
```

Secure decoding amounts to recovering $s = \lfloor x/k \rfloor$. In general, secure decoding can be optimized if k is a power of 2. Moreover, if increment i is also available as an auxiliary secret-shared input $[\![i]\!]$, secure decoding may be implemented basically for free as $[\![s]\!] = ([\![x]\!] - [\![i]\!])/k$.

4.6.3 Secure Class Groups

In this section we focus on class groups of (orders in) *imaginary* quadratic fields, as these have been used most for applications in public key cryptography. We use $Cl(\Delta)$ to denote the *ideal* class group of an imaginary quadratic field with discriminant $\Delta < 0$. We also use $Cl(\Delta)$ to denote the isomorphic *form* class group of integral binary quadratic forms $f(x, y) = ax^2 + bxy + cy^2$ with discriminant $\Delta = b^2 - 4ac < 0$. These forms are written as $f = (a, b, c)$ with $a, b, c \in \mathbb{Z}$, where it is implied that f is **primitive**, that is, $\gcd(a, b, c) = 1$, and f is **positive definite**, that is $a > 0$.

Two forms f and g are **equivalent** if there exists a matrix $((\alpha, \beta), (\gamma, \delta)) \in SL_2(\mathbb{Z})$ such that $g(x, y) = f(\alpha x + \beta y, \gamma x + \delta y)$. A primitive, positive definite form f is called **reduced** if $|b| \leq a \leq c$, where $b \geq 0$ if $|b| = a$ or $a = c$. Any form is equivalent to a unique reduced form. For a reduced form $f = (a, b, c)$ we have that a is bounded above by $\sqrt{|\Delta|/3}$.

For the composition of two forms $f_1, f_2 \in Cl(\Delta)$ we will use the algorithm due to Shanks [Sha71], as presented by Cohen [Coh93, Algorithm 5.4.7]. We slightly adapt the algorithm, skipping some case distinctions that were introduced for efficiency, see Algorithm 2. Apart from the computationally nontrivial reduction performed in the final step of the algorithm, the resulting algorithm only requires two xgcd's and one integer division. This compares favorably with alternatives such as the classical composition algorithm by Dirichlet [LD63] (see also [Cox11, Lemma 3.2] and [Lon19, Algorithm 6.1.1]).

For the special case that a form f is composed with itself, the algorithm can be simplified significantly, see Algorithm 3. We will assume that Δ is odd, which ensures that $b \neq 0$ for all forms, as $\Delta \equiv b \pmod 2$.

```
Algorithm 4 inverse($f$)                                    $f$ primitive, positive definite
    Input: $f = (a, b, c)$
 1: if $b \neq a$ then
 2:     $b \leftarrow -b$
 3: return $f$
```

```
Algorithm 5 reduce($f$)                                     $f \in Cl(\Delta)$ primitive, positive definite
    Input: $f = (a, b, c)$
 1: for $i = 1$ to $\lceil \mathrm{len}(\Delta)/2 \rceil$ do        ▷ invariant $a > 0$ and $c > 0$
 2:     if $b > 0$ then $s_b \leftarrow 1$ else $s_b \leftarrow -1$
 3:     $j \leftarrow \mathrm{len}(b) - \mathrm{len}(a) - 1$          ▷ explain: norm for len, using $\mathrm{len}(b) < \mathrm{len}(\Delta) - i$
 4:     $m \leftarrow -s_b 2^j$                                ▷ compute $2^j$ almost for free with norm
 5:     if $s_b b > 2a$ then                                ▷ $|b| > 2a$
 6:         $f \leftarrow fT_m$                              ▷ $fT_m = (a, b + 2ma, m^2 a + mb + c)$
 7:     if $a > c$ then
 8:         $f \leftarrow fS$                                ▷ $fS = (c, -b, a)$
 9: $m \leftarrow \lfloor \frac{a-b}{2a} \rfloor$                                   ▷ integer division
10: $f \leftarrow fT_m$
11: if $a > c$ then
12:     $f \leftarrow fS$
13: if $(a + b)(a - c) = 0$ and $b < 0$ then               ▷ ensure $b \geq 0$ if $a = -b$ or $a = c$
14:     $b \leftarrow -b$                                    ▷ $f \leftarrow fS = fT_1$
15: return $f$
```

The inverse of a form f can be obtained easily, without a costly reduction, see Algorithm 4.

As secure representation of a form $f = (a, b, c) \in \mathbb{G}$ we define $[\![f]\!]_{\mathbb{G}} = ([\![a]\!]_p, [\![b]\!]_p, [\![c]\!]_p)$, for a sufficiently large prime p. The prime p should be sufficiently large such that the intermediate forms computed by Algorithm 2 do not cause any overflow modulo p (also accounting for the "headroom" needed for secure comparison and secure integer division). Using Protocol 1 for secure xgcd and a protocol for secure integer division, Algorithm 2 is then easily transformed into a protocol for the secure composition of forms. Similarly, Algorithm 3 is transformed into a protocol for secure composition of a form with itself, noting that a secure modular inverse is done by means a of secure xgcd (using only one of the Bezout coefficients). Algorithm 4 is transformed in a very simple and efficient protocol, replacing the if-then statement by a secure conditional: $[\![b]\!] \leftarrow \mathsf{if\text{-}else}([\![b \neq a]\!], [\![-b]\!], [\![b]\!])$.

To reduce a form $f = (a, b, c)$, the classical reduction algorithm by Lagrange (see [BV07, Algorithm 5.3]) runs in at most $2 + \lceil \log_2(a/\sqrt{\Delta}) \rceil$ steps [BV07, Theorem 5.5.4], each step requiring an integer division. Our main goal in devising a protocol for secure reduction is to avoid the expensive integer divisions as much as possible.

To this end, we take the binary reduction algorithm by [AF06, Algorithm 3] as our starting point. We minimize the number of comparisons by exploiting the invariant $a > 0$ and $c > 0$. The algorithm reduces forms by the following transformations in $SL_2(\mathbb{Z})$:

$$S = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \quad T_m = \begin{pmatrix} 1 & m \\ 0 & 1 \end{pmatrix}.$$

The total number of iterations of the main loop required to achieve $|b| \leq 2a$ is at most $\mathrm{len}(b)$, if $f = (a, b, c)$ is the given form. This follows from the fact that if $|b| \leq 2a$ does not hold yet, an iteration of the main loop will reduce $\mathrm{len}(b)$ by at least 1. We will ensure that $\mathrm{len}(b) \leq \mathrm{len}(\Delta)$, such that it suffices to run the main loop for $\mathrm{len}(\Delta)$ iterations, independent of the input. Note that we need to test $a > c$ in each iteration as well.

Algorithm 6 encode(s, $Cl(\Delta)$) s sufficiently small w.r.t. $|\Delta|$

```
 1: $n = s \cdot gap_\ell$
 2: $a \leftarrow n - 1 - (n \bmod 4)$
 3: repeat
 4:     $a \leftarrow a + 4$                                    ▷ $a \equiv 3 \pmod 4$
 5:     $b \leftarrow \Delta^{\frac{a+1}{4}} \bmod a$
 6: until $b^2 \equiv \Delta \pmod a$ and $\gcd(a, b) = 1$
 7: if $\Delta \not\equiv b \pmod 2$ then
 8:     $b \leftarrow a - b$
 9: $f \leftarrow (a, b, \frac{b^2-\Delta}{4a})$                ▷ $\Delta \equiv b^2 \pmod 4$
10: $d \leftarrow n - a$
11: return $f, d$
```

As an important optimization, we limit the number of iterations to $\mathrm{len}(\Delta)/2$ by noting that it suffices to reduce b until $\mathrm{len}(b) < \mathrm{len}(\Delta)/2$. This ensures that $|b| \le \sqrt{|\Delta|}$ after the main loop. If $|b| \le 2a$ does not hold yet, then it follows that $a < |b|/2 \le \sqrt{|\Delta|/4}$, and we only need one "normalization" step to ensure $|b| \le a$.

The result is presented as Algorithm 5. This algorithm avoids the integer division and multiple comparisons in the main loop of Lagrange's reduction algorithm, at the cost of three secure comparisons and two secure bit length computations in the main loop.

For secure encoding to class groups, a given integer s will be mapped to a form (a, b, c) by computing a as a simple function of s and setting b as the square root of Δ modulo $4a$, see Algorithm 6. We note that this encoding improves upon the encoding proposed by [Sch03] (compare to [Sch99] as well), which relies on using prime numbers for a. Our algorithm avoids the need for primality tests, which are dominating the computational cost for the encoding algorithm.

Let k be our parameter as above. Given a worst-case prime gap for an ℓ-bit discriminant, gap_ℓ, we avoid the need to return the distance by setting parameter $k = gap_\ell$ and mapping input a to a prime $a' = a \cdot k + i$ for some $0 \le i < k$. Algorithm 6 is similar to searching for candidates by incrementing i in encoding σ_3 of Section 4.6.1. After a successful encoding of a' per Algorithm 6 to a so-called prime form $f_{a'} = (a', b, c)$, we can discard the distance knowing that $a = \lfloor a'/gap_\ell \rfloor$.

For $|\Delta|$ an odd prime, the frequency of prime forms (a', b, c) approximately corresponds to the quadratic residuosity of a' modulo $|\Delta|$. We refer to [BV07, Proposition 3.4.5] for the exact frequency of prime forms.

We improve upon the encoding suggested by [Sch03]. We do not search for a prime a, which requires a costly primality test. Instead we take $a \equiv 3 \pmod 4$ and simply test whether $b^2 \equiv \Delta \pmod a$ for $b = \Delta^{\frac{a+1}{4}} \bmod a$. This condition will certainly hold if a is prime and Δ is a quadratic residue modulo a, because then we have $b^2 \equiv \Delta^{(a+1)/2} \equiv \Delta^{(a-1)/2}\Delta \equiv \Delta \pmod a$. Hence, the success rate will be no worse than for [Sch03].

Finally, even though class groups are abelian, random sampling is quite hard, because it is already hard to compute a generating set for $Cl(\Delta)$. Given a (sufficiently complete) generating set $\mathcal{G}$, without knowledge of the order $Cl(\Delta)$, the general idea is for $g_i \in \mathcal{G}$ and random $e_i \in \{1, \dots, |\Delta|\}$, for $i \in \{0, \dots, d-1\}$, to compute a random element $\prod_{i=0}^{d-1} g_i^{e_i}$, e.g., using results from [LP92].

4.7 MPC-based Threshold Cryptosystems

In this section we present several applications of secure groups in the context of threshold cryptosystems, using the well-known threshold ElGamal cryptosystem as a basic example [Ped91]. We focus on the case of security against passive adversaries, assuming an MPC setting with m parties and a corruption threshold of t, $0 \leq t < m/2$, cf. Section 4.1.

Let $\mathbb{G} = \langle g \rangle$ be a cyclic group of large prime order p. Given our protocols for secure groups, a simple $(t+1, m)$-threshold ElGamal cryptosystem is obtained as follows:

Distributed key generation. The parties generate $[\![x]\!]$ with $x \in_R \mathbb{Z}_p$, and use secure exponentiation to compute $h = g^x$. The parties keep private key $[\![x]\!]$ in shares and output public key h.

Encryption. Given message $M \in \mathbb{G}$, pick $u \in_R \mathbb{Z}_n$. The ciphertext for public key h is the pair $(g^u, h^u M)$.

Threshold decryption. Given ciphertext (A, B), the parties use $[\![x]\!]$ to compute A^x by secure exponentiation. The parties output message $M = B/A^x$.

The complexity of the protocols for distributed key generation and threshold decryption is entirely hidden in the underlying MPC framework supporting secure groups.

We now extend the threshold ElGamal cryptosystem noting that instead of using message $M \in \mathbb{G}$ in the clear we can also use $[\![M]\!]_{\mathbb{G}}$ in shares, just as we keep the private key $[\![x]\!]$ in shares, using our protocols for secure groups:

Encryption of shared message. Given message $[\![M]\!]_{\mathbb{G}}$, the parties generate $[\![u]\!]$ with $u \in_R \mathbb{Z}_p$, and output the pair $(g^{[\![u]\!]}, h^{[\![u]\!]}[\![M]\!]_{\mathbb{G}})$ as ciphertext for public key h. Here, secure exponentiation is used twice, either with public output (A, B) or with secret output $([\![A]\!]_{\mathbb{G}}, [\![B]\!]_{\mathbb{G}})$.

Threshold decryption to shared message. Given ciphertext (A, B), the parties compute $[\![A^x]\!]_{\mathbb{G}} = A^{[\![x]\!]}$ using secure exponentiation. The parties compute and keep message $[\![M]\!]_{\mathbb{G}} = B/[\![A^x]\!]_{\mathbb{G}}$ in shares. Similarly, the parties may decrypt a ciphertext $([\![A]\!]_{\mathbb{G}}, [\![B]\!]_{\mathbb{G}})$.

Combining these protocols we can do things like reencryption for another public key h': use threshold decryption to a shared message and then encryption under the new public key. We can also do a reencryption as follows:

Proxy reencryption. Assume the parties hold shares for two private keys $[\![x_1]\!]$ and $[\![x_2]\!]$, then they may compute $[\![x_1/x_2]\!]$ and use this with secure exponentiation to convert ciphertext (A, B) for public key $h_1 = g^{x_1}$ into ciphertext $(A^{[\![x_1/x_2]\!]}, B)$ for public key $h_2 = g^{x_2}$.

Proxy reencryption key. Assume the parties hold shares for two private keys $[\![x_1]\!]$ and $[\![x_2]\!]$, then they may compute and open $[\![x_1/x_2]\!]$ as a proxy reencryption key.

4.8 Implementation

The *secure groups scheme* is implemented in Python using the MPyC package [Sch18]. This implementation includes secure representations for various elliptic curve groups (Weierstrass, Edwards), the symmetric group, the group of quadratic residues and class groups. To demonstrate its applicability, we include demos for the threshold conversion protocol using quadratic residues and twisted Edwards curves, computations with class groups using Protocol 1 for xgcd, random sampling in the Rubik's cube group, the well-known Pinocchio ZK-SNARK [PHGR13] and the Trinocchio multiparty zero knowledge proof [SVdV16] based on Pinocchio. The secure groups extension to MPyC is available from `https://github.com/toonsegers/sec_groups` under the MIT license. The zero knowledge proof examples are part of the verifiable MPC extension to MPyC, which is available from `https://github.com/toonsegers/verifiable_mpc` under the MIT license.

4.9 Conclusion

Secure groups are a convenient and powerful abstraction to use finite groups in the MPC setting. With the scheme presented in this contribution, a protocol engineer can use time-tested protocols to perform (cryptographic) group operations in the MPC setting. We illustrate this by demonstrating how secure groups facilitate the extension of a classical threshold cryptosystem. Furthermore, this work includes an xgcd protocol and a protocol to reduce binary quadratic forms, which allow us to implement secure class groups.

Chapter 5

Snarky Ceremonies

Succinct non-interactive arguments of knowledge (SNARKs) have found numerous applications in the blockchain setting and elsewhere. The most efficient SNARKs require a distributed ceremony protocol to generate public parameters, also known as a structured reference string (SRS). This ceremony protocol is the weakest element in the whole setup procedure, and so a careful and systematic treatment of the procedure is necessary. The contributions of this section are two-fold:

- We give a security framework for non-interactive zero-knowledge arguments with a ceremony protocol.
- We revisit the ceremony protocol of Groth's SNARK [Bowe et al., 2017]. 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.

5.1 Introduction

Zero-knowledge proofs of knowledge [GMR85, BG93] allow to prove knowledge of a witness for some NP statement while not revealing any information besides the truth of the statement. The recent progress in zero-knowledge (ZK) Succinct Non-interactive Arguments of Knowledge (SNARKs) [Gro10,Lip12,PHGR13,DFGK14, Gro16] has enabled the use of zero-knowledge proofs in practical systems, especially in the context of blockchains [BCG$^+$14, KMS$^+$16, SBG$^+$19, BCG$^+$20].

Groth16 [Gro16] is the SNARK with the smallest proof size and fastest verifier in the literature, and it is also competitive in terms of prover time. Beyond efficiency, it has several other useful properties. Groth16 is rerandomizable [LCKO19], which is a desirable property for achieving receipt-free voting [LCKO19]. Simultaneously, it also has a weak form of simulation extractability [BKSV20] 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 [BMMV19]. Furthermore, Groth16 is attractive to practitioners due to the vast quantity of implementation and code auditing attention it has already received.

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 [BCTV14] contained a small typo which resulted in a bug that would allow an attacker to print unlimited funds in an undetectable manner [Gab19]. Some would use this example as a reason to avoid any SNARK with a trusted setup ceremony at all costs. And yet people are still not only using Groth16, but actively designing new protocols on top of it, potentially for the reasons listed above. Thus we believe that if this SNARK ceremony is going to be used anyways, it is important to spend as much time and effort on simplifying its description and verifying its security as possible.

The primary purpose of this contribution is to take a formal approach to proving the security of the Groth16 setup ceremony of Bowe, Gabizon, and Miers [BGM17] that is currently being used in practice. This setup ceremony has already been used by Zcash, Aztec protocol, Filecoin, Semaphore, Loopring, and Tornado Cash. 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 [BCG$^+$15, BGG17, ABL$^+$19] (see also [AFK$^+$20]) propose specialized multi-party computation protocols for SRS generation ceremonies. A common feature of these protocols is that they are secure if at least one of the parties is honest. However, these schemes are not robust in the sense that all parties must be fixed before the beginning of the protocol and be active throughout the whole execution. In other words if a single party goes offline between rounds then the protocol will not terminate. Bowe, Gabizon, and Miers [BGM17] showed that the latter problem could be solved if there is access to a random beacon — an oracle that periodically produces bitstrings of high entropy — which can be used to rerandomize the SRS after each protocol phase. Unfortunately, obtaining a secure random beacon is, by itself, an extremely challenging problem [KRDO17, CD17, BBBF18, HYL20, BDD$^+$20]. Secure solutions include unique threshold signatures [HMW18], which themselves require complex setup ceremonies as well as verifiable delay functions [BBBF18, Pie19, Wes19] 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 [GKM$^+$18] takes a different approach and directly constructs a SNARK where the SRS is updatable, that is, anyone can update the SRS and knowledge soundness and zero-knowledge are preserved if at least one of the updaters was honest.[2] Subsequent updatable SNARKS like Sonic [MBKM19], Marlin [CHM$^+$20], and PLONK [GWC19] have improved the efficiency of updatable SNARKs, but they are still less efficient than for example [Gro16]. Mirage [KPPS20] 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 contribution as it amplifies the benefits of a successfully conducted ceremony.

5.1.1 Our Contributions

Our key contributions 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 [MBKM19]. Our definitions take a game-based approach and in particular are less rigid than multi-party computation definitions. Our security notions say 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, whereas it is not clear that the random beacon could be easily 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 [BGM17] in our new security framework.[3] 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

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

[2]Note that one can independently prove subversion ZK [ABLZ17, Fuc18].

[3]In other words Groth16 is ceremonial for [BGM17].

is proven with respect to algebraic adversaries [FKL18] in the random oracle model. We require a single party to be honest in each phase of the protocol in order to guarantee knowledge soundness and subversion zero-knowledge holds unconditionally. Unlike [BGM17], 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 [BGM17] used a novel discrete logarithm argument 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 this work simplifies the widely used protocol of [BGM17] and puts it on firmer foundations.

5.1.2 Our Techniques

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 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 any relation 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 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 [BFS16].

Security proof of setup ceremony

We must prove subversion zero-knowledge and update knowledge-soundness. Subversion zero-knowledge follows from the previous work in [ABLZ17, Fuc18], 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 [FPS20] 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 [FKL18]. 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 refer elements constructed using group-based operations. In contrast to the generic group model, the representation of group elements is visible to $\mathcal{A}_{alg}$, and we must provide a formal reduction to any assumptions used (e.g. discrete logarithm).

Already the original AGM paper [FKL18] 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 values. We prove that this new assumption is very closely related (equivalent for dynamic groups under small change of parameters) to the q-discrete logarithm assumption. In the case with an honest SRS [FKL18] 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 as we see, the simplified environment reduces this complexity.

Revisiting the discrete logarithm argument

One of the key ingredients in the [BGM17] 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 [BGM17] proved only knowledge soundness of Π_{dl}. While proving the security of the setup ceremony, 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 it learn $G^{f(x_1,\ldots,x_n)}$.

We show that Π_{dl} 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 again use AGM to prove simulation-extractability. However, 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.

5.2 Preliminaries

PPT denotes probabilistic polynomial time, and DPT denotes deterministic polynomial time. The security parameter is denoted by λ. We write $y \xleftarrow{r} \mathcal{A}(x)$ when a PPT algorithm $\mathcal{A}$ outputs y on input x and uses random coins r. Often we neglect r for simplicity. If $\mathcal{A}$ runs with specific random coins r, we write $y \leftarrow \mathcal{A}(x; r)$. 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: random coins, its inputs (including ones from the oracles), and outputs[4]. We sometimes refer to the "transcript" implying only the public part of the view: that is interactions of $\mathcal{A}$ with oracles and the challenger.

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 $c_k = \sum_{(i,j)=(1,1);i+j=k+1}^{(n,n)} a_i b_j$ for $k \in \{1, \ldots, 2n-1\}$.

Bilinear Pairings. Let BGen be a bilinear group generator that takes as input a security parameter 1^λ and outputs a pairing 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. That is, $\hat{e}(G, H)$ is a generator of $\mathbb{G}_T$ and for any $a, b \in \mathbb{Z}_p$, $\hat{e}(G^a, H^b) = \hat{e}(G, H)^{ab}$. We call a group dynamic if BGen outputs uniformly distributed generators G, H

5.2.1 Algebraic Group Model with RO and Discrete Logarithm Assumptions

We will use the algebraic group model (AGM) [FKL18] to prove the security of Groth's SNARK. In AGM, we consider only algebraic algorithms that provide a linear explanation for each group element that they output. More precisely, if $\mathcal{A}_{alg}$ has so far received group elements $G_1, \ldots, G_n \in \mathbb{G}$ and outputs a group element $G_{n+1} \in \mathbb{G}$, then it has to also provide a vector of integer coefficients $\vec{C} = (c_1, \ldots, c_n)$ such that $G_{n+1} = \prod_{i=1}^n G_i^{c_i}$. We will use it in a pairing-based setting where we distinguish between group elements of $\mathbb{G}_1$ and $\mathbb{G}_2$. Formally, the set of algebraic coefficients $\vec{C}$ is obtained by calling the algebraic extractor $\vec{C} \leftarrow \mathcal{E}^{\mathsf{agm}}_\mathcal{A}(\mathsf{view}_\mathcal{A})$ that is guaranteed to exist for any algebraic adversary $\mathcal{A}$. This extractor is white-box and requires $\mathcal{A}$'s view to run.

Random Oracle. Fuchsbauer et al. [FKL18] also show how to integrate the AGM with the random oracle (RO) model. Group elements returned by $\mathsf{RO}(\phi)$ are added to the set of received group elements. To simulate update proofs we make use of a weakening of the programmable RO model that we refer to as a transparent RO, presented on Fig. 5.1. For convenience we will denote $RO(\cdot) := RO_0(\cdot)$. The simulator has access to $\mathsf{RO}_1(\cdot)$ and can learn the discrete logarithm r by querying $\mathsf{RO}_1(x)$. It could query $\mathsf{RO}_0(x)$ for G^r but can also compute this value itself. Constructions and the $\mathcal{A}$ in all security definitions only have access to the restricted oracle $\mathsf{RO}_0(\cdot)$.

One remarkable detail in using white-box access to the adversary $\mathcal{A}$ in the RO model is that $\mathsf{view}_\mathcal{A}$ 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 sufficient for our proofs, even though we do not give any explicit access to the RO history besides the view of the adversary to the extractor.

[4]The latter can be derived from the former elements of the list, and is added to $\mathsf{view}_\mathcal{A}$ for convenience, following e.g. [GM17]

