Efficient Non-Interactive Zero-Knowledge Proofs for Privacy Applications

by
Melissa Chase
B. S., Harvey Mudd College, 2003
Sc. M., Brown University, 2005

A dissertation submitted in partial fulfillment of the
requirements for the Degree of Doctor of Philosophy
in the Department of Computer Science at Brown University

Providence, Rhode Island
May 2008

This dissertation by Melissa Chase is accepted in its present form by
the Department of Computer Science as satisfying the dissertation requirement
for the degree of Doctor of Philosophy.

Date ________________ ____
Anna Lysyanskaya, Director

Recommended to the Graduate Council

Date ________________ ____
Roberto Tamassia, Reader

Date ________________ ____
Leonid Reyzin, Reader
(Boston University)

Approved by the Graduate Council

Date ________________ ____
Sheila Bonde
Dean of the Graduate School

Vita

Melissa Chase was born in Seattle, Washington in 1981, and in 1984 she moved to Southern California, where she spent the rest of her childhood. She was introduced to computer programming at an after-school club in sixth grade, and spent much of her middle school and high school years writing computer games with her younger brother. In 1999 she left home to attend Harvey Mudd College, intending to major in Computer Science or Medieval History. In 2003, she graduated with a B.S. with High Honors in Mathematics and Computer Science. She was awarded the National Science Foundation Graduate Research Fellowship, and began graduate study at Brown University. At Brown she discovered cryptography, and immediately decided that that was the area in which she wanted to work, as it combined number theory, algorithms, and complexity theory with practical applications. In the fall of 2006, she was awarded a fellowship from the Institute for Pure and Applied Mathematics at the University of California, Los Angeles, and spent the semester there attending a program on "Securing Cyberspace: Applications and Foundations of Cryptography and Computer Security". While at Brown, her work has appeared in numerous conference articles, and she has presented her research internationally in Great Britain, Denmark, Switzerland, and the Netherlands, as well as at a number of locations within the United States.

Acknowledgements

I would like to thank Anna Lysyanskaya for being a spectacular advisor, for introducing me to cryptography, and for encouragement and guidance over the last five years. I would also like to thank Leo Reyzin and Roberto Tamassia for being on my committee and for offering helpful advice. I thank my other collaborators both at Brown (Mira Belenkiy, Alptekin Kupcu, Chris Erway, and John Jannotti), and away (Jan Camenisch and Markulf Kohlweiss) for many interesting and enjoyable discussions. I would like to thank my parents, who have always been there for me, and my brother Jason for being my first collaborator. Finally, many thanks to Will Headden, Jenine Turner, Naomi Feldman, and especially Greg Cooper for being wonderful supportive and patient friends.

Contents

Chapter 1

Introduction

Modern cryptography focuses on the idea of provable security. We begin with the idea that there are certain problems that we can assume to be hard, e.g. factoring or hard-on-average NP-hard problems, then show that a successful attack on a particular cryptosystem would imply a solution to an underlying a hard problem. As long as the underlying problem is sufficiently hard, we can as a result guarantee that breaking our cryptosystem is also difficult.

However, it can be very difficult to analyze an entire system at once. Thus, we often find it useful to take a more modular approach: we break the system down into smaller building blocks, and show that if these building blocks satisfy certain properties, then the larger system is secure. This simultaneously makes proving security much simpler, and makes the basic intuition behind the construction of the system more apparent.

My research is concerned with provably secure cryptography. This work focuses on two new building blocks which we believe will be useful in a variety of privacy applications. The first defines a signature scheme for which we can issue efficient proofs that a message has been signed, without revealing any additional information. We show that this has applications in the area of anonymous credential systems — systems where users can anonymously obtain and prove possession of various types of credentials.

The second building block is a pseudo-random function together with an efficient proof system that allows users to prove that values have been computed correctly according to this function. This gives the privacy benefits of pseudorandomness while still preventing malicious behavior. We show that this results in the first provably secure compact e-cash scheme, and in addition that it can be used to build efficient proof systems for other languages.

Background

In many applications, maintaining privacy means allowing users to prove that they are acting according to a given protocol and yet to reveal only what information is necessary. One might, for example, want to prove that one has voted according to protocol without revealing one's vote, or to show that one has permission to access certain data without revealing one's entire identity. One solution in such cases is to use zero knowledge proof systems [53], in which a user can prove that a statement is true without revealing any other information.

In general, zero knowledge proof protocols are interactive in that the prover and verifier must exchange several rounds of communication before the verifier determines whether to accept the proof. However, certain applications require these proofs to be noninteractive – the prover must post a proof in such a way that any other user can verify it given access only to the proof and some trusted public parameters [8, 42, 7].

Blum et al. [7] introduced the notion of non-interactive zero knowledge (NIZK) proof systems and showed that such proof system exist for all languages in NP. However, the original NIZK proof systems [7, 46] were extremely inefficient, serving as proofs of concept rather than potentially usable constructions. Recent work has made general noninteractive zero knowledge (NIZK) proof systems significantly more efficient [56]. However, we can achieve systems which are far more efficient if we consider specialized NIZK proof systems for the specific types of statements required for individual privacy applications. A large part of this thesis focuses on identifying useful classes of statements for which we can give such proof systems.

Tools and Techniques.

One underlying tool in many of the proof systems I have worked on is the Groth-Sahai proof system[57]. Groth and Sahai give a particular commitment scheme[1] for bilinear groups, and a proof system in which a user can provide a list of commitments and then give a NIZK proof that these commitments can be opened to values that satisfy a given equation. The only requirement is that the equation must be of a particular form, which they refer to as "a tractable pairing product equation"[57]. Groth and Sahai show that this proof system is perfectly sound, computationally witness indistinguishable and in some cases computationally zero knowledge.

Here we will also define two additional properties satisfied by the Groth-Sahai proof system. The first is f-extraction, which generalizes the standard definition of extractability for proofs of knowledge and formalizes the notion that a prover can prove that he knows some function of a valid witness for a given statement (while he may not know the witness itself). The second is a property which we call randomizability, in which any proof can be "rerandomized" to produce what appears to be a completely fresh and independent proof for the same statement. Randomizability in particular seems to be a completely new concept in the context of non-interactive proofs. Finally, we will show several applications of our primitives which make crucial use of these properties.

P-signatures

Camenisch and Lysyanskaya [23, 24, 25] defined and constructed a primitive for a signature scheme with special properties. These CL signatures allow a user to obtain a signature on a committed message, and to prove knowledge of such a signature, in both cases without revealing any information about the message.

[1] In a cryptographic commitment scheme, a user first chooses a value and some randomness, and combines them to compute something called a commitment, which he can then publish. At some later point, he can decide to "open" the commitment by revealing the value and the randomness used. We require that the commitment scheme be hiding, which means that the commitment reveals no information about the value used to create it. At the same time, we require that it be binding, in that once a commitment is published, there should be at most one value to which he can successfully "open" the commitment.

They have been shown to be useful in a variety of privacy preserving applications [23, 24, 25, 2, 21, 75, 22, 29, 20].. However, these schemes require that the proofs be interactive; in cases where interaction is not possible, the proofs become only heuristically, not provably, secure.

In joint work with Belenkiy, Kohlweiss, and Lysyanskaya, I present a new signature and proof system that allows noninteractive provably secure proofs of knowledge of a signature, which we call P-signatures [5]. Our P-signature can be seen as the analog of CL signature in the non-interactive proof setting.

However, the non-interactive proof setting brings additional complications, particularly when combined with proof systems which are only f-extractable. Note that the underlying signature scheme must have a unforgeability property which is strictly stronger than the standard notion introduced by Goldwasser, Micali, and Rivest[54]. We must show not only that an adversarial party cannot produce a message and a forged signature, but also that he cannot produce a commitment and a proof of a forged signature on the corresponding message, which is potentially a much stronger statement. We provide several constructions for signatures that satisfy this stronger notion. This results in several P-signatures schemes based on different assumptions and applicable in a variety of different settings.

Application to Anonymous credentials

Anonymous credential systems allow a user to prove that he has valid credentials for a service without revealing his identity, or any other information about himself. Camenisch and Lysyanskaya [23] developed an anonymous credential system based on CL signatures.

We show that our P-signatures signatures can give a credential system in which a user can issue noninteractive proofs of valid credentials, which suggests that they can be used in place of CL signatures in situations where we need secure noninteractive proofs. Previously, the only known constructions for such a primitive were based on CL signatures and only heuristically secure (in that they require the random oracle model). We also show that these P-signatures can solve a problem for which no solution was previously known to exist in any setting: the problem of delegatable anonymous credentials.

Application to Delegatable anonymous credentials

Delegatable anonymous credentials are a logical extension of anonymous credentials. Consider the following scenario: Suppose we want to use anonymous credentials to implement an anonymous review system for some conference. The program chair can issue reviewer credentials for the appropriate papers to each member of the committee, who then can prove to the server that they are entitled to upload reviews without revealing their identities. These committee members can in turn issue credentials to any sub-reviewers they recruit. The sub-reviewer does not need to know to whom the paper was originally assigned, only that that person asked them to sub-review. (Note that this is different from traditional credential systems where the user always knows the identity of the party that issues his credentials.) The sub-reviewer could in turn delegate the paper to a second level sub-reviewer, and so on. The server should keep track of which reviews are made by committee members, which by sub-reviewers, and which by sub-sub-reviewers, but should not receive any information about the identities of the reviewers. Thus, we have a credential chain, from chair, to committee member, to sub-reviewer, to sub-sub-reviewer, and each participant in the chain does not know who gave him/her the credential. However, each participant (1) knows the length of his credential chain and knows that this credential chain is rooted at the program chair; and (2) can extend the chain and issue a credential to the next person. Furthermore, we consider an even stronger notion where the issuer or delegator only knows a pseudonym of the person to whom they're delegating, so that the credentials are anonymous both from the point of view of the delegator and of the recipient.

In joint work with Lysyanskaya [31], I gave the first solution to this problem by showing that such a system could be constructed from a primitive that we called signatures of knowledge. However, this construction relies on general simulation sound NIZK proof systems which are extremely inefficient. Here in joint work with Belenkiy, Camenisch, Kohlweiss, Lysyanskaya, and Shacham [3], I show that an extension of P-signatures can be used to build efficient delegatable anonymous credentials. The key to this construction is the randomizability property mentioned above.

Simulatable Verifiable Random Functions

As a second direction, I have been looking at ways to allow a user to choose pseudorandom values and to prove that this has been done honestly. Randomness has been shown to be essential for cryptographically secure privacy, but, as it is considered an expensive resource, often a short random seed is used to generate a pseudorandom function (PRF) [50] — a function which is indistinguishable from a truly random function — giving many seemingly random values. When pseudorandom functions are used as part of larger protocols, it is often possible that a party can cheat by choosing a value in some adversarial way rather than according to the pseudorandom function. Verifiable random functions (VRFs)[66] begin to solve this problem by giving a public key corresponding to each seed and allowing the party to prove that a value has been computed correctly according to a particular public key. However, this comes at the cost of some of the pseudorandomness guarantees — as soon as a proof for a particular value has been published, that value is no longer guaranteed to be pseudorandom.

Thus, in joint work with Lysyanskaya, I introduce a new primitive called simulatable verifiable random functions (sVRFs) [32]. Simulatable verifiable random functions are essentially pseudorandom functions for which a party can efficiently give a NIZK proof that the output was computed correctly (although we also consider somewhat weaker definitions). We now have several constructions for this primitive — the first based on composite order bilinear groups, a newer construction based on the Groth-Sahai proof system, and a proof of concept construction based on general assumptions.

Application to E-cash.

One practical application is that of electronic cash or e-cash [36]. Here we want an electronic currency that is both anonymous (the recipient of a coin cannot tell which user the coin came from, or whether two coins were spent by the same person, even if he colludes with the bank), and unforgeable (a user can only spend as many coins as he has withdrawn from the bank). Camenisch, Hohenberger, and Lysyanskaya [21] gave an e-cash scheme in which users can efficiently withdraw and store large wallets of coins. To withdraw a wallet, the user obtains a signature from the bank on two random seeds. Then each coin spent is identified with a serial number generated by a pseudorandom function using one of the two seeds. The user must provide a proof that this value has been computed correctly and that the seed used has been signed. In the current constructions, this proof system is only heuristically secure. In joint work with Belenkiy, Kohlweiss, and Lysyanskaya, I show that if we replace the PRF with an sVRF and use a P-signature to prove knowledge of a signature on the seed, we can construct an e-cash scheme which is actually provably secure.

What makes this more complex is that each coin must also include a second value called the double spending equation. For the ith coin in a user's wallet, this value is computed as a function of a second PRF F with seed s_2, some transaction specific information R, and an identifier ID which uniquely identifies the user, as follows: $T = F_{s_2}(i)^R * ID$. Note that if the same coin is used in two different transactions, we can use the two transaction values R and R' together with the double spending equations T and T' from each transaction to determine the identity of the doublespender.

Thus, we also need each user to include a proof that the doublespending equation was computed correctly, even though it is not the direct output of F, but rather a function of that output. In general, it seems that there might be many other protocols in which the user must prove, not that a certain value was generated honestly at random, but rather that the value was the result of some computation one of whose inputs was chosen pseudorandomly. We show that we can generalize one of our sVRF constructions to deal which such a situation.

Application to efficient NIZK for other languages

Finally, in joint work with Lysyanskaya [32], I show that sVRFs can be used to help build efficient NIZK proof systems. If a particular language has an efficient proof system that can prove in zero knowledge that a single statement holds, then we can use sVRFs to build an efficient proof system for many statements. This leaves open the problem of finding useful languages with efficient single statement proof systems (which seems to be a much easier problem than finding efficient multi-statement proof systems).

Organization

In Chapter 2, we will summarize our notation, review definitions for basic cryptographic primitives and for cryptographic bilinear maps, and present and discuss the assumptions on which our constructions will be based. In Chapter 3, we summarize the Groth-Sahai proof system [57], describe our two new properties (f-extraction and randomizability), and prove that they are satisfied. Chapter 4 presents definitions and constructions for P-signatures, while Chapter 5 presents definitions and definitions for our sVRFs. Finally, Chapters 6, 7, 8, 9 describe applications to non-interactive anonymous credentials, delegatable anonymous credentials, efficient NIZK for other languages, and provably secure e-cash, respectively.

Chapter 2

Preliminaries

2.1 Basic Cryptographic Primitives

Here we will review several basic cryptographic primitives. We will present basic intuition and definitions for commitment schemes, digital signatures, and non-interactive proof systems.

2.1.1 Commitment Schemes

In a non-interactive commitment scheme, one user can "commit" to a value x, using the algorithm Commit on input x and some randomness r to produce a commitment c which he can then send to a verifier. At some later point, the user can choose to "open" the commitment by sending x and r to the verifier, who checks that those values correctly correspond to the initial commitment c. At a high level, the properties we want are (1) hiding: when the verifier has received only the commitment c, he should have no information about which value x the user committed to, and (2) binding: once the user has formed and sent the commitment c, there should be at most one value x to which he can "open" the commitment, i.e. at most one value x for which he can produce an r such that x, r will be accepted by the verifier. (The randomness r is often referred to as the opening information for this commitment.)

More formally, a non-interactive commitment scheme consists of PPT algorithm ComSetup and deterministic polynomial time algorithm Commit as follows:

$\mathsf{ComSetup}(1^k)$ takes as input the security parameter and outputs public parameters $params_{Com}$ for the commitment scheme.

$\mathsf{Commit}(params_{Com}, x, open)$ is a deterministic function that takes as input a value x and randomness $open$, and outputs $comm$, a commitment to x using randomness $open$.

We will need commitment schemes that are *perfectly binding* and *strongly computationally hiding*:

Definition 1. *A commitment scheme* (ComSetup, Commit) *is perfectly binding if for every bitstring* $comm$*, there exists at most one value* x *such that there exists randomness* $open$ *so that* $comm = \mathsf{Commit}(params, x, open)$.

More formally, for all (potentially unbounded) adversaries $\mathcal{A}$,

$$\Pr[params_{Com} \leftarrow \mathsf{ComSetup}(1^k); comm, x, x', open, open' \leftarrow \mathcal{A}(params_{Com}) \\ : x \neq x' \wedge comm = \mathsf{Commit}(x, open) \wedge comm = \mathsf{Commit}(x', open')] = 0$$

Definition 2. *A commitment scheme* $(\mathsf{ComSetup}, \mathsf{Commit})$ *is strongly computationally hiding if there exists an alternate setup* $\mathsf{HidingSetup}(1^k)$ *that outputs parameters (computationally indistinguishable from the output of* $\mathsf{ComSetup}(1^k)$*) so that the commitments become information-theoretically hiding. More formally, we require that the following properties hold:*

(1) $\{params \leftarrow \mathsf{ComSetup}(1^k) : params\} \approx \{params \leftarrow \mathsf{HidingSetup}(1^k) : params\}$

(2) let R be the set from which the randomness open is chosen. Then for all x_1, x_2 in the domain of the commitment scheme

$$\{params \leftarrow \mathsf{HidingSetup}(1^k); open \leftarrow R; c = \mathsf{Commit}(params, x_1, open) : c\} \\ = \{params \leftarrow \mathsf{HidingSetup}(1^k); open \leftarrow R; c = \mathsf{Commit}(params, x_2, open) : c\}$$

2.1.2 Secure Digital Signature

In a digital signature scheme, a signer can generate a private signing key sk and publish a corresponding public verification key pk. The signer can use sk to generate a signature σ for each message m, and any recipient can use pk to verify that a given signature σ was generated by the owner of the corresponding sk as a signature on the message m.

In the common parameters model, a signature scheme consists of four algorithms:

$\mathsf{SigSetup}(1^k)$ takes as input the security parameter, and produces system parameters $params_{Sig}$.

$\mathsf{Keygen}(params_{Sig})$ takes as input the system parameters and produces a secret key sk and a public key pk.

$\mathsf{Sign}(params_{Sig}, sk, m)$ takes as input the system parameters, a signing key sk, and a message m. It produces a signature σ.

$\mathsf{VerifySig}(params_{Sig}, pk, m, \sigma)$ takes as input the system parameters, a verification key pk, a message m, and a candidate signature σ. It outputs 1 if σ is a valid signature on message m under verification key pk, and 0 otherwise.

Definition 3 (Secure Signature Scheme [54]). *We say that a signature scheme is secure (against adaptive chosen message attacks) if it is* Correct *and* Unforgeable.

Correctness. All signatures obtained using the Sign algorithm should be accepted by the VerifySig algorithm. More formally, for all m,

$$\Pr[params_{Sig} \leftarrow \mathsf{SigSetup}(1^k); (pk, sk) \leftarrow \mathsf{Keygen}(params_{Sig}); \\ \sigma \leftarrow \mathsf{Sign}(params_{Sig}, sk, m) : \mathsf{VerifySig}(params_{Sig}, pk, m, \sigma) = 1] = 1$$

Unforgeability. No adversary should be able to output a valid message/signature pair (m, σ) unless he has previously obtained a signature on m. Formally, for every PPTM adversary $\mathcal{A}$, there exists a negligible function ν such that

$$\begin{aligned}&\Pr[params_{Sig} \leftarrow \mathsf{SigSetup}(1^k); (pk, sk) \leftarrow \mathsf{Keygen}(params_{Sig});\\&(Q_{\mathsf{Sign}}, m, \sigma) \leftarrow \mathcal{A}(params_{Sig}, pk)^{\mathcal{O}_{\mathsf{Sign}}(params_{Sig}, sk, \cdot)} :\\&\mathsf{VerifySig}(params_{Sig}, pk, m, \sigma) = 1 \wedge m \notin Q_{\mathsf{Sign}}] < \nu(k).\end{aligned}$$

$\mathcal{O}_{\mathsf{Sign}}(params_{Sig}, sk, m)$ records m on Q_{Sign} and returns $\mathsf{Sign}(params_{Sig}, sk, m)$.

2.1.3 Non-interactive Proof Systems

In this section, we review security definitions for non-interactive proof systems in the common reference string model [7]. Here we assume that some trusted party has computed a set of public parameters according to an algortihm Setup, and posted them so that they are accessible to all parties in the system. The prover wants to prove some statement of the form "$x \in L$" for an NP language L. Equivalently, we can say that for a given x and a given polynomial time Turing machine M_L, the prover wants to prove a statement of the form "$\exists w$ such that $M_L(x, w) = 1$".

Thus, we have three probabilistic polynomial time (PPT) algorithms:

$\mathsf{Setup}(1^k)$ takes as input a security parameter k. It produces public parameters $params$. If these parameters are chosen at random from $\{0, 1\}^{l(k)}$ for some polynomial l, then this is said to be the common *random* string model (rather than the common *reference* string model).

$\mathsf{Prove}(params, x, w, M_L)$ takes as input the public parameters for the system, and an instance x and a witness w such that $M_L(x, w) = 1$. It outputs a proof π.

$\mathsf{VerifyProof}(params, x, \pi, M_L)$ takes as input the public parameters for the system, an instance x, a TM M_L and a candidate proof of the statement: "$\exists w$ such that $M_L(x, w) = 1$". It outputs accept if it accepts the proof and reject otherwise.

When the language is clear from context, we will omit the input M_L.

We consider proof systems with a variety of different security properties, which we will summarize below.

Completeness

Informally, we say that a proof system satisfies the completeness property if any proof produced by an honest prover with a valid witness for the given statement will be accepted by an honest verifier.

Definition 4. *A non-interactive proof system* ($\mathsf{Setup}, \mathsf{Prove}, \mathsf{VerifyProof}$) *for language L is complete if the following holds:*

For all x, w such that $M_L(x, w) = 1$

$$\Pr[params \leftarrow \mathsf{Setup}(1^k); \pi \leftarrow \mathsf{Prove}(params, x, w, M_L) : \mathsf{VerifyProof}(params, x, \pi, M_L) = 1] = 1$$

where M_L is the TM which accepts x, w iff w is a witness for the statement $x \in L$

Soundness

Informally, a proof system is computationally sound if for any $x \notin L$, a PPT adversary has only negligible probability of producing a proof for $x \in L$ that will be accepted by the honest verifier. A proof system is perfectly sound if no (potentially unbounded) adversary can produce a proof for a $x \notin L$ that will be accepted by the honest verifier.

Definition 5. *A non-interactive proof system* $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{VerifyProof})$ *for language L is called computationally sound if the following holds:*

For all probabilistic polynomial time adversaries $\mathcal{A}$,

$$\begin{aligned}&\Pr[params \leftarrow \mathsf{Setup}(1^k); (\pi, x, L) \leftarrow \mathcal{A}(params, x)\\ &\quad : x \notin L \wedge \mathsf{VerifyProof}(params, x, \pi, M_L) = 1] \leq \nu(k),\end{aligned}$$

where M_L is the TM which accepts (x, w) iff w is a witness for the statement $x \in L$.

We sometimes say that such a proof system has soundness error $\nu(k)$.

Definition 6. *A non-interactive proof system* $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{VerifyProof})$ *is called perfectly sound if the above holds for $\nu(k) = 0$ for all (potentially unbounded) adversaries.*

Witness Indistinguishability

Informally, witness indistinguishability says if there exist two witnesses w_1, w_2 for the statement $x \in L$, no PPT adversary can tell whether a given proof was produced using w_1 or w_2.

Definition 7. *A non-interactive proof system* $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{VerifyProof})$ *is witness indistinguishable (WI) for language L if the following holds:*

For all PPT adversaries $\mathcal{A}_1, \mathcal{A}_2$, there exists a negligible function ν such that:

$$\begin{aligned}&\Pr[params \leftarrow \mathsf{Setup}(1^k); (x, w_1, w_2, state) \leftarrow \mathcal{A}_1(params, x, M_L); b \leftarrow \{0,1\};\\ &\quad \pi \leftarrow \mathsf{Prove}(params, x, w_b, M_L); b' \leftarrow A_2(state, \pi) : M_L(x, w_1) = M_L(x, w_2) = 1 \wedge b = b']\\ &\quad = \frac{1}{2} + \nu(k),\end{aligned}$$

where M_L is the TM which accepts x, w iff w is a witness for the statement $x \in L$.

Definition 8. *A proof system is perfectly witness indistinguishable if the above holds for $\nu(k) = 0$ for all (potentially unbounded) adversaries.*

Zero Knowledge

Zero knowledge is a somewhat stronger notion in which we require (informally) that the verifier should learn nothing besides that the statement is true. We say that there should be simulator who can create proofs given only the statement $x \in L$ (but not the witness w) such that these proofs are indistinguishable from those given by an honest prover. Note that the simulator must have some power that an adversarial prover would not have, as a prover should not be able to produce such a proof. Thus, we also allow the simulator to choose the common reference string and to store some trapdoor information about it, with the requirement that the resulting string should be indistinguishable from the output of the original Setup algorithm.

Here we distinguish between proof systems which are zero knowledge when the common reference string is used only for a single proof, and those which allow for many proofs using the same common reference string.

Definition 9. *A non-interactive proof system* $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{VerifyProof})$ *is a single-theorem zero knowledge proof system for language* L *if there exists PPT simulator algorithm* `SimProveOne` *such that the following holds:*

For all $x \in L$ *and all* w *such that* $M_L(x, w) = 1$*, for all adversaries* $\mathcal{A}$*, the following two distributions are indistinguishable:*

$$\textit{Real}(k) = \{params \leftarrow \mathsf{Setup}(1^k); \pi \leftarrow \mathsf{Prove}(params, x, w, M_L) : (params, \pi)\}$$
$$\textit{and Sim}(k) = \{(params, \pi) \leftarrow \mathtt{SimProveOne}(1^k, x, M_L) : (params, \pi)\}.$$

Definition 10. *A non-interactive proof system* $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{VerifyProof})$ *is a multi-theorem zero knowledge (NIZK) proof system for language* L *if there exists PPT simulator algorithms* $\mathsf{SimSetup}, \mathsf{SimProve}$ *such that the following holds:*

For all PPT adversaries $\mathcal{A}$*, there exists a negligible function* ν *such that:*

$$\begin{aligned}&|\Pr[params \leftarrow \mathsf{Setup}(1^k); b \leftarrow \mathcal{A}^{\mathsf{Prove}(params,\cdot,\cdot,M_L)} : b = 1]\\ &\quad - \Pr[(params, s) \leftarrow \mathsf{SimSetup}(1^k); b \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Sim}}(s,params,\cdot,\cdot,M_L)} : b = 1]| < \nu(k)\end{aligned}$$

where M_L *is the TM which accepts* x, w *iff* w *is a witness for the statement* $x \in L$*, and* $\mathcal{O}_{\mathsf{Sim}}(s, params, x, w, M_L)$ *checks that* $M_L(x, w) = 1$ *and then runs* $\mathsf{SimProve}(params, s, x, M_L)$ *and returns the resulting proof* π*.*

In what follows, if we say simply "zero knowledge" or NIZK, we will be referring to multi-theorem zero-knowledge.

Finally, we recall the notion of composable zero knowledge introduced by Groth [55] which is satisfied by some of the GS proofs. The idea is that it is easier to prove zero knowledge if we do not have to consider the entire multi-theorem zero knowledge game (in which we generate parameters and then prove many theorems) at once. Instead, we first show that the simulated parameters are indistinguishable from the real parameters. Then we consider a setting where both the honest prover and the simulated prover use the simulated parameters. We show that each individual simulated proof is indistinguishable from an honest proof even when the adversary is given the simulation trapdoor. It can be shown by a simple hybrid argument that this implies the multi-theorem zero-knowledge property described above.