```
$\mathsf{RO}_t(\phi)$ // Initially $Q_{\mathsf{RO}} = \emptyset$
if $Q_{\mathsf{RO}}[\phi] \neq \bot$
  $r \leftarrow Q_{\mathsf{RO}}[\phi]$;
else
  $r \leftarrow\$ \mathbb{Z}_p; Q_{\mathsf{RO}}[\phi] \leftarrow r$
if $t = 1$ then return $r$ else return $G^r$
```

Figure 5.1: 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 $\mathsf{RO}(\phi)$ for the interface $\mathsf{RO}_0(\phi)$ provided to protocols.

Assumptions. We recall the (q_1, q_2)-discrete logarithm assumption [FKL18].

Definition 8 ((q_1, q_2)**-dlog).** The (q_1, q_2)-discrete logarithm assumption holds for BGen if for any PPT $\mathcal{A}$, the following probability is negligible in λ,

$$\Pr\left[\mathsf{bp} \leftarrow \mathsf{BGen}(1^\lambda); z \leftarrow\$ \mathbb{Z}_p; z' \leftarrow \mathcal{A}(\mathsf{bp}, \{G^{z^i}\}_{i=1}^{q_1}, \{H^{z^i}\}_{i=1}^{q_2}) : z = z'\right].$$

In our main theorem it is more convenient to use a slight variation of the above assumption.

Definition 9 ((q_1, q_2)**-edlog).** The (q_1, q_2)-*extended* discrete logarithm assumption holds for BGen if for any PPT $\mathcal{A}$, the following probability is negligible in λ,

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

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 prove in the following theorem.

Theorem 8. *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 $r, s \leftarrow\$ \mathbb{Z}_p$ 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 $i = 1, \ldots, q_1$ 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. □

We also state two lemmas that are often useful in conjunction with AGM proofs.

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

Lemma 2 ([BFL20]). *Let Q be a non-zero polynomial in $\mathbb{Z}_p[X_1, \dots, X_n]$ of total degree d. Define $Q'(Z) := Q(R_1Z + S_1, \dots, R_nZ + S_n)$ in the ring $(\mathbb{Z}_p[R_1, \dots, R_n, S_1, \dots, S_n])[Z]$. Then the coefficient of the highest degree monomial in $Q'(Z)$ is a degree d polynomial in $\mathbb{Z}_p[R_1, \dots, R_n]$.*

Lemma 3 (Schwartz-Zippel). *Let P be a non-zero polynomial in $\mathbb{Z}_p[X_1, \dots, X_n]$ of total degree d. Then, $\Pr[x_1, \dots, x_n \leftarrow\$ \mathbb{Z}_p : P(x_1, \dots, x_n) = 0] \leq d/p$.*

5.3 Ceremonial SNARKs

In this section, we put forward 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.

Compared to standard MPC definitions, our definitions do not include a simulator that can manipulate the final SRS to look 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 [GKM$^+$18] to ceremonies that require more than one round.

An argument system Ψ (with a ceremony protocol) for a relation $\mathcal{R}$ contains the following algorithms:

(i) A PPT parameter generator Pgen that takes the security parameter 1^λ as input and outputs a parameter p (e.g., a pairing description)[6]. We assume that $\mathsf{p} \leftarrow \mathsf{Pgen}(1^\lambda)$ and the security parameter is given as input to all algorithms without explicitly writing it.

(ii) A PPT SRS update algorithm Update that takes as input a phase number $\varphi \in \{1, \dots, \varphi_{max}\}$, the current SRS srs, and proofs of previous updates $\{\rho_i\}_i$, and outputs a new SRS srs' and an update proof ρ'. It is expected that Update itself forces a certain phase order, e.g. the sequential one.

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

(iv) A PPT prover algorithm Prove that takes as an input a SRS srs, a statement ϕ, and a witness w, and outputs a proof π.

(v) A DPT verification algorithm Verify that takes as an input a SRS srs, a statement ϕ, and a proof π, and outputs 0 or 1.

(vi) A PPT simulator algorithm Sim that takes as an input a SRS srs, a trapdoor τ, and a statement ϕ, and outputs a simulated proof π.

The description of Ψ also fixes a default $\mathsf{srs}^\mathsf{d} = (\mathsf{srs}^\mathsf{d}_1, \dots, \mathsf{srs}^\mathsf{d}_{\varphi_{max}})$.

We require that a secure Ψ 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 potential access to RO, if some sub-components of Ψ require it.

Completeness of Ψ requires that Update and Prove always satisfy verification.

Definition 10 (Perfect Completeness). An argument Ψ for $\mathcal{R}$ is *perfectly complete* if for any adversary $\mathcal{A}$, it has the following properties:

[6]We do not allow to subvert p in the context of this contribution but in real life systems also this part of the setup should be scrutinized. This is arguable easier since usually p is trapdoor free.

1. Update completeness:

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

2. Prover completeness:

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

Our definition of subversion zero-knowledge follows [ABLZ17]. Intuitively it says that an adversary that outputs a well-formed SRS knows the simulation trapdoor τ 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 τ makes sense only for an efficient adversary) and into an unbounded distinguisher $\mathcal{A}$. We let $\mathcal{Z}$ communicate with $\mathcal{A}$ with a message st.

Definition 11 (Subversion Zero-Knowledge (sub-ZK))**.** An argument Ψ for $\mathcal{R}$ is *subversion zero-knowledge* if for all PPT subverters $\mathcal{Z}$, there exists a PPT extractor $\mathcal{E}_{\mathcal{Z}}$, such that for all (unbounded) $\mathcal{A}$, $|\varepsilon_0 - \varepsilon_1|$ is negligible in λ, where

$$\varepsilon_b := \Pr\left[\begin{array}{l}(\mathsf{srs}, \{\rho_i\}_i, st) \leftarrow \mathcal{Z}(1^\lambda), \tau \leftarrow \mathcal{E}_{\mathcal{Z}}(\mathsf{view}_{\mathcal{Z}}) : \\ \mathsf{VerifySRS}(\mathsf{srs}, \{\rho_i\}_i) = 1 \wedge \mathcal{A}^{\mathcal{O}_b(\mathsf{srs}, \tau, \cdot)}(st) = 1\end{array}\right].$$

$\mathcal{O}_b$ is a proof oracle that takes as input $(\mathsf{srs}, \tau, (\phi, w))$ and only proceeds if $(\phi, w) \in \mathcal{R}$. If $b = 0$, $\mathcal{O}_b$ returns an honest proof $\mathsf{Prove}(\mathsf{srs}, \phi, w)$ and when $b = 1$, it returns a simulated proof $\mathsf{Sim}(\mathsf{srs}, \tau, \phi)$.

Bellare et al. [BFS16] 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 [GKM+18] can be seen as a relaxation of subversion soundness to overcome the impossibility result.

We generalize the notion of update knowledge soundness to multiple 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 VerifySRS. The adversary can query honest updates using UPDATE through a special oracle $\mathcal{O}_{\mathsf{srs}}$, described in Fig. 5.2. Eventually, adversary can propose some srs^*_φ with update proofs Q^* to be finalized through FINALIZE. 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 ϕ and a proof π. The adversary wins if $(\mathsf{srs}, \phi, \pi)$ passes verification, but there is no PPT extractor $\mathcal{E}_{\mathcal{A}}$ that could extract a witness even when given the view of $\mathcal{A}$.

Definition 12 (Update Knowledge Soundness)**.** An argument Ψ for $\mathcal{R}$ is update knowledge-sound if for all PPT adversaries $\mathcal{A}$, there exists a PPT extractor $\mathcal{E}_{\mathcal{A}}$ such that $\Pr[\mathsf{Game}_{\mathsf{uks}}^{\mathcal{A},\mathcal{E}_{\mathcal{A}}}(1^\lambda) = 1]$ is negligible in λ, where $\mathsf{Game}_{\mathsf{uks}}$ is defined as:

$$\mathsf{Game}_{\mathsf{uks}}^{\mathcal{A},\mathcal{E}_{\mathcal{A}}}(1^\lambda) := \left[\begin{array}{l}(\phi, \pi) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{srs}}(\cdot)}(1^\lambda); \text{get } (\mathsf{srs}, \varphi) \text{ from } \mathcal{O}_{\mathsf{srs}}; w \leftarrow \mathcal{E}_{\mathcal{A}}(\mathsf{view}_{\mathcal{A}}); \\ \textbf{return } \mathsf{Verify}(\mathsf{srs}, \phi, \pi) = 1 \wedge (\phi, w) \notin \mathcal{R} \wedge \varphi > \varphi_{max}\end{array}\right],$$

where SRS update oracle $\mathcal{O}_{\mathsf{srs}}$, constructing srs depending on interaction with $\mathcal{A}$, is described in Fig. 5.2.

If $\varphi_{max} = 1$, we obtain the standard notion of update knowledge soundness. In the rest of the contribution, 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

```
$\mathcal{O}_{\mathsf{srs}}(\mathsf{intent}, \mathsf{srs}^*, Q^*)$  // Initially $Q_1 = \cdots = Q_{\varphi_{max}} = \emptyset; \varphi = 1$
if $\varphi > \varphi_{max}$ : return $\perp$;  // SRS already finalized for all phases
$\mathsf{srs}_{\mathsf{new}} \leftarrow (\mathsf{srs}_1, \ldots, \mathsf{srs}_{\varphi-1}, \mathsf{srs}^*_{\varphi}, \ldots, \mathsf{srs}^*_{\varphi_{max}})$;
if $\mathsf{VerifySRS}(\mathsf{srs}_{\mathsf{new}}, Q^*) = 0$ : return $\perp$;  // Invalid SRS
if $\mathsf{intent} = \text{UPDATE}$ :
  $(\mathsf{srs}', \rho') \leftarrow \mathsf{Update}(\varphi, \mathsf{srs}_{\mathsf{new}}, Q^*); Q_{\varphi} \leftarrow Q_{\varphi} \cup \{\rho'\}$;
  return $(\mathsf{srs}', \rho')$;
if $\mathsf{intent} = \text{FINALIZE} \wedge Q_{\varphi} \cap Q^* \neq \emptyset$ :
    Assign $\mathsf{srs}_{\varphi} \leftarrow \mathsf{srs}^*_{\varphi}; \varphi \leftarrow \varphi + 1$;
```

Figure 5.2: SRS update oracle $\mathcal{O}_{\mathsf{srs}}$ given to the adversary in Definition 12. UPDATE returns $\mathcal{A}$ an honest update for φ, and FINALIZE finalizes the current phase. Current phase φ and current SRS srs (created in FINALIZE and stored up to φ) are shared with the KS challenger. $\{Q_{\varphi_i}\}_i$ is a local set of proofs for honest updates, one for each phase.

depends on the concrete relation. We leave it as an open question whether ceremony protocols with $\varphi_{max} > 2$ can provide any additional benefits.

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 that is necessary is the presence of a single honest update in the chain. The default srs $\mathsf{srs}^{\mathsf{d}}$ 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.

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

5.4 Proofs of Update 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 [BGM17]. 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, 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.

5.4.1 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. 5.1.

Because of how this NIZK proof of knowledge is used in our bigger 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 prove system more freely with other protocols while still allowing the adversary to extract. This is somewhat similar to idea of

```
$\mathcal{O}_{se}(\phi)$
// Initially $Q = \emptyset$
$\pi \leftarrow \mathsf{Sim}^{\mathsf{RO}_1(\cdot)}(\phi)$
$Q \leftarrow Q \cup \{(\phi, \pi)\}$
return $\pi$
```

```
$\mathcal{O}^{\mathbb{G}_1}_{\mathsf{poly}}(f(Z_1, \ldots, Z_{d(\lambda)}))$
if $\mathsf{deg}(f) > d(\lambda)$
  return $\perp$
else return $G^{f(z_1,\ldots,z_{d(\lambda)})}$
```

```
$\mathcal{O}^{\mathbb{G}_2}_{\mathsf{poly}}(g(Z_1, \ldots, Z_{d(\lambda)}))$
if $\mathsf{deg}(g) > d(\lambda)$
  return $\perp$
else return $H^{g(z_1,\ldots,z_{d(\lambda)})}$
```

Figure 5.3: Simulation-extraction oracle and two $d-$Poly oracles — for $\mathbb{G}_1$ and $\mathbb{G}_2$. All used in $\mathsf{Game}_{\mathsf{sSE}}$.

universal composability (UC), however, contrary to the basic UC, our extractor is still white-box. Another way would be to use an augmented UC model which allows white-box assumptions (see [KKK21]). In this work 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.[7] With this in mind, 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, $\mathsf{deg}(\prod_i Z_i^{a_i}) := \sum_i |a_i|$, and $\mathsf{deg}(f(Z_1, \ldots, Z_n)) = \mathsf{deg}(\sum_i f_i M_i) := \max\{\mathsf{deg}(M_i)\}$, where M_i are monomials of f. For example, $\mathsf{deg}(x^2\alpha\delta^{-2} + y) = 5$. 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, partially, to account for the SRS of the Groth's SNARK later on.

In addition, our ceremony protocol for Groth's SNARK requires NIZK to be straight-line simulation extractable. This means that knowledge soundness holds even when the adversary sees simulated proofs and extraction works without rewinding the adversary. It is important that the extractor's running time does not blow up if the adversary generates many different update proofs.

Below, we define such a NIZK in the random oracle model.

Let L be a language and $\mathcal{R}$ the corresponding relation. The argument Ψ for $\mathcal{R}$ in the random oracle model consists of the following PPT algorithms: the parameter generator Pgen, the prover $\mathsf{Prove}^{\mathsf{RO}(\cdot)}$, the verifier $\mathsf{Verify}^{\mathsf{RO}(\cdot)}$, and the simulator $\mathsf{Sim}^{\mathsf{RO}_1(\cdot)}$. We make an assumption that all algorithms get $\mathsf{p} \leftarrow \mathsf{Pgen}(1^\lambda)$ as an input without explicitly writing it.

We assume that an argument Ψ in the random oracle model satisfies the following definitions.

Definition 13. An argument Ψ for $\mathcal{R}$ is *perfectly complete* in the random oracle model, if for any adversary $\mathcal{A}$,

$$\Pr\left[(\phi, w) \leftarrow \mathcal{A}^{\mathsf{RO}(\cdot)}, \pi \leftarrow \mathsf{Prove}^{\mathsf{RO}(\cdot)}(\phi, w) : (\phi, w) \in \mathcal{R} \wedge \mathsf{Verify}^{\mathsf{RO}(\cdot)}(\phi, \pi) \neq 1\right] = 0.$$

Definition 14. An argument Ψ 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 $\mathcal{E}_\mathcal{A}$ such that $Pr[\mathsf{Game}^{\mathcal{A}}_{\mathsf{sSE}}(1^\lambda) = 1] = \mathsf{negl}(\lambda)$, where $\mathsf{Game}^{\mathcal{A}}_{\mathsf{sSE}}(1^\lambda) =$

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

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

The oracles $\mathcal{O}_{\mathsf{se}}, \mathcal{O}_{\mathsf{poly}}^{\mathbb{G}_1}, \mathcal{O}_{\mathsf{poly}}^{\mathbb{G}_2}$ are defined on Fig. 5.3.

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 $\mathcal{E}_{\mathcal{A}}$ 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 $\mathsf{Game}_{\mathsf{sSE}}$ with $\mathcal{O}_{\mathsf{poly}}$ removed, and thus is very close to the standard non-RO strong-SE variant.

Definition 15. An argument Ψ for $\mathcal{R}$ is *perfectly zero-knowledge* in the random oracle model if for all PPT adversaries $\mathcal{A}$, $\varepsilon_0 = \varepsilon_1$, where $\varepsilon_b := \Pr\left[\mathcal{A}^{\mathcal{O}_b(\cdot),\mathsf{RO}(\cdot)}(1^\lambda) = 1\right]$. $\mathcal{O}_b$ is a proof oracle that takes as an input (ϕ, w) and only proceeds if $(\phi, w) \in \mathcal{R}$. If $b = 0$, $\mathcal{O}_b$ returns an honest proof $\mathsf{Prove}^{\mathsf{RO}(\cdot)}(\phi, w)$ and when $b = 1$, it returns a simulated proof $\mathsf{Sim}^{\mathsf{RO}_1(\cdot)}(\phi)$.

Note that Sim is allowed to access RO in a transparent way, having access to the RO trapdoors, during simulation.

5.4.2 On the Security of BGM Update Proofs

We now prove that the proof system of [BGM17] satisfies this stronger property.

Bowe et al. [BGM17] 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 most secure 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

$$\mathcal{R}_{dl} = \{(\phi = (m, G^{y_1}, H^{y_2}), w) \mid y_1 = y_2 = w\},$$

where m is an auxiliary input that was used in the original [BGM17] proof of knowledge. The auxiliary input is redundant as we will see, but we still model it to have consistency with the original protocol.

The protocol is given formally in Fig. 5.4. First the prover queries the random oracle on the instance ϕ. 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 $\hat{e}(\pi, H) = \hat{e}(\mathsf{RO}(\phi), H^{y_2})$ which ensures knowledge of y_2. Intuition for the last equation is that $\mathsf{RO}(\phi)$ acts as a fresh random challenge for ϕ and the only way to compute $\pi = \mathsf{RO}(\phi)^{y_2}$ and H^{y_2} is by knowing y_2. The fact that in $\mathcal{R}_{dl}$ every ϕ with $y_1 = y_2$ belongs to $\mathcal{L}_{dl}$ (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".

$\mathsf{Prove}_{dl}^{\mathsf{RO}(\cdot)}(\phi, w)$	$\mathsf{Verify}_{dl}^{\mathsf{RO}(\cdot)}(\phi = (\cdot, G^{y_1}, H^{y_2}), \pi)$	$\mathsf{Sim}_{dl}^{\mathsf{RO}_1(\cdot)}(\phi = (\cdot, G^{y_1}, H^{y_2}))$
$G^r \leftarrow \mathsf{RO}(\phi)$; **return** G^{rw};	$G^r \leftarrow \mathsf{RO}(\phi)$; Verify that $\hat{e}(G^{y_1}, H) = (G, H^{y_2}) \wedge$ $\hat{e}(\pi, H) = \hat{e}(G^r, H^{y_2})$;	Assert $\hat{e}(G^{y_1}, H) = (G, H^{y_2})$; $r_\phi \leftarrow \mathsf{RO}_1(\phi)$; **return** $\pi \leftarrow (G^{y_1})^{r_\phi}$;

Figure 5.4: A discrete logarithm proof of knowledge Π_{dl} where $\mathsf{RO}_t(\cdot)$ denotes a random oracle.

Here we have moderately simplified the description from [BGM17] in the following ways:

- 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, and by doing the check within Π_{dl} the protocol is sound and zero-knowledge independently of its context.

We are now ready to prove the following theorem:

Theorem 9. *The argument* $\Pi_{dl} = (\mathsf{Prove}_{dl}^{\mathsf{RO}(\cdot)}, \mathsf{Verify}_{dl}^{\mathsf{RO}(\cdot)}, \mathsf{Sim}_{dl}^{\mathsf{RO}_1(\cdot)})$ *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* $\mathbb{G}_1$.

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. 5.4. When the simulator gets an input $\phi = (m, G^w, H^w)$ (note that $\phi \in \mathcal{L}$ by definition, so the exponent w is equal in G^w and H^w), it queries r for $G^r = \mathsf{RO}(\phi)$ 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 $\mathsf{Game}_{\mathsf{sSE}}$, and let us denote $\vec{z} = (z_1, \dots, z_{d(\lambda)})$. As $\mathcal{A}$ is algebraic, at the end of $\mathsf{Game}_{\mathsf{sSE}}$ it returns a statement and a proof (ϕ, π) such that $\phi = (m, G^{y'}, H^y)$ 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 $\phi \in \mathcal{L}$ (although does not affect the proof in any other way). For the elements H^y and π, $\mathcal{A}$ returns their representations $(\rho, b_1, \dots, b_{q_2})$ and $(\alpha, a_1, \dots, a_{q_1}, k_1, \dots k_{q_3}, p_1, \dots p_{q_4})$ that satisfy, correspondingly,

$$H^y = H^{\rho + b_1 g_1(\vec{z}) + \cdots + b_{q_2} g_{q_2}(\vec{z})} \tag{5.1}$$

and

$$\pi = G^{\alpha + a_1 f_1(\vec{z}) + \cdots + a_{q_1} f_{q_1}(\vec{z})} \cdot \prod_{j=1}^{q_3} K_j^{k_j} \cdot \prod_{j=1}^{q_4} P_j^{p_j} \tag{5.2}$$

In the former, ρ stands for the power of H, and b_i are linear coefficients of the polynomial evaluations returned by $\mathcal{O}_{\mathsf{poly}}^{\mathbb{G}_2}$. Similarly, for π, the representation is split into powers of the generator G, and coefficients of $\mathcal{O}_{\mathsf{poly}}^{\mathbb{G}_1}$, but it also accounts for the answers to hash queries $K_j, 1 \leq j \leq q_3$, and for the proof elements $P_j, 1 \leq j \leq q_4$, 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 $\mathsf{RO}(\phi) = G^{r^*}$ and $\mathsf{RO}(\phi_j) = G^{r_j}$ for $1 \leq j \leq q_3$. RO responses $\{G^{r_j}\}$, corresponding to the second set of elements $\{r_j\}$, exist in $\mathsf{view}_{\mathcal{A}}$ (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 $\mathsf{view}_{\mathcal{A}}$, 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 $\mathsf{Game}_{\mathsf{sSE}}$, $r^* \in \{r_j\}_{j \in [1,q_3] \setminus S}$. S is excluded from the set of indices, since $\mathcal{A}$ also must not query Sim on ϕ.

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_1^y 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 $\mathcal{O}^{\mathbb{G}_1}_{\mathsf{poly}}$, 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 $\mathsf{RO}(\phi)^y = \pi$. We thus have the two equations describing challenge values G^y and π, corresponding to Equations 5.1 and 5.2, 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 $\mathcal{E}_{\mathcal{A}}$ 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 ϕ; if successful, return k_j, and otherwise fail, returning 0. The intuition behind the extractor is the following. Since honest proofs are $\mathsf{RO}(\phi)^w$ 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.[8] 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 $\mathcal{E}_{\mathcal{A}}$ 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 $\mathcal{E}_{\mathcal{A}}$ only sees *interaction* with the oracle via $\mathsf{view}_{\mathcal{A}}$; therefore, it works only with G^{r_i} and S. Second, $\mathcal{E}_{\mathcal{A}}$ cannot compute exponent y right away merely from the algebraic representation of H^y passed as a part of ϕ. Even though the coefficients $(\rho, b_1, \ldots, b_{q_2})$ are available to $\mathcal{E}_{\mathcal{A}}$ 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 $\mathcal{E}_{\mathcal{A}}$ 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 $\mathcal{E}_{\mathcal{A}}$ fails to return a valid witness for $\mathcal{A}$. Thus if $\mathcal{A}$ has non-negligible advantage in the SE game with respect to $\mathcal{E}_{\mathcal{A}}$, 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 $(z_1, \ldots, z_d) \leftarrow\$ \mathbb{Z}_p$ 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 ϕ_j then $\mathcal{C}$ checks whether $(\phi_j, G^{ct_j+s_j}, (t_j, s_j)) \in Q_{\mathsf{RO}}$ and if yes returns $G^{ct_j+s_j}$.

  Otherwise $\mathcal{C}$ samples $t_j, s_j \leftarrow\$ \mathbb{Z}_p$, adds $(\phi_j, G^{ct_j+s_j}, (t_j, s_j))$ 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 $\phi_j = (m_j, G^{y_j}, H^{y_j})$ then its algebraic 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 $K_j = \mathsf{RO}(\phi_j)$ 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 $\mathcal{E}_{\mathcal{A}}$, knows $\vec{z}$ but not $ct_j + s_j$, thus the simulation strategy is different from Sim).

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

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

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 $\mathcal{E}_\mathcal{A}$ obtains, with the previous randomness now depending on the challenge c. Additionally we assume that $G^{r^*} = \mathsf{RO}(\phi)$ 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 ϕ 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:

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

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 $\mathcal{E}_\mathcal{A}$ 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 ϕ 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 $\mathcal{E}_\mathcal{A}$ succeeds. $\square$

5.5 Groth16 is Ceremonial

We show that Groth16 is ceremonial for a setup ceremony similar to the one proposed in [BGM17]. In this section, we start by giving an intuitive overview of the [BGM17] ceremony protocol. After that, we recall the Groth16 argument and carefully model the ceremony protocol in our security framework.

5.5.1 Ceremony Overview

We briefly remind the main idea of the [BGM17] ceremony protocol.

- The SRS contains elements of the form e.g. $(A_1, \dots, A_n, T) = (G^x, G^{x^2}, \dots, G^{x^n}, G^{\delta h(x)})$ where $t(X)$ is a public polynomial known to all parties, and x and δ are secret trapdoors.[9]
- Parties initialize the SRS to $(A_1, \dots, A_n, T) = (G, \dots, G, G)$.

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

- 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 $G^{t(x)}$ 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 $T_1^{\delta'}$. They must provide a proof of knowledge of δ'.

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

- 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 [BGM17] 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 5.4, 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 work 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.

5.5.2 Formal Description

We present the version of Groth's SNARK [Gro16] from [BGM17] and adjust the ceremony protocol to our security framework by defining Update and $\mathsf{VerifySRS}$ algorithms which follow the intuition of the previous section.

Firstly, let us recall the language of Groth's SNARK. A Quadratic Arithmetic Program (QAP) is described by a tuple

$$\mathsf{QAP} = \left(\mathbb{Z}_p, \{u_i(X), v_i(X), w_i(X)\}_{i=0}^m, t(X)\right)$$

where $u_i(X), v_i(X), w_i(X)$ are degree $n-1$ polynomials over $\mathbb{Z}_p$, and $t(X)$ is a degree n polynomial over $\mathbb{Z}_p$. Let the coefficients of the polynomials be respectively u_{ij}, v_{ij}, w_{ij}, and t_j. We can define the following relation for each QAP,

$$\mathcal{R}_{\mathsf{QAP}} = \left\{ (\phi, w) \left| \begin{array}{l} \phi = (a_0 = 1, a_1, \ldots, a_\ell) \in \mathbb{Z}_p^{1+\ell}, \\ w = (a_{\ell+1}, \ldots, a_m) \in \mathbb{Z}_p^{m-\ell}, \\ \exists h(X) \in \mathbb{Z}_p[X] \text{ of degree } \leq n-2 \text{ such that} \\ \left(\sum_{i=0}^m a_i u_i(X)\right)\left(\sum_{i=0}^m a_i v_i(X)\right) = \sum_{i=0}^m a_i w_i(X) + h(X)t(X) \end{array} \right. \right\}.$$

In particular, the satisfiability of any arithmetic circuit, with a mixture of public and private inputs, can be encoded as a QAP relation (see [GGPR13] for details).

Groth [Gro16] proposed an efficient SNARK for the QAP relation, which is now widely used in practice. Bowe et al. [BGM17] modified original argument's SRS to make it consistent with their distributed SRS generation protocol. The full description of the latter argument is in Fig. 5.5. For the intuition of the construction, we refer the reader to the original paper by Groth.

We adjust the SRS in Fig. 5.5 to our model of NIZK with a ceremony protocol: the default SRS, update algorithm, and a SRS specialization algorithm are described in Fig. 5.6. We obtain the default SRS from the trapdoor

$\underline{\mathsf{Setup}(\mathcal{R}_{\mathsf{QAP}})}$**:** Sample $\tau = (\alpha, \beta, \delta, x) \leftarrow\!\$\, (\mathbb{Z}_p^*)^4$ and return $(\mathsf{srs} = (\mathsf{srs}_u, \mathsf{srs}_s), \tau)$ s.t.

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

$$\mathsf{srs}_s \leftarrow \left(G^{\delta}, H^{\delta}, \{G^{\frac{\beta u_i(x)+\alpha v_i(x)+w_i(x)}{\delta}}\}_{i=\ell+1}^{m}, \{G^{\frac{x^i t(x)}{\delta}}\}_{i=0}^{n-2}\right).$$

$\underline{\mathsf{Prove}(\mathcal{R}_{\mathsf{QAP}}, \mathsf{srs}, \{a_i\}_{i=0}^{m})}$**:** Sample $r, s \leftarrow\!\$\, \mathbb{Z}_p$ and return $\pi = (G^A, H^B, G^C)$, where

$$A = \alpha + \textstyle\sum_{i=0}^{m} a_i u_i(x) + r\delta, \qquad B = \beta + \textstyle\sum_{i=0}^{m} a_i v_i(x) + s\delta,$$

$$C = \frac{\sum_{i=\ell+1}^{m} a_i(\beta u_i(x)+\alpha v_i(x)+w_i(x))+h(x)t(x)}{\delta} + As + Br - rs\delta.$$

$\underline{\mathsf{Verify}(\mathcal{R}_{\mathsf{QAP}}, \mathsf{srs}, \{a_i\}_{i=1}^{\ell}, \pi)}$**:** Parse π as (G^A, H^B, G^C) and verify that

$$\hat{e}(G^A, H^B) = \hat{e}(G^{\alpha}, H^{\beta}) \cdot \hat{e}(\prod_{i=0}^{\ell} G^{a_i(\beta u_i(x)+\alpha v_i(x)+w_i(x))}, H) \cdot \hat{e}(G^C, H^{\delta}).$$

$\underline{\mathsf{Sim}(\mathcal{R}_{\mathsf{QAP}}, \mathsf{srs}, \tau, \{a_i\}_{i=1}^{\ell})}$**:** Return (G^A, H^B, G^C), where

$$A, B \leftarrow\!\$\, \mathbb{Z}_p,\ C = \frac{AB - \alpha\beta - \left(\sum_{i=0}^{\ell} a_i(\beta u_i(x)+\alpha v_i(x)+w_i(x))\right)}{\delta}$$

Figure 5.5: Groth's zk-SNARK description.

$\tau = (1, 1, 1, 1)$. The algorithm Update samples new trapdoors and includes them to the previous SRS by exponentiation as was described in Section 5.5.1. For example, to update 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. However, when updating srs_u, we also derive a consistent srs_s using the Specialize algorithm 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 VerifySRS in Fig. 5.7, 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.

5.6 Security

We prove the security of Groth's SNARK from Section 5.5 in our NIZK with a ceremony framework of Section 5.3.