Definition 11. *A non-interactive proof system* $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{VerifyProof})$ *is a composable zero knowledge (NIZK) proof system for language* L *if there exists PPT simulator algorithms* $\mathsf{SimSetup}, \mathsf{SimProve}$ *such that the following holds:*

(1) The following two distributions are indistinguishable:

$$\mathit{Real}_{params}(k) = \{params \leftarrow \mathsf{Setup}(1^k) : params\}$$
$$\textit{and } \mathit{Sim}_{params}(k) = \{(params, s) \leftarrow \mathsf{SimSetup}(1^k) : params\}$$

(2) For all $x \in L$ with valid witness w, the following distributions are indistinguishable:

$$\mathcal{R}_{\mathit{proof}}(k) = \{(params, s) \leftarrow \mathsf{SimSetup}(1^k); \pi \leftarrow \mathsf{Prove}(params, x, w, M_L) : \pi\}$$
$$\textit{and } \mathsf{Sim}_{\mathit{proof}}(k) = \{(params, s) \leftarrow \mathsf{SimSetup}(1^k); \pi \leftarrow \mathsf{SimProve}(params, s, x, M_L) : \pi\},$$

where M_L is the TM which accepts x, w iff w is a witness for the statement $x \in L$.

Proofs of Knowledge

We review the definition of a noninteractive proof of knowledge (NIPK) introduced by De Santis et al. [71]. Here we want to prove not only that a statement is true, but also that the prover "knows" a valid witness for the statement. We formalize this by saying that there is a PPT extractor algorithm which, given a proof for a given statement which is accepted by an honest verifier together with some trapdoor information about the common reference string, can extract a valid witness for the statement.

Definition 12 (Extractability)**.** *A non-interactive proof system* $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{VerifyProof})$ *is a proof of knowledge for language L if there exists a polynomial-time extractor* $(\mathsf{ExtractSetup}, \mathsf{Extract})$ *such that the following hold: (1)*

$$\{params \leftarrow \mathsf{Setup}(1^k) : params\} \approx \{(params, t) \leftarrow \mathsf{ExtractSetup} : params\}$$

and (2) For all PPT adversaries $\mathcal{A}$ there exists negligible ν such that,

$$\Pr[(params, t) \leftarrow \mathsf{ExtractSetup}; (x, \pi) \leftarrow \mathcal{A}(params); w \leftarrow \mathsf{Extract}(params, t, x, \pi)$$
$$: \mathsf{VerifyProof}(params, x, \pi, M_L) = 1 \wedge M_L(x, w) = 0] = \nu(k)$$

where M_L is the TM which accepts x, w iff w is a witness for the statement $x \in L$.

Definition 13. *We have* perfect *extractability if the above holds for $\nu(k) = 0$ even for potentially unbounded adversaries.*

2.2 Bilinear Maps

Let G_1, G_2, and G_T be groups. A function $e : G_1 \times G_2 \rightarrow G_T$ is called *a cryptographic bilinear map* if it has the following properties: *Bilinear.* $\forall a \in G_1$, $\forall b \in G_2$, $\forall x, y \in \mathbb{Z}$ the following equation holds: $e(a^x, b^y) = e(a, b)^{xy}$. *Non-Degenerate.* If a and b are generators of their respective groups, then $e(a, b)$ generates G_T. Let $\mathsf{BilinearSetup}(1^k)$ be an algorithm that generates the groups G_1, G_2 and G_T, together with algorithms for sampling from these groups, and the algorithm for computing the function e.

The function $\mathsf{BilinearSetup}(1^k)$ outputs $params_{BM} = (p, G_1, G_2, G_T, e, g, h)$, where p is a prime (of length k), G_1, G_2, G_T are groups of order p, g is a generator of G_1, h is a generator of G_2, and $e : G_1 \times G_2 \rightarrow G_T$ is a bilinear map.

2.3 Assumptions

2.3.1 Previous Assumptions

Boyen and Waters [14] defined the Hidden SDH assumption over bilinear maps using symmetric groups $e : G \times G \rightarrow G_T$. We give a definition over asymmetric maps $e : G_1 \times G_2 \rightarrow G_T$. Note that in the symmetric setting, this is identical to the Boyen Waters HSDH assumption.

Definition 14 (Hidden SDH)**.** *On input $g, g^x, u \in G_1$, $h, h^x \in G_2$ and $\{g^{1/(x+c_\ell)}, h^{c_\ell}, u^{c_\ell}\}_{\ell=1\dots q}$, it is computationally infeasible to output a new tuple $(g^{1/(x+c)}, h^c, u^c)$. Formally, we say the HSDH assumption holds for groups output by* $\mathsf{BilinearSetup}$ *if there exists a negligible function ν such that*

$$\begin{aligned}
&\Pr[(p, G_1, G_2, G_T, e, g, h) \leftarrow \mathsf{BilinearSetup}(1^k);\\
&\quad u \leftarrow G_1; x, \{c_\ell\}_{\ell=1\dots q} \leftarrow Z_p;\\
&\quad (A, B, C) \leftarrow \mathcal{A}(p, G_1, G_2, G_T, e, g, g^x, h, h^x, u, \{g^{1/(x+c_\ell)}, g^{c_\ell}, u^{c_\ell}\}_{\ell=1\dots q}) :\\
&\quad (A, B, C) = (g^{1/(x+c)}, h^c, u^c) \wedge c \notin \{c_\ell\}_{\ell=1\dots q}] < \nu(k).
\end{aligned}$$

When $(p, G_1, G_2, G_T, e, g, h)$ and $H = h^x$ are fixed, we refer to tuples of the form $(g^{1/(x+c)}, h^c, u^c)$ as *HSDH tuples*.

Note that we can determine whether (A, B, C) form an HSDH tuple using the bilinear map e, as follows: suppose we get a tuple (A, B, C). We check that $e(A, BH) = e(g, h)$ and that $e(u, B) = e(C, h)$.

We recall the Bilinear Diffie-Hellman Inversion (BDHI) assumption and the Subgroup Decision Assumption (SDA).

Definition 15 ((Q, ν)-BDHI [10])**.** *The groups output by algorithm* $\mathsf{BilinearSetup}$ *satisfy the $(Q(k), \nu(k))$-bilinear Diffie-Hellman inversion assumption if no PPT $\mathcal{A}$, on input $(instance, challenge)$ can distinguish if its challenge is of type 1 or type 2 with advantage asymptotically higher than $\nu(k)$ where $instance$ and $challenge$ are defined as follows: $instance = (G_1, G_2, q, e, g, g^\alpha, g^{\alpha^2}, g^{\alpha^3}, \dots, g^{\alpha^{Q(k)}})$ where q is a prime of length poly(k), G_1, G_2 are groups of order q returned by* $\mathsf{BilinearSetup}(1^k)$*, $e : G_1 \times G_1 \rightarrow G_2$ is a bilinear map, $g \leftarrow G_1$, $\alpha \leftarrow \mathbb{Z}_q^*$, challenge of type 1 is $e(g, g)^{\frac{1}{\alpha}}$, while challenge of type 2 is $e(g, g)^R$ for random $R \leftarrow \mathbb{Z}_q^*$.*

Definition 16 (SDA [13])**.** *The groups output by the algorithm* $\mathsf{BilinearSetup}$ *satisfy the subgroup decision assumption if no PPT $\mathcal{A}$, on input $(instance, challenge)$ can distinguish if its challenge is of type 1 or type 2, where $instance$ and $challenge$ are defined as follows: $instance = (G_1, G_2, n, e, h)$ where $n = pq$ is a product of two primes of length poly(k) (for k a sec. param.), G_1, G_2 are groups of order n returned by* $\mathsf{BilinearSetup}(1^k)$*, $e : G_1 \times G_1 \rightarrow G_2$ is a bilinear map, h is a random generator of G_1, challenge of type 1 is g, a random generator of G_1, while challenge of type 2 is g_p, a random order-p element of G_1.*

The following two assumptions are used to instantiate the Groth-Sahai proof systems described in Chapter 3).

Definition 17 (Decisional Linear Assumption [12])**.** *On input $u, v, w, u^r, v^s \leftarrow G_1$ it is computationally infeasible to distinguish $z_0 \leftarrow w^{r+s}$ from $z_1 \leftarrow G_1$. The assumption is analogously defined for G_2.*

Formally, we say the Decisional Linear Assumption holds with respect to groups output by BilinearSetup *if there exists a negligible function ν such that*

$$\begin{aligned}&\Pr[(p, G_1, G_2, G_T, e, g, h) \leftarrow \mathsf{BilinearSetup}(1^k); r, s \leftarrow Z_p; u, v, w \leftarrow G_1;\\&\quad b \leftarrow \{0,1\}; z_0 \leftarrow w^{r+s}; z_1 \leftarrow G_1 :\\&\quad \mathcal{A}(p, G_1, G_2, G_T, e, g, h, u, v, w, u^r, v^s, z_b) = b] < 1/2 + \nu(k).\end{aligned}$$

SXDH states that the Decisional Diffie Hellman problem is hard in both G_1 and G_2. This precludes efficient isomorphisms between these two groups.

Definition 18 (External Diffie-Hellman Assumption (XDH))**.** *We say the XDH assumption holds with respect to group G_1 output by* BilinearSetup *if there exists a negligible function ν such that*

$$\begin{aligned}&\Pr[(p, G_1, G_2, G_T, e, g, h) \leftarrow \mathsf{BilinearSetup}(1^k); r, s \leftarrow Z_p;\\&\qquad b \leftarrow \{0,1\}; z_0 \leftarrow g^{rs}, z_1 \leftarrow G_1 : \mathcal{A}(p, G, G_T, e, g, g^r, g^s, z_b) = b] < 1/2 + \nu(k).\end{aligned}$$

The XDH assumption can be similarly defined to hold in G_2. The **SXDH** assumption states that XDH holds in both G_1 and G_2. The SXDH assumption was first used by Scott [72], and has been discussed and used extensively since [12, 49, 74, 1].

2.3.2 New Assumptions

Here we introduce several new assumptions.

We relax the HSDH assumption and introduce a new assumption we call the BB-HSDH assumption. Intuitively speaking, we allow the adversary to obtain the values c_ℓ used in his challenge. We call this assumption Boneh-Boyen HSDH, because the adversary is given Weak Boneh Boyen signatures $(g^{1/(x+c_\ell)}, c_\ell)$ for random messages c_ℓ.

Definition 19 (BB-HSDH)**.** *Let $c_1 \dots c_q \leftarrow Z_p$. On input $g, g^x, u \in G_1$, $h, h^x \in G_2$ and $\{g^{1/(x+c_\ell)}, c_\ell\}_{\ell=1\dots q}$, it is computationally infeasible to output a new tuple $(g^{1/(x+c)}, h^c, u^c)$.*

Formally, we say the BB-HSDH assumption holds for groups output by algorithm BilinearSetup *if there exists a negligible function ν such that*

$$\begin{aligned}&\Pr[(p, G_1, G_2, G_T, e, g, h) \leftarrow \mathsf{BilinearSetup}(1^k);\\&\quad u \leftarrow G_1; x, \{c_\ell\}_{\ell=1\dots q} \leftarrow Z_p;\\&\quad (A, B, C) \leftarrow \mathcal{A}(p, G_1, G_2, G_T, e, g, g^x, h, h^x, u, \{g^{1/(x+c_\ell)}, c_\ell\}_{\ell=1\dots q}) :\\&\quad (A, B, C) = (g^{1/(x+c)}, h^c, u^c) \wedge c \notin \{c_\ell\}_{\ell=1\dots q}] < \nu(k).\end{aligned}$$

It is easy to see that the BB-HSDH implies HSDH. Thus our generic group proof for BB-HSDH (See Appendix A) also establishes generic group security for HSDH.

We also introduce a new assumption, we call BB-CDH. It is a relaxed version of CDH, in which the adversary is also given q weak BB signatures as input.

Definition 20 (BB-CDH)**.** *On input $g, g^x, g^y \in G_1$, $h, h^x \in G_2$, $c_1, \ldots, c_q \leftarrow Z_q$ and $g^{\frac{1}{x+c_1}}, \ldots, g^{\frac{1}{x+c_q}}$, it is computationally infeasible to output a g^{xy}.*

Formally, we say the BB-CDH assumption holds for groups output by algorithm BilinearSetup *if there exists a negligible function ν such that*

$$\begin{aligned}
&\Pr[(p, G_1, G_2, G_T, e, g, h) \leftarrow \mathsf{BilinearSetup}(1^k);\\
&\quad x, y, \{c_\ell\}_{\ell=1\ldots q} \leftarrow Z_p;\\
&\quad A \leftarrow \mathcal{A}(p, G_1, G_2, G_T, e, g, g^x, g^y, h, h^x, \{g^{1/(x+c_\ell)}, c_\ell\}_{\ell=1\ldots q}) :\\
&\quad A = g^{xy}] < \nu(k).
\end{aligned}$$

We can show that this is implied by the SDH assumption (See Appendix A). We present it as separate assumption, as this simplifies the proofs and intuition behind our constructions. Note that we obtain generic group security from the generic group proof for SDH.

We extend the HSDH assumption further and introduce a new stronger assumption we call the Interactive HSDH assumption. We allow the adversary to adaptively query an oracle for HSDH triples on c_i of his choice.

Definition 21 (Interactive Hidden SDH (IHSDH) assumption.)**.** *No PPTM adversary can compute a tuple $(g^{1/(x+c)}, h^c, u^c)$ given (g, g^x, h, h^x, u) and permission to make q queries to oracle $\mathcal{O}_x(c)$ that returns $g^{1/(x+c)}$. The c used by the adversary must be different from the values it used to query $\mathcal{O}_x(\cdot)$. Formally, we say the IHSDH assumption holds for groups output by* BilinearSetup *if there exists a negligible function ν such that*

$$\begin{aligned}
&\Pr[(p, G_1, G_2, G_T, e, g, h) \leftarrow \mathsf{BilinearSetup}(1^k);\\
&\qquad x \leftarrow Z_p; u \leftarrow G_1; (A, B, C) \leftarrow \mathcal{A}^{\mathcal{O}_x(\cdot)}(p, G_1, G_2, G_T, e, g, g^x, h, h^x, u) :\\
&\qquad\qquad \exists c \,:\, (A, B, C) = (g^{1/(x+c)}, h^c, u^c)] < \nu(k).
\end{aligned}$$

We introduce another new assumption, we call the Triple DH, which is a slightly stronger variant of BB-CDH.

Definition 22 (Triple DH (TDH))**.** *On input $g, g^x, g^y, h, h^x, \{c_i, g^{1/(x+c_i)}\}_{i=1\ldots q}$, it is computationally infeasible to output a tuple $(h^{\mu x}, g^{\mu y}, g^{\mu x y})$ for $\mu \neq 0$. Formally, we say the TDH assumption holds on groups output by* BilinearSetup *if there exists a negligible function ν such that*

$$\begin{aligned}
&\Pr[(p, G_1, G_2, G_T, e, g, h) \leftarrow \mathsf{BilinearSetup}(1^k);\\
&\quad (x, y) \leftarrow Z_p; \{c_i\}_{i=1\ldots q} \leftarrow Z_p;\\
&\quad (A, B, C) \leftarrow \mathcal{A}(p, G_1, G_2, G_T, e, g, g^x, g^y, h, h^x, \{c_i, g^{1/(x+c_i)}\}_{i=1\ldots q}) :\\
&\quad \exists \mu \neq 0 : (A, B, C) = (h^{\mu x}, g^{\mu y}, g^{\mu x y})] < \nu(k).
\end{aligned}$$

Definition 23 ((Q, ν)-BDHBI)**.** *We say the groups output by* BilinearSetup *satisfy the $(Q(k), \nu(k))$ bilinear Diffie-Hellman* basegroup *inversion assumption if no PPT $\mathcal{A}$, on input $(instance, challenge)$ can distinguish* *if its challenge is of type 1 or type 2 with advantage asymptotically higher than $\nu(k)$, where $instance$ and $challenge$ are defined as follows: $instance = (G_1, G_2, q, e, g, g^{\alpha}, g^{\alpha^2}, g^{\alpha^3}, \ldots, g^{\alpha^{Q(k)}}, g^{\beta})$ where q is a prime of length $poly(k)$, G_1, G_2 are groups of order q returned by $\mathcal{G}(q)$, $e : G_1 \times G_1 \rightarrow G_2$ is a bilinear map, $g \leftarrow G_1$, $\alpha \leftarrow \mathbb{Z}_q^*$, $\beta \leftarrow \mathbb{Z}_q^*$, challenge of type 1 is $g^{\frac{1}{\alpha\beta}}$, while challenge of type 2 is g^R for random $R \leftarrow \mathbb{Z}_q^*$.*

The assumption in Definition 23 is a new assumption which can be shown to imply Q-BDHI. We will assume that it holds for the prime order subgroup of composite order bilinear groups that can be efficiently instantiated [13].

Chapter 3

New Properties of the Groth-Sahai Proof System

In this section we will describe the basic properties and the main ideas in the construction of the Groth-Sahai proof system. In addition, we will define two new properties satisfied by this proof system. The first property generalizes a standard property called extractability, and allows for proofs which are only partially extractable. The second property, which we call randomizability, says that any user should be able to take an existing proof for a given statement and produce a new proof for the same statement. As we will see in the following chapters, these properties can be be used to solve several previously open problems.

3.1 An Overview of the Groth-Sahai Proof System

Groth and Sahai give an efficient commitment scheme Commit and an efficient witness indistinguishable proof system for statements of the following form (referred to as pairing product equations):

We are given bilinear parameters $(p, G_1, G_2, G_T, e, g, h)$, where G_1, G_2, G_T are groups of prime order p, with g a generator of G_1, h a generator of G_2, and $e : G_1 \times G_2 \to G_T$ a cryptographic bilinear map. The statement s to be proven consists of the following list of values: $\{a_q\}_{q=1\ldots Q} \in G_1$, $\{b_q\}_{q=1\ldots Q} \in G_2$, $t \in G_T$, and $\{\alpha_{q,m}\}_{m=1\ldots M, q=1\ldots Q}, \{\beta_{q,n}\}_{n=1\ldots N, q=1\ldots Q} \in Z_p$, as well as a list of commitments $\{c_m\}_{m=1\ldots M}$ to values in G_1 and $\{d_n\}_{n=1\ldots N}$ to values in G_2. Groth and Sahai show how to construct a proof that for all $m \in \{1 \ldots M\}$, C_m is a commitment to $x_m \in G_1$, and for all $n \in \{1 \ldots N\}$, D_n is a commitment to $y_n \in G_2$ such that:

$$\prod_{q=1}^{Q} e(a_q \prod_{m=1}^{M} x_m^{\alpha_{q,m}}, b_q \prod_{n=1}^{N} y_n^{\beta_{q,n}}) = t$$

3.1.1 Groth-Sahai Commitments [57]

We review the properties of the Groth-Sahai [57] commitment scheme. We use their scheme to commit to elements of a group G of prime order p. Technically, their constructions commit to elements of certain modules, but we can apply them to certain bilinear groups elements(see Section 3.2). [1]

$\mathsf{GSComSetup}(p, G, g)$**.** Outputs a common reference string $params_{Com}$.

$\mathsf{GSCommit}(params_{Com}, x, open)$**.** Takes as input $x \in G$ and some value $open$ and outputs a commitment $comm$. Similarly, we can commit to exponents using $\mathsf{GSExpCommit}(params_{Com}, b, \theta, open)$ which takes as input $\theta \in Z_p$ and a base $b \in G$ and outputs $(b, comm)$, where $comm = \mathsf{GSCommit}(params_{Com}, b^{\theta}, open)$.

$\mathsf{VerifyOpening}(params_{Com}, comm, x, open)$**.** Takes $x \in G$ and $open$ as input and outputs accept if $comm$ is a commitment to x. To verify that $(b, comm)$ is a commitment to exponent θ check $\mathsf{VerifyOpening}$ $(params_{Com}, comm, b^{\theta}, open)$.

For brevity, we write $\mathsf{GSCommit}(x)$ to indicate committing to $x \in G$ when the parameters are obvious and the value of $open$ is chosen appropriately at random. Similarly, $\mathsf{GSExpCommit}(b, \theta)$ indicates committing to θ using $b \in G$ as the base.

GS commitments are *perfectly binding*, *strongly computationally hiding*, and *extractable*. They are also multiplicatively homomorphic. Groth and Sahai [57] show how to instantiate commitments that meet these requirements using either the SXDH or DLIN assumptions. Commitments based on SXDH consist of 2 elements in G, while those based on DLIN setting require 3 elements in G. Note that while here we have described commitments for a single group G, in the Groth-Sahai proof system below, this will be used to commit to elements of each bilinear group. In an asymmetric setting, this means we will have two sets of parameters, one for committing to elements of G_1 and one for committing to elements of G_2. In the symmetric setting, we can have one set of parameters to commit to elements in $G_1 = G_2$.

3.1.2 Groth-Sahai Pairing Product Equation Proofs [57]

We formally define the Groth-Sahai proof system. Let $params_{BM} \leftarrow \mathsf{BilinearSetup}(1^k)$.

$\mathsf{GSSetup}(params_{BM})$**.** Calls $\mathsf{GSComSetup}$ to generate $params_1$ and $params_2$ for constructing commitments in G_1 and G_2 respectively, and optional auxiliary values $params_{\pi}$. Outputs $params_{GS} = (params_{BM}, params_1, params_2, params_{\pi})$.

$\mathsf{GSProve}(params_{GS}, s, (\{x_m\}_{1\ldots M}, \{y_n\}_{1\ldots N}, openings))$**.** Takes as input the parameters, the statement $s = \{(c_1, \ldots, c_M, d_1, \ldots, d_N), equations\}$ to be proven, (the statement s includes the commitments and the parameters of the pairing product equations), the witness consisting of the values $\{x_m\}_{1\ldots M}, \{y_n\}_{1\ldots N}$ and opening information $openings$. Outputs a proof π.

$\mathsf{GSVerify}(params_{GS}, \pi)$**.** Returns accept if π is valid, reject otherwise. (Note that it does not take the statement s as input because we have assumed that the statement is always included in the proof π.)

[1]Groth and Sahai also have a construction for composite order groups using the Subgroup Decision assumption; however it lacks the necessary extraction properties for our applications.

$\mathsf{GSExtractSetup}(params_{BM})$**.** Outputs $params_{GS}$ and auxiliary information (td_1, td_2). $params_{GS}$ are distributed identically to the output of $\mathsf{GSSetup}(params_{BM})$. (td_1, td_2) allow an extractor to discover the contents of all commitments formed using $\mathsf{GSCommit}$.

$\mathsf{GSExtract}(params_{GS}, td_1, td_2, \pi)$**.** Outputs $x_1, \ldots, x_M \in G_1$ and $y_1, \ldots, y_N \in G_2$ that satisfy the pairing product equations and that correspond to the commitments (note that the commitments and the equations are included with the proof π).

Groth-Sahai proofs satisfy *correctness*, *extractability*, and *strong witness indistinguishability*. We explain these requirements in a manner compatible with our notation.

Correctness. An honest verifier always accepts a proof generated by an honest prover.

Extractability. If an honest verifier outputs accept, then the statement is true. This means that, given td_1, td_2 corresponding to $params_{GS}$, $\mathsf{GSExtract}$ extracts values from the commitments that satisfy the pairing product equations with probability 1.

Strong Witness Indistinguishability. A simulator $\mathsf{Sim} = (\mathsf{SimSetup}, \mathsf{SimProve})$ with the following two properties exists: (1) $\mathsf{SimSetup}(params_{BM})$ outputs $params_{GS}'$ such that they are computationally indistinguishable from the output of $\mathsf{GSSetup}(params_{BM})$. Let $params_1' \in params_{GS}'$ be the parameters for the commitment scheme in G_1. Using $params_1'$, commitments are perfectly hiding – this means that for all commitments $comm$, $\forall x \in G_1, \exists open : \mathsf{VerifyOpening}(params_1', comm, x, open) = \mathsf{accept}$ (analogous for G_2). (2) Using the $params_{GS}'$ generated by the challenger, GS proofs become perfectly witness indistinguishable. Suppose an unbounded adversary $\mathcal{A}$ generates a statement s consisting of the pairing product equations and a set of commitments $(c_1, \ldots, c_M, d_1, \ldots, d_N)$. The adversary opens the commitments in two different ways $W_0 = (x_1^{(0)}, \ldots, x_M^{(0)}, y_1^{(0)}, \ldots, y_N^{(0)}, openings_0)$ and $W_1 = (x_1^{(1)}, \ldots, x_M^{(1)}, y_1^{(1)}, \ldots, y_N^{(1)}, openings_1)$ (under the requirement that these witnesses must both satisfy s). The values $openings_b$ show how to open the commitments to $\{x_m^{(b)}, y_n^{(b)}\}$. (The adversary can do this because it is unbounded.) The challenger gets the statement s and the two witnesses W_0 and W_1. He chooses a bit $b \leftarrow \{0,1\}$ and computes $\pi = \mathsf{GSProve}(params_{GS}', s, W_b)$. Strong witness indistinguishability means that π is distributed independently of b.

Composable Zero-Knowledge. Note that Groth and Sahai show that if in a given pairing product equation the constant t can be written as $t = e(t_1, t_2)$ for known t_1, t_2, then these proofs can be done in zero knowledge. However, their zero knowledge proof construction is significantly less efficient than the WI proofs. Thus, in most of what follows, we choose to use only the WI construction as a building block. Then we can take advantage of special features of our constructions to create much more efficient proofs that still have the desired zero knowledge properties. (We will however borrow many of the techniques that Groth and Sahai use in their zero-knowledge contruction.)

3.2 Details of the Groth-Sahai Proof System

Groth-Sahai give a construction of the proof system as described above based on modules with certain properties. By expressing the implementation of non-interactive proofs in the language of modules Groth and Sahai can remain general with respect to possible instantiations of their proof system. Modules that fulfill the necessary requirements for their proofs exist both under the SXDH, the DLIN assumption, and with some restrictions the Subgroup Hiding assumption.

Let $(R, +, \cdot, 0, 1)$ be a commutative ring. Formally, an R module is a commutative group $(M, \cdot, 1)$, such that $\forall r, s \in R : \forall u, v \in M : u^{r+s} = u^r u^s \wedge (uv)^r = u^r v^r$. In our case, we will want to build a module from some underlying bilinear group G. Then M will be a direct product of several copies of G (e.g. $M = G \times G$ for the instantiation based on SXDH). If we have a bilinear map $e : G_1 \times G_2 \rightarrow G_T$, we build corresponding modules from M_1, M_2, M_T, and then we can define a bilinear map $E : M_1 \times M_2 \rightarrow M_T$. (See [57] for more details.)

The general idea behind the Groth-Sahai proof construction commitment scheme will contain several module elements, $u_1 ... u_I \in M$. A commitment is formed by multiplying the module element that we are committing to by a random combination of the parameters u_i. Thus, if the values u_i generate the entire module, the commitment scheme will be perfectly hiding. On the other hand, if the parameters only generate some submodule U, then the resulting commitment will be binding within M/U. The idea is to choose the parameters in such a way that it is computationally difficult to determine whether or not they span the entire module.

Setup. The public parameters will include descriptions of two modules M_1, M_2 , and an efficient bilinear map E between them. The setup will also select some number I of elements $u_i \in M_1$, and some number J of elements $v_j \in M_2$. (Note that in the symmetric setting where $M_1 = M_2$, one set of elements is sufficient.) Finally, another set of values $\{\eta_h\}$ is also necessary in certain instantiations, although we will not discuss it here. Thus the parameters will include $M_1, M_2, E, \{u_i\}, \{v_j\}, \{\eta_k\}$.

Commitments. Commitments are realized using a Z_p module. For $u_1, \ldots, u_I$ elements of M, we call U the submodule of M generated by $u_1, \ldots, u_I$. To commit to $x \in G$, x is transformed into a unique element $x' \in M$ that for the perfectly binding setup is not element of U. (For the perfectly hiding setup we create the parameters such that $U = M$, so all module elements are in M.) Now we commit by choosing $r_1, \ldots, r_I \in Z_p$ at random and computing

$$comm = x' \prod_{i=1}^{I} u_i^{r_i}.$$

NIZK Proofs The NIZK proofs require bilinear maps over modules. Let M_1, M_2, M_T be R modules. Then we define the bilinear map $E : M_1 \times M_2 \rightarrow M_T$. Let U generated by $u_1, \ldots, u_I$ be a submodule of M_1 and V generated by $v_1 \ldots v_I$ a submodule of M_2. The commitments to x_i and y_i are defined over M_1 and M_2 respectively.