```
Default SRS: Run Setup in Fig. 5.5 with $\tau = (1,1,1,1)$ to obtain $\mathsf{srs}^{\mathsf{d}}$.

$\mathsf{Update}(\mathcal{R}_{\mathsf{QAP}}, \varphi \in \{1,2\}, (\mathsf{srs} = (\mathsf{srs}_u, \mathsf{srs}_s), Q))$:
 If $\varphi = 1$:
     1. Parse $\mathsf{srs}_u = \big(\{G_{x:i}, H_{x:i}\}_{i=0}^{2n-2}, \{G_{\alpha x:i}, G_{\beta x:i}, H_{\alpha x:i}, H_{\beta x:i}\}_{i=0}^{n-1}\big)$;
     2. Sample $\alpha', \beta', x' \leftarrow\$ \mathbb{Z}_p^*$;
     3. For $\iota \in \{\alpha, \beta, x\}$: $\pi_{\iota'} \leftarrow \mathsf{Prove}_{dl}^{\mathsf{RO}(\cdot)}(G^{\iota'}, H^{\iota'}, \iota')$;
     4. $\rho_{\alpha'} \leftarrow (G_{\alpha x:0}^{\alpha'}, G^{\alpha'}, H^{\alpha'}, \pi_{\alpha'})$;
     5. $\rho_{\beta'} \leftarrow (G_{\beta x:0}^{\beta'}, G^{\beta'}, H^{\beta'}, \pi_{\beta'})$;
     6. $\rho_{x'} \leftarrow (G_{x:1}^{x'}, G^{x'}, H^{x'}, \pi_{x'})$;
     7. $\rho \leftarrow (\rho_{\alpha'}, \rho_{\beta'}, \rho_{x'})$;
     8. $\mathsf{srs}'_u \leftarrow \Big(\{G_{x:i}^{(x')^i}, H_{x:i}^{(x')^i}\}_{i=0}^{2n-2}, \{G_{\alpha x:i}^{\alpha'(x')^i}, G_{\beta x:i}^{\beta(x')^i}, H_{\alpha x:i}^{\alpha(x')^i}, H_{\beta x:i}^{\beta(x')^i}\}_{i=0}^{n-1}\Big)$;
     9. $\mathsf{srs}'_s \leftarrow \mathsf{Specialize}(\mathsf{QAP}, \mathsf{srs}'_u)$;
    10. return $((\mathsf{srs}'_u, \mathsf{srs}'_s), \rho)$;
 If $\varphi = 2$:
     1. Parse $\mathsf{srs}_s \leftarrow \big(G_\delta, H_\delta, \{G_{sum:i}\}_{i=\ell+1}^m, \{G_{t(x):i}\}_{i=0}^{n-2}\big)$;
     2. Sample $\delta' \leftarrow\$ \mathbb{Z}_p^*$;
     3. $\pi_{\delta'} \leftarrow \mathsf{Prove}_{dl}^{\mathsf{RO}(\cdot)}(G^{\delta'}, H^{\delta'}, \delta')$;
     4. $\rho \leftarrow (G_\delta^{\delta'}, G^{\delta'}, H^{\delta'}, \pi_{\delta'})$;
     5. $\mathsf{srs}'_s \leftarrow \Big(G_\delta^{\delta'}, H_\delta^{\delta'}, \{G_{sum:i}^{1/\delta'}\}_{i=\ell+1}^m, \{G_{t(x):i}^{1/\delta'}\}_{i=0}^{n-2}\Big)$;
     6. return $((\mathsf{srs}_u, \mathsf{srs}'_s), \rho)$;
$\mathsf{Specialize}(\mathcal{R}_{\mathsf{QAP}}, \mathsf{srs}_u)$:        // Computes $\mathsf{srs}_s$ with $\delta = 1$
     1. Parse $\mathsf{srs}_u = \big(\{G_{x:i}, H_{x:i}\}_{i=0}^{2n-2}, \{G_{\alpha x:i}, G_{\beta x:i}, H_{\alpha x:i}, H_{\beta x:i}\}_{i=0}^{n-1}\big)$;
     2. $\mathsf{srs}_s \leftarrow \Big(G, H, \{\prod_{j=0}^{n-1} G_{\beta x:j}^{u_{ij}} \cdot G_{\alpha x:j}^{v_{ij}} \cdot G_{x:j}^{w_{ij}}\}_{i=\ell+1}^m, \{\prod_{j=0}^{n} G_{x:(i+j)}^{t_j}\}_{i=0}^{n-2}\Big)$;
     3. return $\mathsf{srs}_s$;
```

Figure 5.6: Default SRS and update algorithm for Groth's SNARK

Theorem 10 (Completeness)**.** *Groth's SNARK 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 8, Section 5.8.

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 $\mathsf{Update}(\mathsf{QAP}, \varphi, (\mathsf{srs}, \{\rho_i\}_i))$ 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 9, Section 5.8.

Prover completeness: Suppose that $\mathcal{A}$ output $(\mathsf{srs}, \{\rho_i\}_i, \phi, w)$ such that $(\phi, w) \in \mathcal{R}_{\mathsf{QAP}}$, 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 [Gro16]. □

Subversion zero-knowledge of Groth's SNARK was independently proven in [ABLZ17] and [Fuc18] under slightly different knowledge assumptions. The main idea is that VerifySRS checks that srs from $\mathcal{A}$ is well-

```
$\underline{\mathsf{VerifySRS}^{\mathsf{RO}(\cdot)}(\mathsf{QAP}, \mathsf{srs}, Q)}$:
  1. Parse $\mathsf{srs} = (\mathsf{srs}_u, \mathsf{srs}_s)$ and $Q = (Q_u, Q_s) = \{\rho_{u,i}\}_{i=1}^{k_u} \cup \{\rho_{s,i}\}_{i=1}^{k_s}$;
  2. Parse $\mathsf{srs}_u = \left(\{G_{x:i}, H_{x:i}\}_{i=0}^{2n-2}, \{G_{\alpha x:i}, G_{\beta x:i}, H_{\alpha x:i}, H_{\beta x:i}\}_{i=0}^{n-1}\right)$ and assert that elements belong to correct groups;
  3. For $i = 1, \ldots, k_u$:
     (a) Parse $\rho_{u,i} = (\rho_{\alpha'}^{(i)}, \rho_{\beta'}^{(i)}, \rho_{x'}^{(i)})$;
     (b) For $\iota \in \{\alpha, \beta, x\}$:
         i. Parse $\rho_{\iota'}^{(i)} = (G_{\iota}^{(i)}, G_{\iota'}^{(i)}, H_{\iota'}^{(i)}, \pi_{\iota'}^{(i)})$;
         ii. Assert $\mathsf{Verify}_{dl}^{\mathsf{RO}(\cdot)}(G_{\iota'}^{(i)}, H_{\iota'}^{(i)}, \pi_{\iota'}^{(i)}) = 1$;
         iii. If $i \neq 1$: Assert $\hat{e}(G_{\iota}^{(i)}, H) = \hat{e}(G_{\iota}^{(i-1)}, H_{\iota'}^{(i)})$;
  4. Assert $G_{x:1} = G_x^{(k_u)} \neq 1$; $G_{\alpha x:0} = G_\alpha^{(k_u)} \neq 1$; $G_{\beta x:0} = G_\beta^{(k_u)} \neq 1$;
  5. For $i = 1, \ldots, 2n-2$: Assert $\hat{e}(G_{x:i}, H) = \hat{e}(G, H_{x:i})$ and $\hat{e}(G_{x:i}, H) = \hat{e}(G_{x:(i-1)}, H_{x:1})$;
  6. For $i = 0, \ldots, n-1$ and $\iota \in \{\alpha, \beta\}$: Assert $\hat{e}(G_{\iota x:i}, H) = \hat{e}(G, H_{\iota x:i})$ and $\hat{e}(G_{\iota x:i}, H) = \hat{e}(G_{x:i}, H_{\iota x:0})$;
  7. Parse $\mathsf{srs}_s \leftarrow \left(G_\delta, H_\delta, \{G_{sum:i}\}_{i=\ell+1}^{m}, \{G_{t(x):i}\}_{i=0}^{n-2},\right)$ and assert that elements belong to correct groups;
  8. For $i = 1, \ldots, k_s$:
     (a) Parse $\rho_{s,i} = (G_\delta^{(i)}, G_{\delta'}^{(i)}, H_{\delta'}^{(i)}, \pi_{\delta'})$;
     (b) Assert $\mathsf{Verify}_{dl}^{\mathsf{RO}(\cdot)}(G_{\delta'}^{(i)}, H_{\delta'}^{(i)}, \pi_{\delta'}) = 1$;
     (c) if $i \neq 1$ assert $\hat{e}(G_\delta^{(i)}, H) = \hat{e}(G_\delta^{(i-1)}, H_{\delta'}^{(i)})$;
  9. Assert $\hat{e}(G_\delta, H) = \hat{e}(G, H_\delta)$ and $G_\delta = G_\delta^{(k_s)} \neq 1$;
  10. For $i = \ell+1, \ldots, m$: Assert $\hat{e}(G_{sum:i}, H_\delta) = \hat{e}(\prod_{j=0}^{n-1} G_{\beta x:j}^{u_{ij}} \cdot G_{\alpha x:j}^{v_{ij}} \cdot G_{x:j}^{w_{ij}}, H)$;
  11. For $i = 0, \ldots, n-2$: 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 5.7: SRS verification algorithm for Groth's SNARK

formed, and then one can use a knowledge assumption to recover the trapdoor from $\mathcal{A}$. If the trapdoor extraction is successful, it can be used to simulate proofs similarly to [Gro16]. Our approach only differs in that we recover the trapdoors from $\mathcal{R}_{dl}$ proofs of knowledge. We refer the reader to [ABLZ17,Fuc18] for details of the proof.

Theorem 11 (sub-ZK). *If Π_{dl} is a non-interactive proof of knowledge, then Groth's SNARK is subversion zero-knowledge.*

5.6.1 Update Knowledge Soundness

The main result of this section is the following update knowledge-soundness theorem.

Theorem 12. *Let us assume the $(2n-1, 2n-2)$-**edlog** assumption holds. Then Groth's SNARK has update knowledge soundness with respect to all PPT algebraic adversaries in the random oracle model.*

Proof. Let $\mathcal{A}$ be an algebraic adversary against update knowledge soundness. We denote the original update knowledge soundness game $\mathsf{Game}_{\mathsf{uks}}$ by Game_0. Given $\mathcal{A}$, we construct an explicit white-box extractor $\mathcal{E}_{\mathcal{A}}$ and prove that it succeeds with an overwhelming probability. The main theorem statement is thus $\mathsf{Adv}_{\mathcal{A},\mathcal{E}_{\mathcal{A}}}^{\mathsf{Game}_0}(\lambda) = \mathsf{negl}(\lambda)$.

$\underline{\mathcal{E}_{\mathcal{A}}(\mathsf{view}_{\mathcal{A}})}$

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

Figure 5.8: The extractor $\mathcal{E}_{\mathcal{A}}$ for update knowledge soundness

Description of the extractor $\mathcal{E}_{\mathcal{A}}$.

We present the extractor $\mathcal{E}_{\mathcal{A}}$ on Fig. 5.8. The extractor takes the adversarial view $\mathsf{view}_{\mathcal{A}}$ as input and extracts the AGM coefficients from an adversary $\mathcal{A}$ that produces a verifying proof. The goal of the extractor is to reconstruct the witness from this information (with an overwhelming probability, when verification succeeds).

The intuition behind its strategy is that, in Prove on Fig. 5.5, C is constructed as $\sum_i a_i(\alpha u_i(x) + \beta v_i(x) + w_i(x))/\delta$, 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 $(\phi, \pi = (A, B, C))$, 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 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.

We introduce the notion of the *critical* query — $i_{\mathsf{crit}_\varphi}$ corresponds to the last honest update $\mathcal{A}$ does in phase φ, which is included into the final SRS. Technically, we define it in the following way. For every phase, the final SRS comes with update proofs $\{\rho_{s,i}\}_{i=1}^{k_\phi}$ (included into Q^*) and at least one of them must be produced by honest update query for finalization to go through in that phase. Since these honest update proofs are outputs of the update oracle, on finalization of SRS, given Q^*, we can merely find the last such element $\rho_{s,i}$ in $Q^* \cap Q_\varphi$. Note that $i_{\mathsf{crit}_\varphi}$ is defined only after φ is finalized.

The extractor $\mathcal{E}_{\mathcal{A}}$, having access to $\mathsf{view}_{\mathcal{A}}$, can deduce $i_{\mathsf{crit}_\varphi}$, since $\mathsf{view}_{\mathcal{A}}$ includes $\mathcal{O}_{\mathsf{srs}}$ responses and Q^*. When $\mathcal{E}_{\mathcal{A}}$ obtains i_{crit_2}, it merely returns the AGM coefficients (which it can obtains 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 will show that $\mathcal{E}_{\mathcal{A}}$ indeed succeeds.

Description of Game_1.

We introduce Game_1, in Fig. 5.9, that differs from Game_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: at a later step we will build a reduction $\mathcal{B}$ that embeds the edlog challenge z into the trapdoors of the fresh SRS. For convenience, we describe Game_1 in terms of communication between the challenger $\mathcal{C}$ (top-level execution code of Game_1) and $\mathcal{A}$.