In order to prove that $c_1, \ldots c_Q, d_1, \ldots d_Q$ are commitments to $x_1, \ldots x_Q, y_1, \ldots y_Q$ respectively, such that $\prod_{q=1}^{Q} e(x_q, y_q) = t$, the prover computes values π_i and ψ_i that fulfill the following verification equation:

$$\prod_{q=1}^{Q} E(c_q, d_q) = t' \prod_{i=1}^{I} E(u_i, \pi_i) \prod_{j=1}^{J} E(\psi_j, v_j).$$

Where t' is a mapping of t to M_T. The values π_i and ψ_i can be computed from the x_i and y_i together with their commitments and opening information.

More specifically, the prover chooses random $r_{qi} \leftarrow Z_p$ for $1 \le i \le I, 1 \le q \le Q$, random $s_{qj} \leftarrow Z_p$ for $1 \le j \le J, 1 \le q \le Q$, random $t_{ij} \leftarrow Z_p$ for $1 \le i \le I, 1 \le j \le J$, and random $t_h \leftarrow Z_p$ for $1 \le h \le H$, and computes $\pi_i = \prod_{j=1}^{J} v_j^{t_{ij}} \prod_{q=1}^{Q} d_q^{r_{qi}}$ and $\psi_j = \prod_{i=1}^{I} u_i^{\sum_{h=1}^{H} t_h \eta_{h_{ij}}} \prod_{i=1}^{I} u_i^{-t_{ij}} \prod_{q=1}^{Q} x_q^{s_{qj}}$.

This can be extended in the logical way to the more complex formulation, which includes constants $a_m, b_n, \alpha_{q,m}, \beta_{q,n}$.

Groth and Sahai show that when the values u_i and v_j span M_1 and M_2 respectively, the proofs generated by different witnesses will be identical (as long as the witnesses are valid). Recall that under these parameters the commitment scheme is perfectly hiding. Suppose there are two sets of values $x_1, \ldots, x_m, y_1, \ldots, y_n$ and $x'_1, \ldots, x'_m, y'_1, \ldots, y'_n$ that each satisfy a given pairing product equation. Since the commitment scheme is perfectly hiding, given commitments $c_1, \ldots c_m, d_1, \ldots d_n$, we know there there exists openings $open$ and $open'$, which allow these commitments to be opened to either set of values (although it will be hard for the adversary to find both of these openings). Generating a proof using $x_1, \ldots x_m, y_1, \ldots y_n, open$ or $x'_1, \ldots x'_m, y'_1, \ldots y'_n, open'$ should produce the same distribution.

At the same time, they show that if the values u_i generate a submodule U of M, then the proof is sound within M/U.

3.3 Instantiating the Groth-Sahai Proofs System

In all of our constructions, we choose bilinear groups G_1, G_2, G_T with bilinear map e, and then use the Groth-Sahai commitments to commit to elements $x \in G_1$ or $x \in G_2$. However, most of [57] focuses on commitments and proofs for elements of modules. Here we describe the techniques suggested by Groth and Sahai for using these commitments to commit to group elements. Using group elements instead of modules also allows us to get the extraction properties necessary for our constructions. We describe commitment to group elements in the SXDH and DLIN settings:

SXDH. In the SXDH setting, one commits to elements in G_1 as follows (committing to elements in G_2 is similar):

The parameters are generated by choosing random s, z and computing $u_1 = (g, g^z)$ and $u_2 = (g^s, g^{sz})$. The public parameters are u_1, u_2. If extraction is necessary, the trapdoor will be s, z.

Groth-Sahai describe commitments to elements in the module $M = G \times G$ as follows: To commit to element $X = (x_1, x_2) \in M$ choose random $r_1, r_2 \in Z_p$, and compute $X u_1^{r_1} u_2^{r_2}$ (where multiplication is entry-wise). One can commit to $x \in G$ by choosing random $r_1, r_2 \in Z_p$ and computing $(1, x) u_1^{r_1} u_2^{r_2}$.

Opening would reveal x, r_1, r_2. In this case, given the trapdoor s, z, we will be able to extract x from a commitment (c_1, c_2) by computing c_2/c_1^z. Thus, this is perfectly binding and extractable.

Note that because all operations in the module M are entry-wise, any relationship that holds over elements $(1, x), (1, y) \in M$ will also hold over group element $x, y \in G$ [2]. Groth-Sahai proofs demonstrate that the proved relationship holds within M/U over any possible opening for the given commitments. Thus, it must hold for the unique $(1, x), (1, y)$ which are produced by the extraction algorithm described above, and as mentioned, this means the proved relationships must hold over group elements x, y.

Simulated parameters are generated by choosing random $s, z, w \in Z_p$ and computing $u_1 = (g, g^z)$ and $u_2 = (g^s, g^w)$. The public parameters will be u_1, u_2. The simulation trapdoor will be s, z, w. Note that these public parameters will be indistinguishable from those described above by SXDH. Note that when these simulated parameters are used, the resulting commitment scheme is perfectly hiding. Further, we can form commitments which are identical to those described above but for which we can use the simulation trapdoor to open to any value for which we know the discrete logarithm. We compute such a commitment by choosing random $c_1, c_2 \in Z_p$ and computing (g^{c_1}, g^{c_2}). To open this commitment to any value g^ϕ, we need only find a solution (r_1, r_2) to the equations $c_1 = r_1 + sr_2$ and $c_2 = \phi + zr_1 + wr_2$.

DLIN. In the DLIN setting one commits to to elements in G_1 as follows (committing to elements in G_2 is similar):

The parameters are generated by choosing random a, b, z, s and computing $u_1 = (g^a, 1, g)$ and $u_2 = (1, g^b, g)$, and $u_3 = (g^{az}, g^{bs}, g^{z+s})$. The public parameters are u_1, u_2, u_3. If extraction is necessary, the trapdoor will be a, b, z, s.

Groth-Sahai describe commitments to elements in the module $M = G \times G \times G$ as follows: To commit to element $X = (x_1, x_2, x_3) \in M$ choose random $r_1, r_2, r_3 \in Z_p$, and compute $X u_1^{r_1} u_2^{r_2} u_3^{r_3}$ (where multiplication is entry-wise). One can commit to $x \in G$ by choosing random $r_1, r_2, r_3 \in Z_p$ and computing $(1, 1, x) u_1^{r_1} u_2^{r_2} u_3^{r_3}$. Opening would reveal x, r_1, r_2, r_3. In this case, given the trapdoor a, b, s, z, we will be able to extract x from a commitment (c_1, c_2, c_3) by computing $c_3/(c_1^{1/a} c_2^{1/b})$. Note that again any relationship that holds over elements $(1, 1, x), (1, 1, y) \in M$ will also hold over group element $x, y \in G$. Thus, we can using Groth-Sahai proofs on commitments to x, y to prove statements about x, y.

Simulated parameters are generated by choosing random $a, b, s, z, w \in Z_p$ and computing $u_1 = (g^a, 1, g)$ and $u_2 = (g^b, 1, g)$ and $u_3 = (g^{az}, g^{bs}, g^w)$. The public parameters will be u_1, u_2, and u_3. The simulation trapdoor will be a, b, s, z. Note that these public parameters will be indistinguishable from those described above by DLIN. Note that when these parameters are used, the resulting commitment scheme is perfectly hiding. Further, we can form commitments which are identical to those described above but for which we can use the simulation trapdoor to open to any value for which we know the discrete logarithm. We compute such a commitment by choosing random $c_1, c_2, c_3 \in Z_p$ and computing $(g^{c_1}, g^{c_2}, g^{c_3})$. To open this commitment to any value g^ϕ, we need only find a solution (r_1, r_2, r_3) to the equations $c_1 = ar_1 + azr_3$, $c_2 = br_2 + bsr_3$ and $c_3 = \phi + r_1 + r_2 + (z + s)r_3$.

[2] The bilinear map E over M is not entry-wise, but does still imply that any relationship over $E((1, x), (1, y))$ also holds over $e(x, y)$.

3.4 Efficiency

We first consider the asymmetric setting, where $G_1 \neq G_2$. Then we will consider the symmetric setting, which allows for some efficiency improvements.

The asymmetric setting. Let L_1 be the number of elements in G_1 used to represent an element of M_1, and let L_2 be the number of elements in G_2 used to represent an element of M_2. Let k be the security parameter.

In the asymmetric setting, our parameters consist of I elements of M_1 and J elements of M_2, which means a total of IL_1 elements of G_1 and JL_2 elements of G_2. Group elements can generally be represented in $O(k)$ bits[3], so the resulting parameters are $O((IL_1 + JL_2)k)$ bits, where I, J, L_1, L_2 are small constants which depend on the assumption used.

A commitment to module element $X \in G_1$ with randomness $r_1, \dots r_I$ requires computing $X \prod_{i=1}^{I} u_i^{r_i}$. Thus, the total computation required is I module exponentiations[4], which in turn means IL_1 exponentiations in G_1. (Similarly committing to an element of G_2 will require JL_2 exponentiations in G_2). This is roughly equivalent to the computation required to compute the same number of modular exponentiations on k-bit integers [5]. The resulting computation should require $O(IL_1k)$ or $O(JL_2k)$ k-bit modular multiplications, where I, J, L_1, L_2 are small constants which depend on the assumption used. The resulting commitment will consist of one module element, which means L_1 elements of G_1 or L_2 elements of G_2.

To prove that given set of commitments contains values $x_1, \dots x_q, y_1, \dots y_q$ that satisfy a pairing product equation $\prod_{q=1}^{Q} e(x_q, y_q) = t$, we must compute:

$$\pi_i = \prod_{j=1}^{J} v_j^{t_{ij}} \prod_{q=1}^{Q} d_q^{r_{qi}} \text{ and } \psi_j = \prod_{i=1}^{I} u_i^{\sum_{h=1}^{H} t_h \eta_{hij}} \prod_{i=1}^{I} u_i^{-t_{ij}} \prod_{q=1}^{Q} x_q^{s_{qj}},$$

where all the values $t_{ij}, r_{qi}, t_h, s_{qj}$ are values in Z_p. Computing each value π_i requires $J + Q$ module exponentiations, and computing each value ψ_j requires $I + Q$ module exponentiations. We need to compute I π_i's and J ψ_j's, so since all operations are componentwise, this comes to a total of $I(J + Q)L_2$ exponentiations in G_2 and $J(I + Q)L_1$ exponentiations in G_1. The resulting computation should require $O((IJ(L_1 + L_2) + (IL_2 + JL_1)Q)k)$ k-bit modular multiplications, where I, J, L_1, L_2 are small constants which depend on the assumption used.

The resulting proof will contain the I elements $\pi_i \in M_2$ and the J elements $\pi_j \in M_1$, so the total size will be IL_1 elements in G_2 and JL_2 elements in G_1, or roughly $O((IL_1 + JL_2)k)$ bits.

Verifying the proof requires computing $Q+I+J$ module pairings, which in the Groth-Sahai instantiations requires $O((Q + I + J)L_1L_2)$ group pairings.

To summarize, we get the following general formulas:

Theorem 1. *In the asymmetric setting, the Groth-Sahai witness-indistinguishable proof system will have the following efficiency: Let L_1 be the number of elements in G_1 used to represent an element of M_1, and let L_2 be the number of elements in G_2 used to represent an element of M_2.*

[3]This is a rough estimate. There are many different constructions for bilinear groups. The exact figure will depend on the choice of instantiation for the bilinear group, and the security parameter required for the assumption being used.

[4]The computation also requires an equal number of multiplications, but as this will clearly be dominated by the exponentiations, we will ignore it here. We will similarly omit mentioning operations which are clearly insignificant from our other computations.

[5]This is again a rough estimate as the cost of multiplication depends on the instantiation of the underlying group.

Parameters *The parameters include a description of the groups and IL_1 elements of G_1 and JL_2 elements of G_2.*

Commitments *Forming a commitment to an element of G_1 will require IL_1 exponentiations in G_1 and the resulting commitment will be represented by L_1 elements of G_1. Similarly, committing to an element of G_2 will require JL_2 elements of G_2, and result in L_2 elements of G_2.*

Proofs *For a pairing equation with a product of Q pairings: Generating the proof will require $J(I+Q)L_1$ exponentiations in G_1 and $I(J+Q)L_2$ exponentiations in G_2. The resulting proof will consist of JL_1 elements of G_1 and IL_2 elements of G_2. Verifying the proof will involve computing $(Q+I+J)L_1L_2$ bilinear group pairings.*

When instantiated with the SXDH assumption, we get $I = J = L_1 = L_2 = 2$. Thus, the following theorem follows:

Theorem 2. *The SXDH instantiation given in Section 3.3 will have the following efficiency:*

Parameters *The parameters include a description of the groups and 4 elements of G_1 and 4 elements of G_2.*

Commitments *Forming a commitment will require 4 exponentiations in either G_1 or G_2, and the resulting commitment will be represented by 2 elements of the appropriate group.*

Proofs *For a pairing equation with a product of Q pairings: Generating the proof will require $8 + 4Q$ exponentiations in each of G_1 and G_2. The resulting proof will consist of 4 elements of each group. Verifying the proof will involve computing $4Q + 16$ bilinear group pairings.*

When instantiated with the DLIN assumption, we get $I = J = L_1 = L_2 = 3$. Thus, the following theorem follows:

Theorem 3. *In the asymmetric setting, the DLIN instantiation given in Section 3.3 will have the following efficiency:*

Parameters *The parameters include a description of the groups and 9 elements of G_1 and 9 elements of G_2.*

Commitments *Forming a commitment will require 9 exponentiations in either G_1 or G_2, and the resulting commitment will be represented by 3 elements of the appropriate group.*

Proofs *For a pairing equation with a product of Q pairings: Generating the proof will require $27 + 9Q$ exponentiations in each of G_1 and G_2. The resulting proof will consist of 9 elements of each group. Verifying the proof will involve computing $9Q + 54$ bilinear group pairings.*

The symmetric setting. The DLIN assumption has also been proposed in the symmetric setting, where $G_1 = G_2 = G$ and the corresponding bilinear map is $e : G \times G \to G_T$. This allows for somewhat more efficient constructions. Here we will examine the efficiency in the symmetric setting in general, and then consider the specific instantiation based the DLIN assumption.

Let L be the number of elements in G used to represent an element of M. Let k be the security parameter.

In the symmetric setting, we only need one set of parameters consisting of I elements of M, which means a total of IL elements of G. Group elements can generally be represented in $O(k)$ bits[6], so the resulting parameters are $O((IL)k)$ bits, where I, L are small constants which depend on the assumption used.

A commitment to module element $X \in G$ is computed as in the asymmetric case. Thus, the total computation required is IL exponentiations in G. This is roughly equivalent to the computation required to compute the same number of modular exponentiations on k-bit integers [7]. The resulting computation should require $O(ILk)$ k-bit modular multiplications, where I, L are small constants which depend on the assumption used. The resulting commitment will consist of one module element, which means L elements of G.

The asymmetric setting does simplify the resulting proofs, since we now have only one set of parameters. To prove that given set of commitments contains values $x_1, \dots x_q, y_1, \dots y_q$ that satisfy a pairing product equation $\prod_{q=1}^{Q} e(x_q, y_q) = t$, we must compute $\pi_i = \prod_{q=1}^{Q} d_q^{r_{qi}} \prod_{j=1}^{I} u_j^{\sum_{h=1}^{H} t_h \eta_{h_{ji}}} \prod_{q=1}^{Q} x_q^{s_{qi}}$, where all the values $t_{ij}, r_{qi}, t_h, s_{qj}$ are values in Z_p. Computing each value π_i requires $I+2Q$ module exponentiations. We need to compute I π_i's, so since all operations are componentwise, this comes to a total of $I(I+2Q)L$ exponentiations in G. The resulting computation should require $O((I^2L + ILQ)k)$ k-bit modular multiplications, where I, L are small constants which depend on the assumption used.

The resulting proof will contain the I elements $\pi_i \in M$, so the total size will be IL elements in G, or roughly $O(ILk)$ bits.

Verifying the proof requires computing $Q + I$ module pairings, which in the Groth-Sahai instantiations requires $O((Q + I)L^2)$ group pairings.

To summarize, we get the following general formulas:

Theorem 4. *In the symmetric setting, the Groth-Sahai witness-indistinguishable proof system will have the following efficiency: Let L be the number of elements in G used to represent an element of M.*

Parameters *The parameters include a description of the groups and IL elements of G_1.*

Commitments *Forming a commitment to an element of G will require IL exponentiations in G and the resulting commitment will be represented by L elements of G.*

Proofs *For a pairing equation with a product of Q pairings: Generating the proof will require $I(I+2Q)L$ exponentiations in G. The resulting proof will consist of IL elements of G. Verifying the proof will involve computing $(Q+I)L^2$ bilinear group pairings.*

When instantiated with the DLIN assumption, we get $I = L = 3$. Thus, the following theorem follows:

Theorem 5. *In the symmetric setting, the DLIN instantiation given in Section 3.3 will have the following efficiency:*

Parameters *The parameters include a description of the groups and* 9 *elements of G.*

[6]See footnote 3.

[7]See footnote 5.

Commitments *Forming a commitment will require* 9 *exponentiations in* G*, and the resulting commitment will be represented by* 3 *elements of* G*.*

Proofs *For a pairing equation with a product of* Q *pairings: Generating the proof will require* $27 + 18Q$ *exponentiations in* G*. The resulting proof will consist of* 9 *elements in* G*. Verifying the proof will involve computing* $9Q + 27$ *bilinear group pairings.*

3.5 Zero-Knowledge Proof of Equality of Committed Values

As mentioned above, most of our constructions rely only on the Groth-Sahai witness indistinguishable proofs. However, we will make heavy use of one particular zero-knowledge proof. Here we are given two commitments, and construct a composable zero-knowledge proof that both commitments can be opened to the same value. Such a proof can be constructed using Groth-Sahai techniques as follows:

Suppose we know $c_1 = \mathsf{GSCommit}(params_1, a)$ and $c_2 = \mathsf{GSCommit}(params_1, a)$ as well as the opening information to c_1 and c_2.[8] We want to prove that c_1 is a commitment to A and c_2 is a commitment to B such that $A = B$.

To do this, we calculate $d = \mathsf{GSCommit}(params_2, h)$. Then we construct two witness indistinguishable proofs π_1, π_2. The first shows that c_1 commits to A and c_2 commits to B and d commits to Y such that $e(A/B, Y) = 1$. The second shows that d commits to Y such that $e(g, Y) = e(g, h)$. The final proof is π_1, π_2, d.

Extractability. We use $\mathsf{GSExtractSetup}(params)$ to generate $params_{GS}$ and a trapdoor td that lets us open all commitments. Suppose an adversary gives us a proof π. We extract a, b, and y from c_1, c_2, d. By the soundness of the GS proof system, we have that $e(g, y) = e(g, h)$, so $y = h$. We can now transform the clause $e(a/b, y) = 1$ to $e(a/b, h) = 1$. Since e is non-degenerate, this means $a/b = 1$, and thus $a = b$.

Composable Zero Knowledge. Recall that the definition of composable zero knowledge requires that there be an alternate setup algorithm SimSetup which produces simulated parameters (which are indistinguishable from the standard parameters) and a corresponding trapdoor, and an algorithm SimProve which uses this trapdoor to generate proofs without knowing the corresponding witnesses. The resulting proofs should be distributed identically to those generated by an honest prover using the simulated parameters. Thus, we need to show how to construct $\mathsf{Sim} = (\mathsf{SimSetup}, \mathsf{SimProve})$.

We will use the GSSimSetup algorithm provided by Groth and Sahai. Using the corresponding trapdoor, we can generate a commitment $\mathsf{GSExpCommit}(params_i, h, \theta, open)$ which we can open to any other value h^γ as long as we know the appropriate discrete logarithm γ. (Similarly, for commitments in G_1, we can commit to g^θ and open it to any other g^γ as long as both θ and γ are known.)

Furthermore, because of the perfect witness indistinguishability property that holds for the Groth-Sahai proofs under these simulated parameters, we can generate a proof for one equation using one opening for

[8]Proofs for G_2 are done analogously.

a given commitment, and a proof for the second equation using a different opening, and the result will be identical to a proof which uses the same value in both equations.

Thus we get the following algorithm $\mathsf{SimProve}$: The simulator gets as input c_1 and c_2. All the simulator needs to do is construct a witness for the individual equations of the proof. It computes $d = \mathsf{GSExpCommit}$ $(params_2, h, 0, open)$. Thus, we satisfy the pairing product equation $e(A/B, Y) = 1$ because $Y = h^0 = 1$, so we can use these values as a witness to form π_1. To satisfy the second pairing product equation, we open d to $Y = h^1 = h$. Thus, we satisfy $e(g, Y) = e(g, h)$, and we can use this value as a witness to form π_2. [9] As described above, the perfect witness indistinguishability under these parameters means that these proofs will be distributed identically to those generated by an honest prover using the same parameters. Thus we get composable zero-knowledge.

Efficiency The resulting proof requires one additional commitment and GS proofs for two pairing product equations, each with $Q = 1$. Thus, we get the following efficiency results:

Lemma 1. *When instantiated using the SXDH instantiation given in Section 3.3 the above proof system will have the following efficiency: To prove that two commitments commit to the same value in G_1 (G_2 is analogous): Generating the proof will require* 24 *exponentiations in G_1 and* 28 *exponentiations in G_2. The resulting proof will consist of* 8 *elements of G_1 and* 10 *elements of G_2. Verifying the proof will involve computing* 40 *bilinear group pairings.*

When instantiated using the symmetric setting version of the DLIN instantiation given in Section 3.3, the above proof system will have the following efficiency: To prove that two commitments commit to the same value: Generating the proof will require 99 *exponentiations in G. The resulting proof will consist of* 21 *elements in G. Verifying the proof will involve computing* 72 *bilinear group pairings.*

3.6 F-Extraction

Recall that in a traditional proof of knowledge system, there exists an extractor, which can use some trapdoor information about the parameters to extract an entire valid witness from an accepted proof.

The extractability property of Groth-Sahai commitments and proofs gives us something very similar in that, given an extraction trapdoor, one can extract values $x_1, \ldots x_m$ and $y_1 \ldots y_n$ from the commitments involved. However, note that here we are only able to extract part of the witness for the statement being proved: the true witness would also include the opening information for each of the commitments. Furthermore, if we consider proofs about commitments to elements of Z_p formed using $\mathsf{GSExpCommit}$, we get evn less extractability: given a commitment $\mathsf{GSExpCommit}(params_{GS}, h, \alpha)$ to $\alpha \in Z_p$, we can only extract the group element g^{α}.

Thus, we introduce the notion of f-extractability to capture this notion of partial extractability. We will define it more formally, and then give a convenient notation with which to describe such proofs.

[9]Note that there is an additional subtlety here in that the simulator doesn't know any opening for the commitments c_1, c_2. However, c_1, c_2 can be seen as commitments with randomness 0 to the module elements c_1, c_2. Thus, the values $0, c_1, c_2$ can be used as a witness. Witness indistinguishability means that the resulting proof will be identical to one generated using an opening to a valid set of group elements.

3.6.1 Definition

We first apply a standard generalization to extend the notion of NIPK for a language L to languages parameterized by $params$ – we allow the Turing machine M_L to receive $params$ as a separate input. Next, we generalize extractability to f-extractability. We say that a NIPK system is f-extractable if Extract outputs y, such that there $\exists x : M_L(params, s, x) = \mathsf{accept} \wedge y = f(params, x)$. If $f(params, \cdot)$ is the identity function, we get the usual notion of extractability. (Note, here we have changed our notation from that given in Section 2.1.3, so that s now represents the statement or instance, and x is the witness.)

More formally:

Definition 24 (Extractability). *A non-interactive proof system* $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{VerifyProof})$ *is a proof of knowledge for language L parameterized by* $params$ *if there exists a polynomial-time extractor* $(\mathsf{ExtractSetup}, \mathsf{Extract})$ *such that the following hold: (1)*

$$\{params \leftarrow \mathsf{Setup}(1^k) : params\} \approx \{(params, t) \leftarrow \mathsf{ExtractSetup} : params\}$$

and (2) For all PPT adversaries $\mathcal{A}$ there exists negligible ν such that,

$$\begin{aligned}
&\Pr[(params, t) \leftarrow \mathsf{ExtractSetup}; (s, \pi) \leftarrow \mathcal{A}(params); y \leftarrow \mathsf{Extract}(params, t, s, \pi) \\
&\quad : \mathsf{VerifyProof}(params, x, \pi, M_L) = 0 \vee (\exists x s.t. y = f(params, x) \wedge M_L(params, s, x) = 1)] \\
&\quad = 1 - \nu(k)
\end{aligned}$$

where M_L is the TM which accepts $params, s, x$ iff x is a witness for the statement $s \in L(params)$.

3.6.2 Notation

We denote an f-extractable proof π obtained by running $\mathsf{Prove}(params, s, x)$ as

$$\pi \leftarrow \mathsf{NIPK}\{params, s, f(params, x) : M_L(params, s, x) = \mathsf{accept}\}.$$

We omit the $params$ where they are obvious. In our applications, s is a conditional statement about the witness x, so $M_L(s, x) = \mathsf{accept}$ if $\mathsf{Condition}(x) = \mathsf{accept}$. Thus the statement $\pi \leftarrow \mathsf{NIPK}\{f(x) : \mathsf{Condition}(x)\}$ is well defined. Suppose s includes a list of commitments $c_n = \mathsf{Commit}(x_n, open_n)$. The witness is $x = (x_1, \ldots, x_N, open_1, \ldots, open_N)$, however, we typically can only extract $x_1, \ldots, x_N$. We write

$$\begin{aligned}
\pi \leftarrow \mathsf{NIPK}\{(x_1, \ldots, x_n) :&\mathsf{Condition}(x) \\
&\wedge \forall \ell\ \exists open_\ell : c_\ell = \mathsf{Commit}(params_{Com}, x_\ell, open_\ell)\}.
\end{aligned}$$

We introduce shorthand notation for the above expression: $\pi \leftarrow \mathsf{NIPK}\{((c_1 : x_1), \ldots, (c_n : x_n)) : \mathsf{Condition}(x)\}$. Sometimes we use the alternate notation: $\pi \leftarrow \mathsf{NIPK}[x_1 \,\mathrm{in}\, c_1, \ldots, x_n, \mathrm{in}\, c_n]\{(x_1, \ldots, x_n) : \mathsf{Condition}(x)\}$. We use the notation $\pi \in \mathsf{NIPK}[x_1 \,\mathrm{in}\, c_1, \ldots, x_n, \mathrm{in}\, c_n]\{(x_1, \ldots, x_n) : \mathsf{Condition}(x)\}$. to say that π is a proof that will be accepted by the corresponding verification procedure.

For simplicity, we assume the proof π includes s.

3.7 Randomizability

The basic intuition behind the randomizability property is that we would like to have an efficient algorithm which will take any non-interactive proof generated by our proof system and "rerandomize" it to produce something that is distributed indistinguishably from a completely independent freshly generated proof of the same statement.

Here we first formalize the notions of concatenated proofs and projected proofs. Then we formally describe the randomizability property. Finally, we will show that the Groth-Sahai proofs do satisfy this property by giving an appropriate randomization algorithm and proving that it acts as desired.

3.7.1 Concatenated and Projected Proofs

Intuitively, we want a randomizable proof system to allow for concatenated and projected proofs:

Concatenated proofs. Informally, given two proofs π and π' that each prove a statement about a set of commitments, their concatenation $\pi \circ \pi'$ is a proof of the AND of the two statements.

Projected proofs. Informally, if a proof π proves a statement about the contents of a set of commitments, then it also implies a statement about the contents of the subset of the commitments.

Here, we give a formal definition of the concatenation and projection operations on proofs.