$\mathcal{C}$ of Game_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 to q_1 and q_2 correspondingly (polynomial upper bounds on number of queries for $\mathcal{A}$), in such a way attempting to guess

```
$\mathsf{Game}_1^{\mathcal{A},\mathcal{E}_\mathcal{A}}(1^\lambda)$
$\mathsf{srs} \leftarrow \mathsf{srs}^\mathsf{d}, \varphi = 1,$
$Q_1, Q_2 \leftarrow \emptyset; i_\mathsf{call} \leftarrow 0; i_{\mathsf{guess}_1} \leftarrow\$ [0, q_1]; i_{\mathsf{guess}_2} \leftarrow\$ [0, q_2]; \{z_\iota\}_{\iota\in\{x,\alpha,\beta,\delta\}} \leftarrow\$ \mathbb{Z}_p;$
Initialize $\mathsf{RO}_t(\cdot)$
$(\phi, \pi) \leftarrow \mathcal{A}^{\mathcal{O}_\mathsf{srs},\mathsf{RO}}$; $w \leftarrow \mathcal{E}_\mathcal{A}(\mathsf{view}_\mathcal{A});$
$\mathbf{return}\ \mathsf{Verify}(\mathsf{srs}, \phi, \pi) = 1 \wedge (\phi, w) \notin \mathcal{R} \wedge \varphi > \varphi_{max}$

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

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

Figure 5.9: Description of Game_1, a modified update KS game.

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 Game_0 for phase φ. $\mathcal{C}$ will generate fresh SRS for at most two (randomly picked) update queries through $\mathcal{O}_\mathsf{srs}$, 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 Game_1 (see Fig. 5.9).

Now, it is not possible for $\mathcal{C}$ to generate a fresh PoK as in Game_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 new one.

Therefore, it uses a specific technique to simulate the update proof of knowledge using the procedure $\mathsf{SimUpdProof}$ (see Fig. 5.9). The task of $\mathsf{SimUpdProof}$ 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 $(G^{\hat{\iota}\hat{\iota}'})^{\hat{\iota}^{-1}}$. This is the strategy of $\mathcal{C}$: it uses $\mathsf{view}_\mathcal{A}$ to extract the trapdoors ι_j for all the k_u proper updates that led to srs^*_φ, and thus to obtain $\hat{\iota}$. Notice that these updates can be both honest and adversarial, but, importantly, none of them are simulated (because we perform this procedure once per phase only), which guarantees that extraction succeeds. Next, $\mathsf{SimUpdProof}$ 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 $G^{\iota'}_{x:1}$, and for $\iota \in \{\alpha, \beta\}$ it is $G^{\iota'}_{\iota x:0}$), so we set the value to G^{z_ι}. The last element, proof-of-knowledge of $\hat{\iota}'$, we create by black-box simulation, since PoK is perfectly ZK. Namely, since we already have $\phi = (\perp, G^{\hat{\iota}'}, H^{\hat{\iota}'})$, we just pass it into Sim_{dl}, and attach the resulting $\pi_{\iota'}$ to the update proof. Although we know z_ι in Game_1 (and therefore know ϕ exponent $\hat{\iota}'$), it is not necessary to perform the reverse computation in Game_1 — technically, the procedure only requires G^{z_ι}. This is critical for the last part of our theorem, reduction to edlog, since in that case z_ι contains embedded edlog challenge, and we know only the corresponding group element, but not the exponent. Once again we emphasize that PoK simulation is not necessary in Game_1, since the discrete logarithm of ϕ is known; nevertheless, it is a good place to introduce it.

We now argue that the game Game_1 that we introduced is indistinguishable from Game_0 for $\mathcal{A}$. We recall that $(1,0)$-**dlog** assumption is implied by $(2n-1, 2n-2)$-**edlog** assumption.

Lemma 4. *Assuming $(1,0)$-**dlog**, the difference between advantage of $\mathcal{A}$ in winning Game_0 and Game_1 is negligible:* $\mathsf{Adv}^{\mathsf{Game}_0}_{\mathcal{A},\mathcal{E}_\mathcal{A}}(\lambda) \le \mathsf{Adv}^{\mathsf{Game}_1}_{\mathcal{A},\mathcal{E}_\mathcal{A}}(\lambda) + \mathsf{negl}(\lambda)$.

Proof. We introduce the intermediate game $\mathsf{Game}_{1/2}$, and prove the lemma in two steps, corresponding to the transitions between Game_0 and $\mathsf{Game}_{1/2}$, and between $\mathsf{Game}_{1/2}$ and Game_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 $\mathsf{Game}_{1/2}$, 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 $\phi = (\perp, G^{\hat{\iota}'}, H^{\hat{\iota}'})$, and simulate for this ϕ. Game_0 and $\mathsf{Game}_{1/2}$ are indistinguishable by perfect ZK of the PoK, thus $\mathsf{Adv}^{\mathsf{Game}_0}_{\mathcal{A},\mathcal{E}_\mathcal{A}}(\lambda) \le \mathsf{Adv}^{\mathsf{Game}_{1/2}}_{\mathcal{A},\mathcal{E}_\mathcal{A}}(\lambda) + \mathsf{negl}(\lambda)$. The formal reduction breaking ZK uses $\mathcal{O}_b$ (the real prover, or the simulator) in the critical queries; every other part of the game is the same.

Step 2. Next, we recall Game_1 which, compared to $\mathsf{Game}_{1/2}$, generates a fresh SRS with trapdoors $\{z_\iota\}_\iota$, and reconstructs ϕ 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 ϕ 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 $\mathsf{Game}_{1/2}$, the probability for PoK extractor to fail is negligible by SE. Therefore, $\mathsf{Game}_{1/2}$ is indistinguishable from Game_1: $\mathsf{Adv}^{\mathsf{Game}_{1/2}}_{\mathcal{A},\mathcal{E}_\mathcal{A}}(\lambda) \le \mathsf{Adv}^{\mathsf{Game}_1}_{\mathcal{A},\mathcal{E}_\mathcal{A}}(\lambda) + \mathsf{negl}(\lambda)$.

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 9 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 monomial, 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. $\square$

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 can 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 Game_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, we will use $\hat{Z}_\iota$ that is equal to the previously defined $Z_\iota(Z)$, but now as a function of Z_ι: $\hat{Z}_\iota = Z_\iota \prod \iota_j^{\mathcal{A}}$ for $\iota \in \{x, \alpha, \beta\}$, and $\hat{Z}_\delta = Z_\delta / \prod \delta_j^{\mathcal{A}}$.

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

1. Critical honest trapdoor variables: $Z_\alpha, Z_\beta, Z_x, Z_\delta$. The elements depending on them may be used by $\mathcal{A}$ independently, e.g. having G^{Z_x} in the first phase SRS $\mathcal{A}$ can set $A = G^{kZ_x + \cdots}$, but we expect that since final SRS contains $\hat{Z}_x$ as a trapdoor, k will contain adversarial trapdoor contribution too, that is $kZ_x = k'\hat{Z}_x$.
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 Game_1.
   - RO values for honest update proofs $\vec{R}_T = \{R_{T:i:\iota}\}_{i,\iota}$. First phase update number i corresponds to three values $R_{T:i:x}, R_{T:i:\alpha}, R_{T:i:\beta}$, and second phase update number j corresponds to $R_{T:j:\delta}$.
   - Direct RO values $\vec{R}_{\mathcal{A}}$. 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 of possible variants of $Q(\vec{\Psi})$ is quite big, 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 $\mathcal{A}$ success in Game_1. In other words, in order to win Game_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.

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

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: $\Pr[\mathsf{Game}_1^{\mathcal{A},\mathcal{E}_\mathcal{A}}(1^\lambda) = 1 \mid \textbf{sim}_i]$ is the same for all simulation strategies $\textbf{sim}_i$, including the lucky one.

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

Our choice of $\{i_{\mathsf{guess}_\varphi}\}_\varphi$, and thus the chosen simulation strategy $\textbf{sim}_i$ is independent from the success of $\mathcal{A}$. Note that 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 Game_1.

Therefore, when in Game_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:

$$Q(\vec{\Psi}) := \left(A(\vec{\Psi})B(\vec{\Psi}) - \hat{Z}_\alpha\hat{Z}_\beta - \sum_{i=0}^{\ell} a_i(\hat{Z}_\beta u_i(\hat{Z}_x) + \hat{Z}_\alpha v_i(\hat{Z}_x) + w_i(\hat{Z}_x)) - C(\vec{\Psi})\hat{Z}_\delta\right) \tag{5.3}$$

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. The function $Q(\vec{\Psi})$ reconstructs verification equation of the proof in this particular game execution: in particular, $Q(\vec{\psi}) = 0 \iff \mathsf{Verify}(\mathsf{srs}, \phi, \pi) = 1$.[10]

Note that the form of functions $A(\vec{\Psi}), B(\vec{\Psi})$, and $C(\vec{\Psi})$ still heavily 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.

Lemma 5. *The general form of $Q(\vec{\Psi})$ is as presented in Eq. (5.3). Moreover, A, B, C are Laurent polynomials in Ψ_2 when viewed over $Z_p[\vec{C}]$, where $\vec{C}$ are AGM coefficients, abstracted as variables. In another words, $A, B, C \in (Z_p[\vec{C}, \vec{\Psi}_1])[\vec{\Psi}_2]$ are Laurent. Therefore, Q also is Laurent when viewed as $(Z_p[\vec{C}, \vec{\Psi}_1])[\vec{\Psi}_2]$.*

Proof. We will first argue why the form of $Q(\vec{\Psi})$, and concretely its public elements that are included in it ($\hat{Z}_\alpha\hat{Z}_\beta$ for instance), is as in Eq. (5.3). Consider the first phase for now. When $\mathcal{A}$ finalizes srs_u we locate in Q^*_u

[10]The form of the public equation parts (the second and the third terms, and $1/Z_\delta$ in the last term) is due to our critical-step-simulation strategy, e.g. they only depend on the challenge variables plus last adversarial trapdoors. This is where guessing the *last* query really helps: otherwise these terms would also depend on some $\vec{T}$.

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

Figure 5.10: Description of Game_2, an extension of Game_1 with **bad** event introduced. $Q(\vec{\Psi}_1, \vec{\Psi}_2)$ 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 $\{Z_\iota\}$.

($Q^* = (Q^*_u, Q^*_s)$) the critical update proofs for x, α, β — let their position be $j \in [1, k_u]$ (j is not equal to the $\mathcal{O}_{\mathsf{srs}}$ *query index* i_{crit_1} since there can be many adversarial updates in Q^*_u). These update proofs are followed by (potentially non-empty) set of adversarial proofs with indices $j+1, \dots, k_u$ — 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 $G_{\alpha:0}$ in the final SRS corresponds to $Z_\alpha \prod \alpha_i^\mathcal{A}$. In step 3.b of SRS verification we do a cascade verification: in particular, on the $j+1$ step we check $\hat{e}(G^j_\alpha, H) = \hat{e}(G^{(j-1)}_\alpha, H^j_{\alpha'})$. So, if the exponent of $H^j_{\alpha'}$ is some $\alpha_j^\mathcal{A}$, then we know after this loop ends that $G_\alpha = G^{z_\alpha \prod \alpha_i^\mathcal{A}}$. Same is applicable for G_x, G_β. Then we can use other $\mathsf{VerifySRS}$ equations, similarly to the style in Lemma 8, 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). And we can argue 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 public part of $Q(\vec{\Psi})$:

$$\left\{ \hat{Z}_\alpha \hat{Z}_\beta, \sum_{i=0}^{\ell} a_i(\hat{Z}_\beta u_i(\hat{Z}_x) + \hat{Z}_\alpha v_i(\hat{Z}_x) + w_i(\hat{Z}_x)), \hat{Z}_\delta \right\}$$

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 $\mathcal{O}_{\mathsf{srs}}$ 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, $\mathsf{VerifySRS}$ 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 $\mathsf{VerifySRS}$, $\mathsf{VerifySRS}$ will merely produce linear combinations of E', which will be again Laurent in $\vec{\Psi}_2$. $\square$

Description of Game_2**.**

The following game, presented on Fig. 5.10 extends Game_1 with two additions. Firstly, it introduces the event **bad**. The condition that we are trying 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. Otherwise, we can argue that $\mathcal{A}$ used the specific value of z_ι, even though it is hidden in the exponent.

Secondly, we require that adversary wins only if the event **lucky** happens. Since lucky is an independent event, then $\Pr[\mathsf{Game}_2^{\mathcal{A},\mathcal{E}_\mathcal{A}}(1^\lambda) = 1] = \Pr[\mathsf{Game}_1^{\mathcal{A},\mathcal{E}_\mathcal{A}}(1^\lambda) = 1 \wedge \mathbf{lucky}] = \Pr[\mathsf{Game}_1^{\mathcal{A},\mathcal{E}_\mathcal{A}}(1^\lambda) = 1]/(q_1 q_2)$. The last transition is due to independence of winning Game_1 and **lucky** explained earlier ($\Pr[\mathsf{Game}_1^{\mathcal{A},\mathcal{E}_\mathcal{A}}(1^\lambda) = 1] = \Pr[\mathsf{Game}_1^{\mathcal{A},\mathcal{E}_\mathcal{A}}(1^\lambda) = 1 \mid \mathbf{lucky}]$). We can use the total probability formula to condition winning in Game_2 on the event **bad**.

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

The next two lemmas will upperbound this probability. The Lemma 6 will bound the the first term of the sum and the Lemma 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 $\mathcal{E}_\mathcal{A}$ is a QAP witness.

Lemma 6. *In* Game_2 *when* $\neg$**bad**, *if* $\mathcal{A}$ *produces a verifying proof,* $\mathcal{E}_\mathcal{A}$ *succeeds:*

$$\Pr[\mathsf{Game}_2^{\mathcal{A},\mathcal{E}_\mathcal{A}}(1^\lambda) = 1 \mid \neg\mathbf{bad}] = \mathsf{negl}(\lambda)$$

Proof. Assume $\mathsf{Verify}(\mathsf{srs}, \phi, \pi) = 1$, the event **lucky** happens since otherwise $\mathcal{A}$ cannot win Game_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. (5.3).

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 13. 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: Update 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 Update 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.

$\square$

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.

$$\begin{aligned}
A(\vec{\Psi}_2) = {} & A_0 + \sum_{i=1}^{2n-2} A_{1:i} Z_x^i + \sum_{i=0}^{n-1} (A_{2:i} Z_\alpha Z_x^i + A_{3:i} Z_\beta Z_x^i) + A_4 Z_\delta \\
& + \sum_{i=l+1}^{m} A_{5:i} \frac{\hat{Z}_\beta u_i(\hat{Z}_x) + \hat{Z}_\alpha v_i(\hat{Z}_x) + w_i(\hat{Z}_x)}{Z_\delta} + \sum_{i=0}^{n-2} A_{6:i} \frac{\hat{Z}_x^i t(\hat{Z}_x)}{Z_\delta} \\
& + \sum_{\iota} (A_{7:\iota} \frac{Z_\iota}{(\prod_{\mathcal{I}_1} T_{i,\iota})(\prod_{\mathcal{I}_2} \iota_i^{\mathcal{A}})} + A_{8:\iota} \frac{K_\iota Z_\iota}{(\prod_{\mathcal{I}_1} T_{i,\iota})(\prod_{\mathcal{I}_2} \iota_i^{\mathcal{A}})}) \\
B(\vec{\Psi}_2) = {} & B_0 + \sum_{i=1}^{2n-2} B_{1:i} Z_x^i + \sum_{i=0}^{n-1} (B_{2:i} Z_\alpha Z_x^i + B_{3:i} Z_\beta Z_x^i) + B_4 Z_\delta \\
& + \sum_{i,\iota} (B_{7:\iota} \frac{Z_\iota}{(\prod_{\mathcal{I}_1} T_{i,\iota})(\prod_{\mathcal{I}_2} \iota_i^{\mathcal{A}})})
\end{aligned}$$

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 SimUpdProof of Game$_1$. 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 [BGM17], 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 [BGM17]. 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 $Z_\alpha Z_x^i$:

$$\begin{aligned}(\textstyle\sum A_{2,i} Z_\alpha Z_x^i)(B_0 + \sum B_{1,i} Z_x^i) + (\sum A_{5,i} \hat{Z}_\alpha v_i(\hat{Z}_x)) B_4 + \\ (\textstyle\sum B_{2,i} Z_\alpha Z_x^i)(A_0 + \sum A_{1,i} Z_x^i) - \sum a_i \hat{Z}_\alpha v_i(\hat{Z}_x) - (\sum C_{5,i} \hat{Z}_\alpha v_i(\hat{Z}_x)) = 0\end{aligned}$$

For $Z_\beta Z_x^i$:

$$\begin{aligned}(\textstyle\sum A_{3,i} Z_\beta Z_x^i)(B_0 + \sum B_{1,i} Z_x^i) + (\sum A_{5,i} \hat{Z}_\beta u_i(\hat{Z}_x)) B_4 + \\ (\textstyle\sum B_{3,i} Z_\beta Z_x^i)(A_0 + \sum A_{1,i} Z_x^i) - \sum a_i \hat{Z}_\beta u_i(\hat{Z}_x) - (\sum C_{5,i} \hat{Z}_\beta u_i(\hat{Z}_x)) = 0\end{aligned}$$

And for Z_x^i:

$$\begin{aligned}(B_0 + \textstyle\sum B_{1,i} Z_x^i)(A_0 + \sum A_{1,i} Z_x^i) + (\sum A_{5,i} w_i(\hat{Z}_x) + \sum A_{6,i} \hat{Z}_x^i t(\hat{Z}_x)) B_4 - \\ \textstyle\sum a_i w_i(\hat{Z}_x) - \sum C_{5,i} w_i(\hat{Z}_x) - \sum C_{6,i} \hat{Z}_x^i t(\hat{Z}_x) = 0\end{aligned}$$

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.

$$\begin{aligned}
Z_\alpha^2 Z_x^i &: (\sum_{i=0}^{n-1} A_{2:i} Z_\alpha Z_x^i)(\sum_{i=0}^{n-1} B_{2:i} Z_\alpha Z_x^i) = 0 \implies \forall i \in [0, 2n-2] : \sum_{j,k:(0,0);j+k=i}^{(n-1,n-1)} A_{2:j} B_{2:k} = 0 \\
Z_\beta^2 Z_x^i &: (\sum_{i=0}^{n-1} A_{3:i} Z_\beta Z_x^i)(\sum_{i=0}^{n-1} B_{3:i} Z_\beta Z_x^i) = 0 \implies \forall i \in [0, 2n-2] : \sum_{j,k:(0,0);j+k=i}^{(n-1,n-1)} A_{3:j} B_{3:k} = 0 \\
Z_\alpha Z_\beta Z_x^i &: (\sum_{i=0}^{n-1} A_{2:i} Z_\alpha Z_x^i)(\sum_{i=0}^{n-1} B_{3:i} Z_\beta Z_x^i) + (\sum_{i=0}^{n-1} A_{3:i} Z_\beta Z_x^i)(\sum_{i=0}^{n-1} B_{2:i} Z_\alpha Z_x^i) = \alpha^{\mathcal{A}} \beta^{\mathcal{A}} (\neq 0) \\
Z_\alpha^2 Z_x^i / Z_\delta &: (\sum_{i=l+1}^{m} A_{5,i} \hat{Z}_\alpha v_i(\hat{Z}_x)/Z_\delta)(\sum_{i=0}^{n-1} B_{2,i} Z_\alpha Z_x^i) = 0 \\
Z_\beta^2 Z_x^i / Z_\delta &: (\sum_{i=l+1}^{m} A_{5,i} \hat{Z}_\beta u_i(\hat{Z}_x)/Z_\delta)(\sum_{i=0}^{n-1} B_{3,i} Z_\beta Z_x^i) = 0 \\
Z_\alpha Z_\beta Z_x^i / Z_\delta &: (\sum_{i=l+1}^{m} A_{5,i} \hat{Z}_\alpha v_i(\hat{Z}_x)/Z_\delta)(\sum_{i=0}^{n-1} B_{3,i} Z_\beta Z_x^i) + (\sum_{i=l+1}^{m} A_{5,i} \hat{Z}_\beta u_i(\hat{Z}_x)/Z_\delta)(\sum_{i=0}^{n-1} B_{2,i} Z_\alpha Z_x^i) = 0 \\
Z_\alpha Z_x^i / Z_\delta &: (\sum_{i=l+1}^{m} A_{5,i} \hat{Z}_\alpha v_i(\hat{Z}_x)/Z_\delta)(\sum_{i=0}^{2n-2} B_{1,i} Z_x^i) +
\end{aligned}$$

$$
\begin{aligned}
&(\sum_{i=l+1}^{m} A_{5,i} w_i(\hat{Z}_x)/Z_\delta + \sum_{i=0}^{n-2} A_{6,i} \hat{Z}_x{}^i t(\hat{Z}_x)/Z_\delta)(\sum_{i=0}^{n-1} B_{2,i} Z_\alpha Z_x^i) = 0 \\
Z_\beta Z_x^i/Z_\delta : &(\sum_{i=l+1}^{m} A_{5,i} \hat{Z}_\beta u_i(\hat{Z}_x)/Z_\delta)(\sum_{i=0}^{2n-2} B_{1,i} Z_x^i) + \\
&(\sum_{i=l+1}^{m} A_{5,i} w_i(\hat{Z}_x)/Z_\delta + \sum_{i=0}^{n-2} A_{6,i} \hat{Z}_x{}^i t(\hat{Z}_x)/Z_\delta)(\sum_{i=0}^{n-1} B_{3,i} Z_\beta Z_x^i) = 0
\end{aligned}
$$

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:

$$
\begin{aligned}
Z_\alpha Z_\beta Z_x^i &: A_{3:0} B_{2:0} = 1 \\
Z_\alpha^2 Z_x^i/Z_\delta &: (\sum_{i=l+1}^{m} A_{5,i} \hat{Z}_\alpha v_i(\hat{Z}_x)/Z_\delta) B_{2,0} Z_\alpha = 0 \\
Z_\alpha Z_\beta Z_x^i/Z_\delta &: (\sum_{i=l+1}^{m} A_{5,i} \hat{Z}_\beta u_i(\hat{Z}_x)/Z_\delta) B_{2,0} Z_\alpha = 0
\end{aligned}
$$

$$Z_\alpha Z_x^i/Z_\delta : (\sum_{i=l+1}^{m} A_{5,i}\hat{Z}_\alpha v_i(\hat{Z}_x)/Z_\delta)(\sum_{i=0}^{2n-2} B_{1,i}Z_x^i)+$$
$$(\sum_{i=l+1}^{m} A_{5,i}w_i(\hat{Z}_x)/Z_\delta + \sum_{i=0}^{n-2} A_{6,i}\hat{Z}_x{}^i t(\hat{Z}_x)/Z_\delta)B_{2,0}Z_\alpha = 0$$
$$Z_\beta Z_x^i/Z_\delta : (\sum_{i=l+1}^{m} A_{5,i}\hat{Z}_\beta u_i(\hat{Z}_x)/Z_\delta)(\sum_{i=0}^{2n-2} B_{1,i}Z_x^i) = 0$$

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:

$$Z_\alpha Z_x^i : B_{2,0}(A_0 + \sum A_{1,i}Z_x^i) = \sum a_i\alpha^{\mathcal{A}} v_i(\hat{Z}_x) + (\sum C_{5,i}\alpha^{\mathcal{A}} v_i(\hat{Z}_x))$$
$$Z_\beta Z_x^i : A_{3,0}(B_0 + \sum B_{1,i}Z_x^i) = \sum a_i\beta^{\mathcal{A}} u_i(\hat{Z}_x) + (\sum C_{5,i}\beta^{\mathcal{A}} u_i(\hat{Z}_x))$$
$$Z_x^i : (B_0 + \sum B_{1,i}Z_x^i)(A_0 + \sum A_{1,i}Z_x^i) = \sum a_i w_i(\hat{Z}_x) + \sum C_{5,i}w_i(\hat{Z}_x) + \sum C_{6,i}\hat{Z}_x^i t(\hat{Z}_x)$$

Express 1 and 2 and substitute into 3:

$$\frac{\beta^{\mathcal{A}}\alpha^{\mathcal{A}}}{A_{3,0}B_{2,0}}(\sum_{i=0}^{l} a_i u_i(\hat{Z}_x) + \sum_{i=l+1}^{m} C_{5,i}u_i(\hat{Z}_x))(\sum_{i=0}^{l} a_i v_i(\hat{Z}_x) + \sum_{i=l+1}^{m} C_{5,i}v_i(\hat{Z}_x)) =$$
$$\sum_{i=0}^{l} a_i w_i(\hat{Z}_x) + \sum_{i=l+1}^{m} C_{5,i}w_i(\hat{Z}_x) + \sum_{i=0}^{n-2} C_{6,i}\hat{Z}_x^i t(\hat{Z}_x)$$

$A_{3,0}B_{2,0} = \beta^{\mathcal{A}}\alpha^{\mathcal{A}}$, 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. □

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. The intuition behind this is that 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 $\vec{\psi}'_2$. And this means that $\mathcal{A}$ has knowledge of the exponent element, which is impossible assuming edlog.

Lemma 7. *The probability of* **bad** *in* Game_2 *is negligible under the* $(2n-1, 2n-2)$-***edlog*** *assumption.*