Definition 25 (Concatenated proof). *Let $(C_1, \ldots, C_n)$ and $(C'_1, \ldots, C'_m)$ be two sets of commitments, and suppose that ℓ is such that $C_i = C'_i$ for $1 \le i \le \ell$. Let $\pi \in \mathsf{NIPK}[x_1 \,\mathrm{in}\, C_1, \ldots, x_n \,\mathrm{in}\, C_n]\{f(x_1, \ldots, x_n, y) : \mathsf{Condition}(x_1, \ldots, x_n, y)\}$ and $\pi' \in \mathsf{NIPK}[x'_1 \,\mathrm{in}\, C'_1, \ldots, x'_m \,\mathrm{in}\, C'_m]\{f(x'_1, \ldots, x'_n, y') : \mathsf{Condition}'(x'_1, \ldots, x'_n, y')\}$. Then the concatenated proof $\pi \circ \pi'$ is the proof of the AND of the two conditions; more precisely:*

$$\begin{aligned} \pi \circ \pi' \quad \in \quad & \mathsf{NIPK}[x_1 \,\mathrm{in}\, C_1, \ldots, x_\ell \,\mathrm{in}\, C_\ell, x_{\ell+1} \,\mathrm{in}\, C_{\ell+1}, \ldots, x'_{\ell+1} \,\mathrm{in}\, C'_{\ell+1}, \ldots] \\ & \{f(x_1, \ldots, x_n, y), f'(x_1, \ldots, x_\ell, x'_{\ell+1}, \ldots, x_m) \\ & : \mathsf{Condition}(x_1, \ldots, x_n, y) \wedge \mathsf{Condition}'(x_1, \ldots, x_\ell, x'_{\ell+1}, \ldots, x'_m)\} \end{aligned}$$

Note that the concatenated proof is f'' extractable, where f'' outputs the concatenation of the output of f and f'. It also remains zero-knowledge since a simulator for the concatenated proof system will simply simulate the constituent proofs and then concatenate them.

Definition 26 (Projected proof). *Let $\pi \in \mathsf{NIPK}[x_1 \,\mathrm{in}\, C_1, \ldots, x_n \,\mathrm{in}\, C_n]\{f(x_1, \ldots, x_n, y) : \mathsf{Condition}(x_1, \ldots, x_n, y)\}$. Suppose $\ell < n$. Let π' be obtained from π by replacing the description of f with that of f', where $f'(x_1, \ldots, x_\ell, (x_\ell + 1, \ldots, x_n, y)) = f(x_1, \ldots, x_n, y)$, and the description of $\mathsf{Condition}$ with that of $\mathsf{Condition}'$ where, similarly, $\mathsf{Condition}'(x_1, \ldots, x_\ell, (x_\ell + 1, \ldots, x_n, y)) = \mathsf{Condition}(x_1, \ldots, x_n, y)$. Then*

$$\pi' \in \mathsf{NIPK}[x_1 \,\mathrm{in}\, C_1, \ldots, x_n \,\mathrm{in}\, C_\ell]\{f'(x_1, \ldots, x_\ell, y) : \mathsf{Condition}'(x_1, \ldots, x_\ell, y)\}.$$

It is easy to see that the resulting proof system is f'-extractable and zero-knowledge.

3.7.2 Randomized NIZK Proofs of Knowledge

Let $\mathsf{Commit}(params_{PK}, \cdot, \cdot) : X \times Y \mapsto \{0,1\}^*$ be a perfectly binding non-interactive commitment scheme. X denotes the domain of the inputs to the commitment, and Y denotes the domain that the random value *open* comes from (both may depend on $params_{PK}$). To commit to x, one chooses *open* at random from Y and computes $C = \mathsf{Commit}(params_{PK}, x, open)$. We say that this commitment scheme is *randomizable* if Y is an efficiently samplable group with efficiently computable '0' element and '+' and '-' operations, and there is an efficient algorithm RandComm that, on input $params_{PK}$, $C = \mathsf{Commit}(params_{PK}, x, open)$ and $open' \in Y$, outputs $C' = \mathsf{Commit}(params_{PK}, x, open + open')$.

Let PKSetup, PKProve, PKVerify constitute a composable proof system as defined above. Then there exists SimSetup algorithm that outputs $(params_{PK}, sim)$ where $params_{PK}$ is indistinguishable from the output of PKSetup. Following Groth and Sahai, we require that, when parameterized by $params_{PK}$ chosen by SimSetup, Commit is perfectly hiding. Let RandProof be an efficient algorithm that on input $(C_1, \ldots, C_n)$, $(open'_1, \ldots, open'_n)$ and π outputs a proof π'.

Consider the following two experiments. In both experiments we are given $params_{PK}$ chosen by $\mathsf{SimSetup}(1^k)$, and also y, $(x_1, \ldots, x_n)$, $(open_1, \ldots, open_n)$, $(open'_1, \ldots, open'_n)$, Condition, and a proof π such that (a) $\mathsf{Condition}(x_1, \ldots, x_n, y)$ accepts; (b) $\pi \in \mathsf{NIPK}[x'_1 \,\mathrm{in}\, C_1, \ldots, x'_n \,\mathrm{in}\, C_n]\{f(x'_1, \ldots, x'_n, y') : \mathsf{Condition}(x'_1, \ldots, x'_n, y')\}$. Let $(C_1, \ldots, C_n)$ and $(C'_1, \ldots, C'_n)$ be defined as follows: $C_i = \mathsf{Commit}(x_i, open_i)$, $C'_i = \mathsf{Commit}(x_i, open_i + open'_i)$.

In the first experiment, π'_1 is computed by PKProve on input the commitments $\{C'_i\}$, their openings $open_i + open'_i$, and the values $(x_1, \ldots, x_n, y)$. In the second experiment, π'_2 is the output of RandProof $(params_{PK}, (C_1, \ldots, C_n), (open'_1, \ldots, open'_n), \pi)$.We say that RandProof randomizes the proof system if with all but negligible probability over the choice of $params_{PK}$, for all π, $(x_1, \ldots, x_n)$, y, $(open_1, \ldots, open_n)$, $(open'_1, \ldots, open'_n)$, Condition satisfying (a) and (b) above, π'_1 is distributed identically to π'_2 even given $(x_1, \ldots, x_n)$, $(open_1, \ldots, open_n)$, $(open'_1, \ldots, open'_n)$, the proof π, and the value y.

Finally, we say that RandProof is a correct randomization algorithm if given parameters $params_{PK}$ chosen by the real PKSetup, whenever a proof π passes the verification algorithm, then it's rerandomization will pass the verification algorithm. More formally, for all commitments $C_1, \ldots, C_n$, for all $(open'_1, \ldots, open'_n)$, for all proofs $\pi \in \mathsf{NIPK}[x'_1 \,\mathrm{in}\, C_1, \ldots, x'_n \,\mathrm{in}\, C_n]\{f(x'_1, \ldots, x'_n, y') : \mathsf{Condition}(x'_1, \ldots, x'_n, y')\}$, we are guaranteed that $\mathsf{RandProof}(params_{PK}, (C_1, \ldots, C_n), (open'_1, \ldots, open'_n), \pi)$ produces π' such that $\pi' \in \mathsf{NIPK}[x'_1 \,\mathrm{in}\, \mathsf{RandComm}(C_1, open'_1), \ldots, x'_n \,\mathrm{in}\, \mathsf{RandComm}(C_n, open'_n)]\{f(x'_1, \ldots, x'_n, y') : \mathsf{Condition}$ $(x'_1, \ldots, x'_n, y')\}$.

3.7.3 Groth-Sahai Proofs Are Randomizable

We now show how to randomize Groth-Sahai proofs.

Randomizing commitments. A GS commitment for G_1 is a function $G_1 \times R^I \mapsto M_1$. The domain of opening $Y = R^I$ is an additive group under element wise addition in R. On input $open' = (s_1, \ldots, s_I) \in R^J$ and a commitment $C = \mu_1(x) \cdot \prod_{i=1}^{I} u_i^{r_i}$ we compute $C' = C \cdot \prod_{i=1}^{I} u_i^{s_i} = \mu_1(x) \cdot \prod_{i=1}^{I} u_i^{r_i + s_i}$. Similar properties hold for commitments to elements in G_2 except that we use $open' = (z_1, \ldots, z_J) \in R^J$. For simplicity $\mathsf{RandComm}(params_{GS}, m, open')$ determines the group of m and follows the instructions for the respective group.

Randomizing Proofs. $\mathsf{RandProof}$ gets as input commitments $(\hat{comm}_1, \ldots, \hat{comm}_{\hat{n}})$, $(open'_1, \ldots, open'_{\hat{n}})$, and the proof
$[(\pi_1, \ldots, \pi_I, \psi_1, \ldots, \psi_J), \Pi]$. Π contains the internal commitments $C_1, \ldots, C_M$ and $D_1, \ldots, D_N$, and the pairing product equation.[10]

First it appropriately rerandomizes the commitments C_m and D_n to C'_m and D'_n, and stores the $s_{m,i}$ and $z_{n,j}$ it used: It matches the $\hat{comm}_{\hat{m}}$ to the C_m and D_n using our statement. If a C_m has a matching $\hat{comm}_{\hat{m}}$ it sets $(s_{m,1}, \ldots, s_{m,I}) = open'_{\hat{m}}$; otherwise it assigns random $(s_{m,1}, \ldots, s_{m,I}) \leftarrow R^I$. Similarly, it sets $(z_{n,1}, \ldots, z_{n,J}) = open_{\hat{n}}$ if there is a match, or assigns random $(z_{n,1}, \ldots, z_{n,J}) \leftarrow R^J$ otherwise. Commitment $C'_m = \mathsf{RandComm}(params_{GS}, C_m, (s_{m,1}, \ldots, s_{m,I}))$ and $C'_m = \mathsf{RandComm}(params_{GS}, C_m, (z_{m,1}, \ldots, z_{m,J}))$.

Then it computes $\hat{s}_{q,i} = \sum_{m=1}^{M} s_{m,i} \cdot \alpha_{q,m}$ and $\hat{z}_{q,j} = \sum_{n=1}^{N} z_{n,j} \cdot \beta_{q,n}$. Next, the prover sets $\pi'_i \leftarrow \pi_i \cdot \prod_{q=1}^{Q} (D'_q)^{\hat{s}_{q,i}}$ and $\psi'_j \leftarrow \psi_j \cdot \prod_{q=1}^{Q} (C_q)^{\hat{z}_{q,j}}$. These π'_i and ψ'_j will satisfy the verification equation for the new commitments. See Appendix 3.7.4 for details.

Now the prover must make a certain technical step to fully randomize the proof. Intuitively, for every set of commitments, there are many proofs $(\pi_1, \ldots, \pi_I, \psi_1, \ldots, \psi_J)$ that can satisfy the verification equation. Given one such proof, we can randomly choose another: The prover chooses $t_{i,j}, t_h \leftarrow R$, and multiplies each $\pi_i := \pi_i \cdot \prod_{j=1}^{J} v_j^{t_{i,j}}$ and each $\psi_j := \psi_j \cdot \prod_{i=1}^{I} u_i^{\sum_{h=1}^{H} t_h \eta_{h,i,j}} \prod_{i=1}^{I} u_i^{t_{i,j}}$. The value $\eta_{h,i,j}$ is defined by Groth and Sahai, see [57, Section 5, Equation 1] for a detailed explanation of this operation.

The algorithm outputs the new proof $[(\pi'_1, \ldots, \pi'_I, \psi'_1, \ldots, \psi'_J), \Pi']$ where Π' contains the internal commitments $C'_1, \ldots, C'_M$ and $D'_1, \ldots, D'_N$, and the description of the original pairing product equation.

Efficiency

From the algorithms above, it follows that it takes an equal amount of time to randomize a commitment or a proof as it does to to generate a fresh commitment or proof for the same type of statement. Thus, we refer to Section 3.4 for full details on the efficiency of the operations.

3.7.4 Proof that Groth-Sahai Proofs Are Randomizable

Correctness. Suppose an honest prover gets as input commitments $\{c_m\}$ and $\{d_n\}$, and a valid proof $\{\pi_i\}$ and $\{\psi_j\}$. The prover randomizes it and sends the verifier the new commitments $\{c'_m\}$ and $\{d'_n\}$, the new proof $\{\pi'_i\}$ and $\{\psi'_i\}$, and the original pairing product equation.

The verifier computes $c'_q \leftarrow \mu_1(a_q) \cdot \prod_{m=1}^{M} {c'}_m^{\alpha_{q,m}}$ and $d'_q \leftarrow \mu_2(b_q) \cdot \prod_{n=1}^{N} {d'}_n^{\beta_{q,n}}$.

[10]The internal commitments are needed because the GS proof system uses a commitment for every value it proves something about. In some sense saying that a statement is about a commitment is just assigning a name to this commitment.