```
$\mathcal{B}(\{G^{z^i}\}_{i=1}^{2n-1}, \{H^{z^i}\}_{i=1}^{2n-2}, r_\delta, s_\delta, G^{\frac{1}{r_\delta z + s_\delta}}, H^{\frac{1}{r_\delta z + s_\delta}})$
Initialize $\mathsf{RO}_t(\cdot)$
$\{r_\iota, s_\iota\}_{\iota \in \{x,\alpha,\beta\}} \leftarrow\$ \mathbb{Z}_p$;
Set implicitly $z_\iota \leftarrow r_\iota z + s_\iota$ for critical query embeddings for $\iota \in \{\alpha, \beta, x\}$;
Similarly set $z_\delta \leftarrow \dfrac{1}{r_\delta z + s_\delta}$
Run $\mathcal{A}$ and $\mathcal{E}$ as in $\mathsf{Game}_1$ using dlog challenge elements to embed $z_\iota$
   into critical SRS updates, and modified $\mathsf{SimUpdProof}_\mathcal{B}$
assert $\mathsf{Verify}(\mathsf{srs}, \phi, \pi) = 1 \land (\phi, w) \notin \mathcal{R}$;

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 and return it;

$\mathsf{SimUpdProof}_\mathcal{B}(\iota, \varphi)$
We compute $G^{\hat{\iota}'}, H^{\hat{\iota}'}$, as before, except now we do not know exponent of $G^{z_\iota}, H^{z_\iota}$;
   Notice: for $\delta$, $G^{\hat{\iota}'} = (G^{\frac{1}{r_\delta z + s_\delta}})^{\hat{\iota}^{-1}}$ due to inverted embedding.
As in $\mathsf{SimUpdProof}$, create $\phi$ and call $\mathsf{Sim}_{dl}^{\mathsf{RO}_1(\cdot)}$ on it to obtain $\pi_{\iota'}$.
return $(G^{z_\iota}, G^{\hat{\iota}'}, H^{\hat{\iota}'}, \pi_{\iota'})$
```

Figure 5.11: Adversary $\mathcal{B}$ against $(2n-1, 2n-2)$-extended dlog assumption in Theorem 12. It is parameterized by a full update knowledge soundness algebraic adversary $\mathcal{A}$, and the extractor $\mathcal{E}_\mathcal{A}$ as in Fig. 5.8. Its main task is to simulate Game_1 to $\mathcal{A}$, embedding the edlog instance z into SRS on critical queries.

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.

Let $\mathcal{A}$ be a PPT adversary in Game_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 Game_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. 5.11. 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

$$\mathsf{srs}'_u = \begin{pmatrix} \{G^{(r_x z + s_x)^i}, H^{(r_x z + s_x)^i}\}_{i=0}^{2n-2}, \{G^{(r_\alpha z + s_\alpha)(r_x z + s_x)^i}, G^{(r_\beta z + s_\beta)(r_x z + s_x)^i}, \\ H^{(r_\alpha z + s_\alpha)(r_x z + s_x)^i}, H^{(r_\beta z + s_\beta)(r_x z + s_x)^i}\}_{i=0}^{n-1} \end{pmatrix}.$$

Similarly $\mathsf{SimUpdProof}$ is computed exactly as in Game_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 $(\pi^{\mathcal{A}}_{1:1}, \ldots, \pi^{\mathcal{A}}_{1:t_1})$ 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

$$\mathsf{srs}_u = \begin{pmatrix} \{G^{\left((r_x z+s_x)\prod_j x_j^{\mathcal{A}}\right)^i}, H^{\left((r_x z+s_x)\prod_j x_j^{\mathcal{A}}\right)^i}\}_{i=0}^{2n-2}, \\ \{G^{\left((r_\alpha z+s_\alpha)\prod_j \alpha_j^{\mathcal{A}}\right)\left((r_x z+s_x)\prod_j x_j^{\mathcal{A}}\right)^i}, G^{\left((r_\beta z+s_\beta)\prod_j \beta_j^{\mathcal{A}}\right)\left((r_x z+s_x)\prod_j x_j^{\mathcal{A}}\right)^i}, \\ H^{\left((r_\alpha z+s_\alpha)\prod_j \alpha_j^{\mathcal{A}}\right)\left((r_x z+s_x)\prod_j x_j^{\mathcal{A}}\right)^i}, H^{\left((r_\beta z+s_\beta)\prod_j \beta_j^{\mathcal{A}}\right)\left((r_x z+s_x)\prod_j x_j^{\mathcal{A}}\right)^i}\}_{i=0}^{n-1} \end{pmatrix}.$$

where $j = 1, \dots, t_1$. The reasoning of why the form of srs_u is that is similar to Lemma 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 $Z_x(Z) = (r_x Z+s_x)\prod_j x_j^{\mathcal{A}}$ and $Z_\alpha(Z) = (r_\alpha Z+s_\alpha)\prod_j \alpha_j^{\mathcal{A}}$ and $Z_\beta(Z) = (r_\beta Z+s_\beta)\prod_j \beta_j^{\mathcal{A}}$. 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 $\mathcal{O}_{\mathsf{poly}}^{\mathbb{G}_1}, \mathcal{O}_{\mathsf{poly}}^{\mathbb{G}_2}$ oracles (the extraction success is argued similarly to how it is done in Lemma 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

$$\mathsf{srs}_s = \begin{pmatrix} G^{\frac{1}{r_\delta z+s_\delta}}, H^{\frac{1}{r_\delta z+s_\delta}}, \{G^{(r_\delta z+s_\delta)\left(Z_\beta(z)u_i(Z_x(z))+Z_\alpha(z)v_i(Z_x(z))+w_i(Z_x(z))\right)}\}_{i=\ell+1}^{m}, \\ \{G^{(r_\delta z+s_\delta)(Z_x(z))^i t(Z_x(z))}\}_{i=0}^{n-2} \end{pmatrix}.$$

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 $G^{(\alpha u_i(x)+\dots)/\delta}$ 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 ϕ. 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 $(\pi_{2:1}^{\mathcal{A}}, \dots, \pi_{2:t_2}^{\mathcal{A}})$ 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

$$\mathsf{srs}_s = \begin{pmatrix} G^{\frac{\prod_j \delta_j^{\mathcal{A}}}{r_\delta z+s_\delta}}, H^{\frac{\prod_j \delta_j^{\mathcal{A}}}{r_\delta z+s_\delta}}, \left\{G^{\frac{(r_\delta z+s_\delta)\left(\hat{Z}_\beta(z)u_i(\hat{Z}_x(z))+\hat{Z}_\alpha(z)v_i(\hat{Z}_x(z))+w_i(\hat{Z}_x(z))\right)}{\prod_j \delta_j^{\mathcal{A}}}}\right\}_{i=\ell+1}^{m}, \\ \left\{G^{\frac{(r_\delta z+s_\delta)(\hat{Z}_x(z))^i t(\hat{Z}_x(z))}{\prod_j \delta_j^{\mathcal{A}}}}\right\}_{i=0}^{n-2} \end{pmatrix}$$

where $j = 1, \ldots, t_2$. We, as before, set $Z_\delta(Z) = \frac{r_\delta Z + s_\delta}{\prod_j \delta_j^{\mathcal{A}}}$.

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_δ^i 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_δ^{-i}. 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 $Q_4(\{Z_\iota\}_\iota) := Q_3(\{Z_\iota\}_\iota) \cdot Z_\delta^2$, where Z_δ is a formal variable. Q_4 is a positive power polynomial since Q_3 can only have at most Z_δ^{-2} as a negative degree monomial: e.g. Z_δ^{-1} 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 $\mathbb{Z}_p[R_\alpha, R_\beta, R_x, R_\delta, S_\alpha, S_\beta, S_x, S_\delta]$. According to Lemma 2, 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 3), 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. □

Now, combining the results of the last two lemmas with the previous game transitions:

$$\begin{aligned}
\Pr[\mathsf{Game}_0^{\mathcal{A},\mathcal{E}_\mathcal{A}}(1^\lambda) = 1] &\leq \Pr[\mathsf{Game}_1^{\mathcal{A},\mathcal{E}_\mathcal{A}}(1^\lambda) = 1] + \mathsf{negl}(\lambda) \\
&= (q_1 q_2) \Pr[\mathsf{Game}_2^{\mathcal{A},\mathcal{E}_\mathcal{A}}(1^\lambda) = 1] + \mathsf{negl}(\lambda) \\
&\leq (q_1 q_2)\big(\Pr[\mathsf{Game}_2^{\mathcal{A},\mathcal{E}_\mathcal{A}}(1^\lambda) = 1 \mid \neg\textbf{bad}] + \Pr[\textbf{bad}]\big) + \mathsf{negl}(\lambda) \\
&= (q_1 q_2)(\mathsf{negl}(\lambda) + \mathsf{negl}(\lambda)) + \mathsf{negl}(\lambda) = \mathsf{negl}(\lambda)
\end{aligned}$$

This concludes the proof of the update knowledge soundness theorem. □

5.7 Batched VerifySRS

We provide an optimized VerifySRS algorithm for Groth's SNARK. It follows closely the batching techniques used in [ALSZ20] 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 $s_1, \ldots, s_n \leftarrow\$ \mathbb{Z}_p$ 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-exponentiations 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 VerifySRS in figure 5.12 to construct a batched batchVerifySRS.

Theorem 14. *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 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 $i \in \{1, \ldots, t\}$ 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 $s_i \leftarrow\$ \{0,1\}^{\kappa}$. All of the batched equations in 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 $p(X_1, \ldots, X_n) = \sum_{i=1}^{t}(a_i b_i - c_i d_i) X_i$. If one of the initial equations is not satisfied then p is a non-zero polynomial and the probability $p(s_1, \ldots, s_t) = 0$ is bounded by $1/2^{\kappa}$. Given that we batch 12 sets of equations, $\Pr[\mathsf{VerifySRS}(\mathsf{QAP}, \mathsf{srs}, Q) = 0 \wedge \mathsf{batchVerifySRS}(\mathsf{QAP}, \mathsf{srs}, Q) = 1] \leq 12/2^{\kappa}$. □

5.8 Lemmas for Groth16 Completeness

This section presents the additional lemmas for the completeness proof of Theorem 10.

Lemma 8. *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 us assume that its exponent is δ), and that exponent of H_{δ} is the same.

```
$\underline{\mathsf{batchVerifySRS}^{\mathsf{RO}(\cdot)}(\mathsf{QAP}, \mathsf{srs}, Q)}$:
  1. Parse $\mathsf{srs} = (\mathsf{srs}_u, \mathsf{srs}_s)$ and $Q = (Q_u, Q_s) = \{\rho_{u,i}\}_{i=1}^{k_u} \cup \{\rho_{s,i}\}_{i=1}^{k_s}$;
  2. Parse $\mathsf{srs}_u = \left(\{G_{x:i}, H_{x:i}\}_{i=0}^{2n-2}, \{G_{\alpha x:i}, G_{\beta x:i}, H_{\alpha x:i}, H_{\beta x:i}\}_{i=0}^{n-1}\right)$ and assert that elements be-
     long to correct groups;
  3. Sample $s_0, \dots, s_{\max} \leftarrow\$ \{0,1\}^{\kappa}$ where $\max = \max\{2n-2, m, k_u, k_s\}$;
  4. For $i = 1, \dots, k_u$:
     (a) Parse $\rho_{u,i} = (\rho_{\alpha'}^{(i)}, \rho_{\beta'}^{(i)}, \rho_{x'}^{(i)})$;
     (b) For $\iota \in \{\alpha, \beta, x\}$: Parse $\rho_{\iota'}^{(i)} = (G_{\iota}^{(i)}, G_{\iota'}^{(i)}, H_{\iota'}^{(i)}, \pi_{\iota'}^{(i)})$; $R_{\iota'}^{(i)} \leftarrow \mathsf{RO}(G_{\iota'}^{(i)}, H_{\iota'}^{(i)})$;
  5. For $\iota \in \{\alpha, \beta, x\}$:
     (a) Assert $\hat{e}(\prod_{i=2}^{k_u}(G_{\iota}^{(i)})^{s_i}, H) = \prod_{i=2}^{k_u} \hat{e}((G_{\iota}^{(i-1)})^{s_i}, H_{\iota'}^{(i)})$;
     (b) Assert $\hat{e}(\prod_{i=1}^{k_u}(G_{\iota'}^{(i)})^{s_i}, H) = \hat{e}(G, \prod_{i=1}^{k_u}(H_{\iota'}^{(i)})^{s_i})$;
     (c) Assert $\hat{e}(\prod_{i=1}^{k_u}(\pi_{\iota'}^{(i)})^{s_i}, H) = \prod_{i=1}^{k_u} \hat{e}((R_{\iota'}^{(i)})^{s_i}, H_{\iota'}^{(i)})$;
  6. Assert $G_{x:1} = G_x^{(k_u)} \neq 1$; $G_{\alpha x:0} = G_{\alpha}^{(k_u)} \neq 1$; $G_{\beta x:0} = G_{\beta}^{(k_u)} \neq 1$;
  7. Assert $\hat{e}(\prod_{i=1}^{2n-2} G_{x:i}^{s_i}, H) = \hat{e}(G, \prod_{i=1}^{2n-2} H_{x:i}^{s_i})$ and $\hat{e}(\prod_{i=1}^{2n-2} G_{x:i}^{s_i}, H) =$
     $\hat{e}(\prod_{i=1}^{2n-2} G_{x:(i-1)}^{s_i}, H_{x:1})$;
  8. For $\iota \in \{\alpha, \beta\}$: Assert $\hat{e}(\prod_{i=0}^{n-1} G_{\iota x:i}^{s_i}, H) = \hat{e}(G, \prod_{i=0}^{n-1} H_{\iota x:i}^{s_i})$ and $\hat{e}(\prod_{i=0}^{n-1} G_{\iota x:i}^{s_i}, H) =$
     $\hat{e}(\prod_{i=0}^{n-1} G_{x:i}^{s_i}, H_{\iota x:0})$;
  9. Parse $\mathsf{srs}_s \leftarrow \left(G_\delta, H_\delta, \{G_{sum:i}\}_{i=\ell+1}^{m}, \{G_{t(x):i}\}_{i=0}^{n-2},\right)$ and assert that elements belong to correct
     groups;
 10. For $i = 1, \dots, k_s$: Parse $\rho_{s,i} = (G_{\delta}^{(i)}, G_{\delta'}^{(i)}, H_{\delta'}^{(i)}, \pi_{\delta'})$; $R_{\delta'}^{(i)} \leftarrow \mathsf{RO}(G_{\delta'}^{(i)}, H_{\delta'}^{(i)})$;
 11. (a) Assert $\hat{e}(\prod_{i=2}^{k_s}(G_{\delta}^{(i)})^{s_i}, H) = \prod_{i=2}^{k_s} \hat{e}((G_{\delta}^{(i-1)})^{s_i}, H_{\delta'}^{(i)})$;
     (b) Assert $\hat{e}(\prod_{i=1}^{k_s}(G_{\delta'}^{(i)})^{s_i}, H) = \hat{e}(G, \prod_{i=1}^{k_s}(H_{\delta'}^{(i)})^{s_i})$;
     (c) Assert $\hat{e}(\prod_{i=1}^{k_s}(\pi_{\delta'}^{(i)})^{s_i}, H) = \prod_{i=1}^{k_s} \hat{e}((R_{\delta'}^{(i)})^{s_i}, H_{\delta'}^{(i)})$;
 12. Assert $\hat{e}(G_\delta, H) = \hat{e}(G, H_\delta)$ and $G_\delta = G_{\delta}^{(k_s)} \neq 1$;
 13. Assert $\hat{e}(\prod_{i=\ell+1}^{m} G_{sum:i}^{s_i}, H_\delta) = \hat{e}(\prod_{i=\ell+1}^{m} \left(\prod_{j=0}^{n-1} G_{\beta x:j}^{u_{ij}} \cdot G_{\alpha x:j}^{v_{ij}} \cdot G_{x:j}^{w_{ij}}\right)^{s_i}, H)$;
 14. 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 5.12: Batched SRS verification algorithm for Groth's SNARK where $\kappa \approx 2^{40}$

- 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 Groth's SNARK Setup. □

Lemma 9. *Groth's SNARK has update completeness.*

Proof. Again, we are analysing Update 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.

$\square$

Chapter 6

Conclusion

This document presented recent advances of PRIViLEDGE partners in designing privacy-preserving cryptographic primitives for distributed ledgers. While the deliverable D2.2 illustrates some security notions for cryptographic primitives, and D2.3 presents concrete protocols that implement these notions, the current document extends, and builds on D2.3, providing improvements to the cryptographic primitives presented earlier, as well as new standalone contributions in line with the work done previously. All the presented contributions — working with smart contracts, NIZK and their setup ceremonies, and secure groups — are closely related to the relevant issues and advances in the area of distributed ledger protocols.

Bibliography

[ABL$^+$19] Behzad Abdolmaleki, Karim Baghery, Helger Lipmaa, Janno Siim, and Michal Zajac. UC-secure CRS generation for SNARKs. In Johannes Buchmann, Abderrahmane Nitaj, and Tajje eddine Rachidi, editors, *AFRICACRYPT 19: 11th International Conference on Cryptology in Africa*, volume 11627 of *Lecture Notes in Computer Science*, pages 99–117, Rabat, Morocco, July 9–11, 2019. Springer, Heidelberg, Germany.

[ABLZ17] Behzad Abdolmaleki, Karim Baghery, Helger Lipmaa, and Michal Zajac. A subversion-resistant SNARK. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017, Part III*, volume 10626 of *Lecture Notes in Computer Science*, pages 3–33, Hong Kong, China, December 3–7, 2017. Springer, Heidelberg, Germany.

[ACS02] Joy Algesheimer, Jan Camenisch, and Victor Shoup. Efficient computation modulo a shared secret with application to the generation of shared safe-prime products. In Moti Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 417–432, Santa Barbara, CA, USA, August 18–22, 2002. Springer, Heidelberg, Germany.

[AF06] Saurabh Agarwal and Gudmund Skovbjerg Frandsen. A new GCD algorithm for quadratic number rings with unique factorization. In José R. Correa, Alejandro Hevia, and Marcos A. Kiwi, editors, *LATIN 2006: Theoretical Informatics, 7th Latin American Symposium, Valdivia, Chile, March 20-24, 2006, Proceedings*, volume 3887 of *Lecture Notes in Computer Science*, pages 30–42. Springer, 2006.

[AFK$^+$20] Antonis Aggelakis, Prastudy Fauzi, Georgios Korfiatis, Panos Louridas, Foteinos Mergoupis-Anagnou, Janno Siim, and Michal Zajac. A non-interactive shuffle argument with low trust assumptions. In Stanislaw Jarecki, editor, *Topics in Cryptology – CT-RSA 2020*, volume 12006 of *Lecture Notes in Computer Science*, pages 667–692, San Francisco, CA, USA, February 24–28, 2020. Springer, Heidelberg, Germany.

[ALSZ20] Behzad Abdolmaleki, Helger Lipmaa, Janno Siim, and Michał Zając. On subversion-resistant SNARKs. Cryptology ePrint Archive, Report 2020/668, 2020. `https://eprint.iacr.org/2020/668`.

[Ank52] N. C. Ankeny. The least quadratic non residue. *Annals of Mathematics*, 55(1):65–72, 1952.

[BBBF18] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 757–788, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany.

[BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *Advances in Cryptology - CRYPTO 2004, 24th Annual International CryptologyConference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, pages 41–55, 2004.

[BCG$^+$14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474, Berkeley, CA, USA, May 18–21, 2014. IEEE Computer Society Press.

[BCG$^+$15] Eli Ben-Sasson, Alessandro Chiesa, Matthew Green, Eran Tromer, and Madars Virza. Secure sampling of public parameters for succinct zero knowledge proofs. In *2015 IEEE Symposium on Security and Privacy*, pages 287–304, San Jose, CA, USA, May 17–21, 2015. IEEE Computer Society Press.

[BCG$^+$20] Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. ZEXE: Enabling decentralized private computation. In *2020 IEEE Symposium on Security and Privacy*, pages 947–964, San Francisco, CA, USA, May 18–21, 2020. IEEE Computer Society Press.

[BCNP04] Boaz Barak, Ran Canetti, Jesper Buus Nielsen, and Rafael Pass. Universally composable protocols with relaxed set-up assumptions. In *45th Symposium on Foundations of Computer Science (FOCS 2004), 17-19 October 2004, Rome, Italy, Proceedings*, pages 186–195, 2004.

[BCTV14] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von neumann architecture. In Kevin Fu and Jaeyeon Jung, editors, *USENIX Security 2014: 23rd USENIX Security Symposium*, pages 781–796, San Diego, CA, USA, August 20–22, 2014. USENIX Association.

[BDD$^+$20] Carsten Baum, Bernardo David, Rafael Dowsley, Jesper Buus Nielsen, and Sabine Oechsner. Craft: Composable randomness and almost fairness from time. Cryptology ePrint Archive, Report 2020/784, 2020. `https://eprint.iacr.org/2020/784`.

[BDMP91] Manuel Blum, Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. Noninteractive zero-knowledge. *SIAM J. Comput.*, 20(6):1084–1118, 1991.

[BFL20] Balthazar Bauer, Georg Fuchsbauer, and Julian Loss. A classification of computational assumptions in the algebraic group model. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part II*, volume 12171 of *Lecture Notes in Computer Science*, pages 121–151, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Heidelberg, Germany.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 103–112, 1988.

[BFS16] Mihir Bellare, Georg Fuchsbauer, and Alessandra Scafuro. NIZKs with an untrusted CRS: Security in the face of parameter subversion. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016, Part II*, volume 10032 of *Lecture Notes in Computer Science*, pages 777–804, Hanoi, Vietnam, December 4–8, 2016. Springer, Heidelberg, Germany.

[BG93] Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. In Ernest F. Brickell, editor, *Advances in Cryptology – CRYPTO'92*, volume 740 of *Lecture Notes in Computer Science*, pages 390–420, Santa Barbara, CA, USA, August 16–20, 1993. Springer, Heidelberg, Germany.

[BGG17] Sean Bowe, Ariel Gabizon, and Matthew D. Green. A multi-party protocol for constructing the public parameters of the pinocchio zk-SNARK. Cryptology ePrint Archive, Report 2017/602, 2017. `https://eprint.iacr.org/2017/602`.

[BGM17] Sean Bowe, Ariel Gabizon, and Ian Miers. Scalable multi-party computation for zk-SNARK parameters in the random beacon model. Cryptology ePrint Archive, Report 2017/1050, 2017. `https://eprint.iacr.org/2017/1050`.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 1–10. ACM, 1988.

[BH84] Duncan A Buell and Richard H Hudson. On runs of consecutive quadratic residues and quadratic nonresidues. *BIT Numerical Mathematics*, 24(2):243–247, 1984.

[BIB89] Judit Bar-Ilan and Donald Beaver. Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In Piotr Rudnicki, editor, *8th ACM Symposium Annual on Principles of Distributed Computing*, pages 201–209, Edmonton, Alberta, Canada, August 14–16, 1989. Association for Computing Machinery.

[BK85] R. P. Brent and H. T. Kung. A systolic algorithm for integer gcd computation. In *1985 IEEE 7th Symposium on Computer Arithmetic (ARITH)*, pages 118–125, 1985.

[BK14] Iddo Bentov and Ranjit Kumaresan. How to use bitcoin to design fair protocols. In *CRYPTO*, pages 421–439, 2014.

[BKSV20] Karim Baghery, Markulf Kohlweiss, Janno Siim, and Mikhail Volkhov. Another look at extraction and randomization of groth's zk-SNARK. Cryptology ePrint Archive, Report 2020/811, 2020. `https://eprint.iacr.org/2020/811`.

[BKZZ16] Foteini Baldimtsi, Aggelos Kiayias, Thomas Zacharias, and Bingsheng Zhang. Indistinguishable proofs of work or knowledge. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part II*, volume 10032 of *Lecture Notes in Computer Science*, pages 902–933, 2016.

[BL07] Daniel J. Bernstein and Tanja Lange. Faster addition and doubling on elliptic curves. In Kaoru Kurosawa, editor, *Advances in Cryptology – ASIACRYPT 2007*, volume 4833 of *Lecture Notes in Computer Science*, pages 29–50, Kuching, Malaysia, December 2–6, 2007. Springer, Heidelberg, Germany.

[BLS04] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. *J. Cryptol.*, 17(4):297–319, 2004.

[Blu86] Manuel Blum. How to prove a theorem so no one else can claim it. In *In Proceedings of the International Congress of Mathematicians*, pages 1444–1454, 1986.

[BMMV19] Benedikt Bünz, Mary Maller, Pratyush Mishra, and Noah Vesely. Proofs for inner pairing products and applications. Cryptology ePrint Archive, Report 2019/1177, 2019. `https://eprint.iacr.org/2019/1177`.

[BOSS20] Carsten Baum, Emmanuela Orsini, Peter Scholl, and Eduardo Soria-Vazquez. Efficient constant-round MPC with identifiable abort and public verifiability. In *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part II*, pages 562–592, 2020.

[BP04] Boaz Barak and Rafael Pass. On the possibility of one-message weak zero-knowledge. In *Theory of Cryptography, First Theory of Cryptography Conference, TCC 2004, Cambridge, MA, USA, February 19-21, 2004, Proceedings*, pages 121–132, 2004.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993.*, pages 62–73, 1993.

[BS05] Boaz Barak and Amit Sahai. How to play almost any mental game over the net - concurrent composition via super-polynomial simulation. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005), 23-25 October 2005, Pittsburgh, PA, USA, Proceedings*, pages 543–552. IEEE Computer Society, 2005.

[Bur63] DA Burgess. A note on the distribution of residues and non-residues. *Journal of the London Mathematical Society*, 1(1):253–256, 1963.

[BV07] Johannes A. Buchmann and Ulrich Vollmer. *Binary quadratic forms - an algorithmic approach*, volume 20 of *Algorithms and computation in mathematics*. Springer, 2007.

[BW] Bitcoin Wiki. Confirmation in bitcoin. `https://en.bitcoin.it/wiki/Confirmation`.

[BY19a] Daniel J. Bernstein and Bo-Yin Yang. Fast constant-time gcd computation and modular inversion. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(3):340–398, 2019. `https://tches.iacr.org/index.php/TCHES/article/view/8298`.

[BY19b] Daniel J. Bernstein and Bo-Yin Yang. Fast constant-time gcd computation and modular inversion. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(3):340–398, 2019.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 136–145, 2001.

[CD98] Ronald Cramer and Ivan Damgård. Zero-knowledge proofs for finite field arithmetic; or: Can zero-knowledge be for free? In Hugo Krawczyk, editor, *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*, volume 1462 of *Lecture Notes in Computer Science*, pages 424–441. Springer, 1998.

[CD17] Ignacio Cascudo and Bernardo David. SCRAPE: Scalable randomness attested by public entities. In Dieter Gollmann, Atsuko Miyaji, and Hiroaki Kikuchi, editors, *ACNS 17: 15th International Conference on Applied Cryptography and Network Security*, volume 10355 of *Lecture Notes in Computer Science*, pages 537–556, Kanazawa, Japan, July 10–12, 2017. Springer, Heidelberg, Germany.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Advances in Cryptology - CRYPTO '94, 14th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1994, Proceedings*, pages 174–187, 1994.

[CG15] Pyrros Chaidos and Jens Groth. Making sigma-protocols non-interactive without random oracles. In *Public-Key Cryptography - PKC 2015 - 18th IACR International Conference on Practice and Theory in Public-Key Cryptography, Gaithersburg, MD, USA, March 30 - April 1, 2015, Proceedings*, pages 650–670, 2015.

[CH19] Yang Chu and Robert Hough. Solution of the 15 puzzle problem, 2019.

[CHM$^+$20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part I*, volume 12105 of

Lecture Notes in Computer Science, pages 738–768, Zagreb, Croatia, May 10–14, 2020. Springer, Heidelberg, Germany.

[CJS14] Ran Canetti, Abhishek Jain, and Alessandra Scafuro. Practical UC security with a global random oracle. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, pages 597–608, 2014.

[Coh93] Henri Cohen. *A course in computational algebraic number theory*, volume 138 of *Graduate texts in mathematics*. Springer, 1993.

[Cox11] David A Cox. *Primes of the form x2+ ny2: Fermat, class field theory, and complex multiplication*, volume 34. John Wiley & Sons, 2011.

[CPS$^+$16] Michele Ciampi, Giuseppe Persiano, Alessandra Scafuro, Luisa Siniscalchi, and Ivan Visconti. Improved or-composition of sigma-protocols. In Eyal Kushilevitz and Tal Malkin, editors, *Theory of Cryptography - 13th International Conference, TCC 2016-A, Tel Aviv, Israel, January 10-13, 2016, Proceedings, Part II*, volume 9563 of *Lecture Notes in Computer Science*, pages 112–141. Springer, 2016.

[CPSV16] Michele Ciampi, Giuseppe Persiano, Luisa Siniscalchi, and Ivan Visconti. A transform for NIZK almost as efficient and general as the fiat-shamir transform without programmable random oracles. In *Theory of Cryptography - 13th International Conference, TCC 2016-A, Tel Aviv, Israel, January 10-13, 2016, Proceedings, Part II*, pages 83–111, 2016.

[CS03] R. Cramer and V. Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, 33(1):167–226, 2003.

[CS10] Octavian Catrina and Amitabh Saxena. Secure computation with fixed-point numbers. In Radu Sion, editor, *FC 2010: 14th International Conference on Financial Cryptography and Data Security*, volume 6052 of *Lecture Notes in Computer Science*, pages 35–50, Tenerife, Canary Islands, Spain, January 25–28, 2010. Springer, Heidelberg, Germany.

[DFGK14] George Danezis, Cédric Fournet, Jens Groth, and Markulf Kohlweiss. Square span programs with applications to succinct NIZK arguments. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 532–550, Kaoshiung, Taiwan, R.O.C., December 7–11, 2014. Springer, Heidelberg, Germany.

[DFK$^+$06] Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In Shai Halevi and Tal Rabin, editors, *TCC 2006: 3rd Theory of Cryptography Conference*, volume 3876 of *Lecture Notes in Computer Science*, pages 285–304, New York, NY, USA, March 4–7, 2006. Springer, Heidelberg, Germany.

[DFN06] Ivan Damgård, Nelly Fazio, and Antonio Nicolosi. Non-interactive zero-knowledge from homomorphic encryption. In *Theory of Cryptography, Third Theory of Cryptography Conference, TCC, 2006, New York, NY, USA, March 4-7, 2006, Proceedings*, pages 41–59, 2006.

[Dix08] John D. Dixon. Generating random elements in finite groups. *Electron. J. Comb.*, 15(1), 2008.

[DMP87] Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. Non-interactive zero-knowledge proof systems. In *Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings*, pages 52–72, 1987.

[DNT12] Morten Dahl, Chao Ning, and Tomas Toft. On secure two-party integer division. In Angelos D. Keromytis, editor, *FC 2012: 16th International Conference on Financial Cryptography and Data*

Security, volume 7397 of *Lecture Notes in Computer Science*, pages 164–178, Kralendijk, Bonaire, February 27 – March 2, 2012. Springer, Heidelberg, Germany.

[Fis05] Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 152–168. Springer, 2005.

[FKL18] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 33–62, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany.

[FLS90] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple non-interactive zero knowledge proofs based on a single random string (extended abstract). In *31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22-24, 1990, Volume I*, pages 308–317. IEEE Computer Society, 1990.

[FLS99] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple non-interactive zero knowledge proofs under general assumptions. *SIAM J. on Computing*, 29(1):1–28, 1999.

[FPS20] Georg Fuchsbauer, Antoine Plouviez, and Yannick Seurin. Blind schnorr signatures and signed ElGamal encryption in the algebraic group model. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part II*, volume 12106 of *Lecture Notes in Computer Science*, pages 63–95, Zagreb, Croatia, May 10–14, 2020. Springer, Heidelberg, Germany.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, pages 186–194, 1986.

[Fuc18] Georg Fuchsbauer. Subversion-zero-knowledge SNARKs. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018: 21st International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 10769 of *Lecture Notes in Computer Science*, pages 315–347, Rio de Janeiro, Brazil, March 25–29, 2018. Springer, Heidelberg, Germany.

[Gab19] Ariel Gabizon. On the security of the BCTV pinocchio zk-SNARK variant. Cryptology ePrint Archive, Report 2019/119, 2019. `https://eprint.iacr.org/2019/119`.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 626–645, Athens, Greece, May 26–30, 2013. Springer, Heidelberg, Germany.

[GKM$^+$18] Jens Groth, Markulf Kohlweiss, Mary Maller, Sarah Meiklejohn, and Ian Miers. Updatable and universal common reference strings with applications to zk-SNARKs. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part III*, volume 10993 of *Lecture Notes in Computer Science*, pages 698–728, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany.

[GM17] Jens Groth and Mary Maller. Snarky signatures: Minimal signatures of knowledge from simulation-extractable SNARKs. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part II*, volume 10402 of *Lecture Notes in Computer Science*, pages 581–612, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *17th Annual ACM Symposium on Theory of Computing*, pages 291–304, Providence, RI, USA, May 6–8, 1985. ACM Press.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[GMY06] Juan A. Garay, Philip MacKenzie, and Ke Yang. Strengthening zero-knowledge protocols using signatures. *Journal of Cryptology*, 19(2):169–209, 2006.

[GO94] Oded Goldreich and Yair Oren. Definitions and properties of zero-knowledge proof systems. *J. Cryptology*, 7(1):1–32, 1994.

[GQ88] Louis C. Guillou and Jean-Jacques Quisquater. A practical zero-knowledge protocol fitted to security microprocessor minimizing both transmission and memory. In Christoph G. Günther, editor, *Advances in Cryptology - EUROCRYPT '88, Workshop on the Theory and Application of of Cryptographic Techniques, Davos, Switzerland, May 25-27, 1988, Proceedings*, volume 330 of *Lecture Notes in Computer Science*, pages 123–128. Springer, 1988.

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In Masayuki Abe, editor, *Advances in Cryptology – ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 321–340, Singapore, December 5–9, 2010. Springer, Heidelberg, Germany.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 305–326, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany.

[GRR98] Rosario Gennaro, Michael O. Rabin, and Tal Rabin. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In Brian A. Coan and Yehuda Afek, editors, *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing, PODC '98, Puerto Vallarta, Mexico, June 28 - July 2, 1998*, pages 101–111. ACM, 1998.

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019. `https://eprint.iacr.org/2019/953`.

[HM16] Derek Holt and Henning Makholm. What are some Group representation of the rubik's cube group? StackExchange Mathematics `https://math.stackexchange.com/questions/1587307/what-are-some-group-representation-of-the-rubiks-cube-group`, 2016. [Online; accessed 23-January-2020].

[HMW18] Timo Hanke, Mahnush Movahedi, and Dominic Williams. Dfinity technology overview series, consensus system. *arXiv preprint arXiv:1805.04548*, 2018. `https://arxiv.org/abs/1805.04548`.

[Hoo12] S.J.A. Hoogh, de. *Design of large scale applications of secure multiparty computation : secure linear programming*. PhD thesis, Department of Mathematics and Computer Science, 2012.

[Hum03] Patrick Hummel. On consecutive quadratic non-residues: a conjecture of issai schur. *Journal of Number Theory*, 103(2):257–266, 2003.

[HWCD08] Hüseyin Hisil, Kenneth Koon-Ho Wong, Gary Carter, and Ed Dawson. Twisted edwards curves revisited. In Josef Pieprzyk, editor, *Advances in Cryptology - ASIACRYPT 2008, 14th International Conference on the Theory and Application of Cryptology and Information Security, Melbourne,* *Australia, December 7-11, 2008. Proceedings*, volume 5350 of *Lecture Notes in Computer Science*, pages 326–343. Springer, 2008.

[HYL20] Runchao Han, Jiangshan Yu, and Haoyu Lin. RandChain: Decentralised randomness beacon from sequential proof-of-work. Cryptology ePrint Archive, Report 2020/1033, 2020. `https://eprint.iacr.org/2020/1033`.

[IOZ14] Yuval Ishai, Rafail Ostrovsky, and Vassilis Zikas. Secure multi-party computation with identifiable abort. In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 369–386. Springer, 2014.

[KKK21] Thomas Kerber, Aggelos Kiayas, and Markulf Kohlweiss. Composition with knowledge assumptions. Cryptology ePrint Archive, Report 2021/165, 2021. `https://eprint.iacr.org/2021/165`.

[KMS$^+$16] Ahmed E. Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *2016 IEEE Symposium on Security and Privacy*, pages 839–858, San Jose, CA, USA, May 22–26, 2016. IEEE Computer Society Press.

[Kob87] Neal Koblitz, editor. *Elliptic curve cryptosystems*, volume 5350 of *Mathematics of Computation 48*. American Mathematical Society, 1987.

[KPPS20] Ahmed E. Kosba, Dimitrios Papadopoulos, Charalampos Papamanthou, and Dawn Song. MIRAGE: Succinct arguments for randomized algorithms with applications to universal zk-SNARKs. In Srdjan Capkun and Franziska Roesner, editors, *USENIX Security 2020: 29th USENIX Security Symposium*, pages 2129–2146. USENIX Association, August 12–14, 2020.

[KRDO17] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 357–388, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.

[LCKO19] Jiwon Lee, Jaekyoung Choi, Jihye Kim, and Hyunok Oh. SAVER: Snark-friendly, additively-homomorphic, and verifiable encryption and decryption with rerandomization. Cryptology ePrint Archive, Report 2019/1270, 2019. `https://eprint.iacr.org/2019/1270`.

[LD63] Peter Lejeune Dirichlet. Vorlesungen über zahlentheorie, 1863.

[Lin15] Yehuda Lindell. An efficient transform from sigma protocols to NIZK with a CRS and non-programmable random oracle. In *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part I*, pages 93–109, 2015.

[Lip12] Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In Ronald Cramer, editor, *TCC 2012: 9th Theory of Cryptography Conference*, volume 7194 of *Lecture Notes in Computer Science*, pages 169–189, Taormina, Sicily, Italy, March 19–21, 2012. Springer, Heidelberg, Germany.

[Lon19] Lipa Long. Binary quadratic forms. GitHub `https://github.com/Chia-Network/vdf-competition/blob/master/classgroups.pdf`, 2019. [Online; accessed 23-January-2020].

[LP92] H. W. Lenstra and Carl Pomerance. A rigorous time bound for factoring integers. *Journal of the American Mathematical Society*, 5(3):483–483, sep 1992.

[Mau15] Ueli Maurer. Zero-knowledge proofs of knowledge for group homomorphisms. *Designs, Codes and Cryptography*, pages 1–14, 2015.

[MBKM19] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019: 26th Conference on Computer and Communications Security*, pages 2111–2128. ACM Press, November 11–15, 2019.

[MP03] Daniele Micciancio and Erez Petrank. Simulatable commitments and efficient concurrent zero-knowledge. In *Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings*, pages 140–159, 2003.

[Par95] Ian Parberry. A real-time algorithm for the (n2- 1)-puzzle. *Information Processing Letters*, 56(1):23–28, 1995.

[Pas03a] Rafael Pass. On deniability in the common reference string and random oracle model. In *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 316–337. Springer, 2003.

[Pas03b] Rafael Pass. Simulation in quasi-polynomial time, and its application to protocol composition. In Eli Biham, editor, *Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings*, volume 2656 of *Lecture Notes in Computer Science*, pages 160–176. Springer, 2003.

[Pas04a] Rafael Pass. Alternative variants of zero-knowledge proofs. Master's thesis, Kungliga Tekniska Högskolan, 2004. Licentiate Thesis Stockholm, Sweden.

[Pas04b] Rafael Pass. Bounded-concurrent secure multi-party computation with a dishonest majority. In László Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 232–241. ACM, 2004.

[Ped91] Torben P. Pedersen. A threshold cryptosystem without a trusted party (extended abstract) (rump session). In Donald W. Davies, editor, *Advances in Cryptology – EUROCRYPT'91*, volume 547 of *Lecture Notes in Computer Science*, pages 522–526, Brighton, UK, April 8–11, 1991. Springer, Heidelberg, Germany.

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy*, pages 238–252, Berkeley, CA, USA, May 19–22, 2013. IEEE Computer Society Press.

[Pie19] Krzysztof Pietrzak. Simple verifiable delay functions. In Avrim Blum, editor, *ITCS 2019: 10th Innovations in Theoretical Computer Science Conference*, volume 124, pages 60:1–60:15, San Diego, CA, USA, January 10–12, 2019. LIPIcs.

[RCB16] Joost Renes, Craig Costello, and Lejla Batina. Complete addition formulas for prime order elliptic curves. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part I*, volume 9665 of *Lecture Notes in Computer Science*, pages 403–428, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany.

[SBG$^+$19] Samuel Steffen, Benjamin Bichsel, Mario Gersbach, Noa Melchior, Petar Tsankov, and Martin T. Vechev. zkay: Specifying and enforcing data privacy in smart contracts. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019: 26th Conference on Computer and Communications Security*, pages 1759–1776. ACM Press, November 11–15, 2019.

[Sch89] C.P. Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *Advances in Cryptology — CRYPTO' 89 Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 239–252. Springer New York, 1989.

[Sch99] J. Schaub. Implementiering von public-key-kryptosystemen über imaginär-quadratischen ordnungen (master's thesis). Technische Universität Darmstadt, Facbereich Informatik, 1999.

[Sch03] Daniel Schielzeth. Realisierung der elgamal-verschlüsselung in quadratischen zählkorpern (master's thesis). Technische Universität Berlin `http://www.math.tu-berlin.de/~kant/publications.html`, 2003.

[Sch18] Berry Schoenmakers. MPyC secure multiparty computation in Python. GitHub `https://github.com/lschoe/mpyc`, 2018.

[Ser77] Jean-Pierre Serre. *Linear representations of finite groups*, volume 42 of *Graduate texts in mathematics*. Springer, 1977.

[Sha71] Daniel Shanks. Class number, a theory of factorization, and genera. In *Proc. of Symp. Math. Soc., 1971*, volume 20, pages 41–440, 1971.

[SVdV16] Berry Schoenmakers, Meilof Veeningen, and Niels de Vreede. Trinocchio: Privacy-preserving outsourcing by distributed verifiable computation. In Mark Manulis, Ahmad-Reza Sadeghi, and Steve Schneider, editors, *ACNS 16: 14th International Conference on Applied Cryptography and Network Security*, volume 9696 of *Lecture Notes in Computer Science*, pages 346–366, Guildford, UK, June 19–22, 2016. Springer, Heidelberg, Germany.

[SZ04] Damien Stehlé and Paul Zimmermann. A binary recursive gcd algorithm. In Duncan Buell, editor, *Algorithmic Number Theory*, pages 411–425, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[Tof07] T. Toft. *Primitives and applications for multi-party computation*. PhD thesis, Aarhus University, 2007.

[VV09] Carmine Ventre and Ivan Visconti. Co-sound zero-knowledge with public keys. In *Progress in Cryptology - AFRICACRYPT 2009, Second International Conference on Cryptology in Africa, Gammarth, Tunisia, June 21-25, 2009. Proceedings*, pages 287–304, 2009.

[Wes19] Benjamin Wesolowski. Efficient verifiable delay functions. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part III*, volume 11478 of *Lecture Notes in Computer Science*, pages 379–407, Darmstadt, Germany, May 19–23, 2019. Springer, Heidelberg, Germany.