Because the prover is honest, we know that

$$\begin{aligned}
\forall q : c'_q &= \mu_1(a_q) \cdot \prod_{m=1}^{M} {c'}_m^{\alpha_{q,m}} \\
&= \mu_1(a_q) \cdot \prod_{m=1}^{M} \big(c_m \prod_{i=1}^{I} u_i^{s_{m,i}}\big)^{\alpha_{q,m}} \\
&= \Big(\mu_1(a_q) \cdot \prod_{m=1}^{M} c_m^{\alpha_{q,m}}\Big) \cdot \Big(\prod_{m=1}^{M}\prod_{i=1}^{I} u_i^{s_{m,i}\alpha_{q,m}}\Big) \\
&= c_q \cdot \prod_{i=1}^{I} u_i^{\hat{s}_{q,i}} \\
\forall q : d'_q &= d_q \cdot \prod_{j=1}^{J} v_j^{\hat{z}_{q,j}}.
\end{aligned}$$

Before the prover fully randomizes the $\{\pi'_i\}$ and $\{\psi'_j\}$, we have that

$$\pi'_i = \pi_i \cdot \prod_{q=1}^{Q} (d'_q)^{\hat{s}_{q,i}} \qquad\qquad \psi'_j = \psi_j \cdot \prod_{q=1}^{Q} (c_q)^{\hat{z}_{q,j}}.$$

Due to the results of Groth and Sahai, we know that if a prover starts with a valid proof $\{\pi_i\}$ and $\{\psi_j\}$ and multiplies each π_i by $\prod_{j=1}^{J} v_j^{t_{i,j}}$ and each ψ_j by $\prod_{i=1}^{I} u_i^{\sum_{h=1}^{H} t_h \eta_{h,i,j}} \prod_{i=1}^{I} u_i^{t_{i,j}}$ the result is still a valid proof. Therefore, all we need to show is that if the prover skips this multiplication step, that the verification formula $\prod_{q=1}^{Q} E(c'_q, d'_q) = \mu_T(t) \cdot \prod_{i=1}^{I} E(u_i, \pi'_i) \cdot \prod_{j=1}^{J} E(\psi'_j, v_j)$. holds.

$$\begin{aligned}
\prod_{q=1}^{Q} E(c'_q, d'_q) &= \prod_{q=1}^{Q} E(c_q \prod_{i=1}^{I} u_i^{\hat{s}_{q,i}}, d_q \prod_{j=1}^{J} v_j^{\hat{z}_{q,j}}) \\
&= \Big[\prod_{q=1}^{Q} E(c_q, d_q)\Big] \cdot \Big[\prod_{q=1}^{Q} E(c_q, \prod_{j=1}^{J} v_j^{\hat{z}_{q,j}})\Big] \cdot \Big[\prod_{q=1}^{Q} E(\prod_{i=1}^{I} u_i^{\hat{s}_{q,i}}, d_q)\Big] \cdot \Big[\prod_{q=1}^{Q} E(\prod_{i=1}^{I} u_i^{\hat{s}_{q,i}}, \prod_{j=1}^{J} v_j^{\hat{z}_{q,j}})\Big] \\
&= \Big[\prod_{q=1}^{Q} E(c_q, d_q)\Big] \cdot \Big[\prod_{q=1}^{Q} E(c_q, \prod_{j=1}^{J} v_j^{\hat{z}_{q,j}})\Big] \cdot \Big[\prod_{q=1}^{Q} E(\prod_{i=1}^{I} u_i^{\hat{s}_{q,i}}, d_q)\Big] \cdot \Big[\prod_{q=1}^{Q}\prod_{i=1}^{I} E(u_i^{\hat{s}_{q,i}}, \prod_{j=1}^{J} v_j^{\hat{z}_{q,j}})\Big] \\
&= \Big[\prod_{q=1}^{Q} E(c_q, d_q)\Big] \cdot \Big[\prod_{j=1}^{J} E(\prod_{q=1}^{Q} c_q^{\hat{z}_{q,j}}, v_j)\Big] \cdot \Big[\prod_{i=1}^{I} E(u_i, \prod_{q=1}^{Q} d_q^{\hat{s}_{q,i}})\Big] \cdot \Big[\prod_{i=1}^{I} E(u_i, \prod_{q=1}^{Q}\prod_{j=1}^{J} v_j^{\hat{z}_{q,j}\hat{s}_{q,i}})\Big] \\
&= \Big[\prod_{q=1}^{Q} E(c_q, d_q)\Big] \cdot \Big[\prod_{j=1}^{J} E(\prod_{q=1}^{Q} c_q^{\hat{z}_{q,j}}, v_j)\Big] \cdot \Big[\prod_{i=1}^{I} E(u_i, \prod_{q=1}^{Q} (d_q \prod_{j=1}^{J} v_j^{\hat{z}_{q,j}})^{\hat{s}_{q,i}})\Big] \\
&= \Big[\prod_{q=1}^{Q} E(c_q, d_q)\Big] \cdot \Big[\prod_{j=1}^{J} E(\prod_{q=1}^{Q} c_q^{\hat{z}_{q,j}}, v_j)\Big] \cdot \Big[\prod_{i=1}^{I} E(u_i, \prod_{q=1}^{Q} (d'_q)^{\hat{s}_{q,i}})\Big] \\
&= \Big[\mu_T(t) \cdot \prod_{i=1}^{I} E(u_i, \pi_i) \cdot \prod_{j=1}^{J} E(\psi_j, v_j)\Big] \cdot \Big[\prod_{j=1}^{J} E(\prod_{q=1}^{Q} c_q^{\hat{z}_{q,j}}, v_j)\Big] \cdot \Big[\prod_{i=1}^{I} E(u_i, \prod_{q=1}^{Q} {d'_q}^{\hat{s}_{q,i}})\Big] \\
&= \mu_T(t) \cdot \Big[\prod_{i=1}^{I} E(u_i, \pi_i) \prod_{i=1}^{I} E(u_i, \prod_{q=1}^{Q} {d'_q}^{\hat{s}_{q,i}})\Big] \cdot \Big[\prod_{j=1}^{J} E(\psi_j, v_j) \prod_{j=1}^{J} E(\prod_{q=1}^{Q} c_q^{\hat{z}_{q,j}}, v_j)\Big] \\
&= \mu_T(t) \cdot \Big[\prod_{i=1}^{I} E(u_i, \pi_i \prod_{q=1}^{Q} {d'_q}^{\hat{s}_{q,i}})\Big] \cdot \Big[\prod_{j=1}^{J} E(\psi_j \prod_{q=1}^{Q} c_q^{\hat{z}_{q,j}}, v_j)\Big] \\
&= \mu_T(t) \cdot \prod_{i=1}^{I} E(u_i, \pi'_i) \cdot \prod_{j=1}^{J} E(\psi'_j, v_j)
\end{aligned}$$

Thus we see that the verification equation holds.

Extractability. The Groth-Sahai extractor works by using the trapdoor to "decrypt" the commitments. Recall that a commitment is of the form $c = \mathsf{Commit}(x, (r_1, \ldots, r_I)) = x \cdot \prod_{i=1}^{I} u_i^{r_i}$ for an *arbitrary* vector $(r_1, \ldots, r_I)$. When we randomize c, we compute

$$\begin{aligned}
c' &\leftarrow c \cdot \prod_{i=1}^{I} u_i^{s_i} \\
&= x \cdot \prod_{i=1}^{I} u_i^{r_i} \cdot \prod_{i=1}^{I} u_i^{s_i} \\
&= x \cdot \prod_{i=1}^{I} u_i^{r_i + s_i}.
\end{aligned}$$

Thus, we still have a commitment to x.

Randomness. First we consider the randomization of commitments. As we saw above, each new commitment is set to $c' \leftarrow x \cdot \prod_{i=1}^{I} u_i^{r_i+s_i}$. The randomness of c' depends on how the s_i are chosen. If the s_i

are chosen at random, then c' is a randomly chosen commitment for x. If the s_i are not chosen at random (especially if they are all set to 0!), then we have no such guarantees.

Now we argue about the randomization of proofs. When the prover computes $\pi'_i \leftarrow \pi_i \cdot \prod_{q=1}^{Q} (d'_q)^{\hat{s}_{q,i}}$ and $\psi'_j \leftarrow \psi_j \cdot \prod_{q=1}^{Q} (c_q)^{\hat{z}_{q,j}}$, the result is a valid proof (as we demonstrated above). Then the prover multiplies the $\{\pi'_i\}$ and $\{\psi'_i\}$ by a certain factor. Groth and Sahai show that the result is a randomly chosen proof from the space of all valid proofs, given that the commitments and pairing product equation are fixed. Since, as we showed above, the commitments are completely randomized, the result is a randomly chosen proof given a fixed solution $\{x_m\}, \{y_n\}$ to a particular pairing product equation.

Chapter 4

NIZK Proofs for Signature Schemes

In a series of papers, Camenisch and Lysyanskaya [23, 24, 25] identified a key building block commonly called "a CL-signature", which has been used in many privacy applications. These include constructions for anonymous credentials [23, 24, 25], for electronic cash [2, 21, 75, 22, 29], and for anonymous authentication [20]. A CL-signature is a signature scheme with a pair of useful protocols.

The first protocol, called *Issue*, lets a user obtain a signature on a committed message without revealing the message. The user wishes to obtain a signature on a value x from a signer with public key pk. The user forms a commitment $comm$ to value x and gives $comm$ to the signer. After running the protocol, the user obtains a signature on x, and the signer learns no information about x other than the fact that he has signed the value that the user has committed to.

The second protocol, called *Prove*, is a zero-knowledge proof of knowledge of a signature on a committed value. The prover has a message-signature pair $(x, \sigma_{pk}(x))$. The prover has obtained it by either running the Issue protocol, or by querying the signer on x. The prover also has a commitment $comm$ to x. The verifier only knows $comm$. The prover proves in zero-knowledge that he knows a pair (x, σ) and a value $open$ such that $\mathsf{VerifySig}(pk, x, \sigma) = \mathsf{accept}$ and $comm = \mathsf{Commit}(x, open)$.

It is clear that using general secure two-party computation [76] and zero-knowledge proofs of knowledge of a witness for any NP statement [51], we can construct the Issue and Prove protocols from any signature scheme and commitment scheme. Camenisch and Lysyanskaya's contribution was to construct specially designed signature schemes that, combined with Pedersen [69] and Fujisaki-Okamoto [48] commitments, allowed them to construct Issue and Prove protocols that are efficient enough for use in practice. In turn, CL-signatures have been implemented and standardized [18, 16]. They have also been used as a building block in many other constructions [60, 2, 21, 22, 40, 20, 73, 29, 26].

A shortcoming of the CL signature schemes is that the Prove protocol is interactive. Rounds of interaction are a valuable resource. In certain contexts, proofs need to be verified by third parties who are not present during the interaction. For example, in off-line e-cash, a merchant accepts an e-coin from a buyer and later deposits the e-coin to the bank. The bank must be able to verify that the e-coin is valid.

There are two known techniques for making the CL Prove protocols non-interactive. We can use the Fiat-Shamir heuristic [47], which requires the random-oracle model. A series of papers [30, 45, 52] show that proofs of security in the random-oracle model do not imply security. The other option is to use general techniques: [8, 42, 7] show how any statement in NP can be proven in non-interactive zero-knowledge. This option is prohibitively expensive.

We give the first *practical* non-interactive zero-knowledge proof of knowledge of a signature on a committed message. We have two constructions using two different practical signature schemes and a special class of commitments due to Groth and Sahai [57]. Our constructions are secure in the common reference string model.

Due to the fact that these protocols are so useful for a variety of applications, it is important to give a careful treatment of the security guarantees they should provide. In this paper, we introduce the concept of P-signatures — signatures with efficient **P**rotocols, and give a definition of security. The main difference between P-signatures and CL-signatures is that P-signatures have non-interactive proof protocols. (Our definition can be extended to encompass CL signatures as well.)

TECHNICAL ROADMAP. We use Groth and Sahai's f-extractable non-interactive proofs of knowledge [57] to build P-signatures. Groth and Sahai give three instantiations for their proof system, using SXDH, DLIN, and SDA assumptions. We can use either of the first two instantiations. The SDA-based instantiation does not give us the necessary extraction properties.

Another issue we confront is that Groth-Sahai proofs are f-extractable and not fully extractable. Suppose we construct a proof whose witness x contains $a \in Z_p$ and the opening of a commitment to a. For this commitment, we can only extract $b^a \in f(x)$ from the proof, for some base b. Note that the proof can be about multiple committed values. Thus, if we construct a proof of knowledge of (m, σ) where $m \in Z_p$ and $\mathsf{VerifySig}(pk, m, \sigma) = \mathsf{accept}$, we can only extract some function $F(m)$ from the proof. However, even if it is impossible to forge (m, σ) pairs, it might be possible to forge $(F(m), \sigma)$ pairs. Therefore, for our proof system to be meaningful, we need to define F-unforgeable signature schemes, i.e. schemes where it is impossible for an adversary to compute a $(F(m), \sigma)$ pair on his own.

Our first construction uses the Weak Boneh-Boyen (WBB) signature scheme [11]. Using a rather strong assumption, we prove that WBB is F-unforgeable and our P-signature construction is secure. Our second construction uses a weaker assumption and is based on the Full Boneh-Boyen signature scheme [11]. In this case we had to modify the Boneh-Boyen construction, however, because the GS proof system would not allow the knowledge extraction of the entire signature. Our first construction is much simpler, but it's security relies on an interactive and thus much stronger assumption, thus, we believe that both constructions are interesting. We also give two constructions which allow signatures on multiple messages, one in which the public key size grows linearly with the number of messages, and a second in which the public key size can remain fixed independent of the number of messages (although the underlying security assumption will depend on this number).

4.1 P-signatures: Definitions

A non-interactive P-signature scheme extends a signature scheme $(\mathsf{Setup}, \mathsf{Keygen}, \mathsf{Sign}, \mathsf{VerifySig})$ and a non-interactive commitment scheme $(\mathsf{Setup}, \mathsf{Commit})$. It consists of the following algorithms $(\mathsf{Setup},$

$\mathsf{Keygen}, \mathsf{Sign}, \mathsf{VerifySig}, \mathsf{Commit}, \mathsf{ObtainSig}, \mathsf{IssueSig}, \mathsf{Prove}, \mathsf{VerifyProof}, \mathsf{EqCommProve}, \mathsf{VerEqComm})$.

$\mathsf{Setup}(1^k)$**.** Outputs public parameters $params$. These parameters include parameters for the signature scheme and the commitment scheme.

$\mathsf{ObtainSig}(params, pk, m, comm, open) \leftrightarrow \mathsf{IssueSig}(params, sk, comm)$**.** These two interactive algorithms execute a signature issuing protocol between a user and the issuer. The user takes as input $(params, pk, m, comm, open)$ such that the value $comm = \mathsf{Commit}(params, m, open)$ and gets a signature σ as output. If this signature does not verify, the user sends "reject" to the issuer. The issuer gets $(params, sk, comm)$ as input and gets nothing as output.

$\mathsf{Prove}(params, pk, m, \sigma)$**.** Outputs the values $(comm, \pi, open)$, such that $comm = \mathsf{Commit}(params, m, open)$ and π is a proof of knowledge of a signature σ on m.

$\mathsf{VerifyProof}(params, pk, comm, \pi)$**.** Takes as input a commitment to a message m and a proof π that the message has been signed by owner of public key pk. Outputs accept if π is a valid proof of knowledge of $F(m)$ and a signature on m, and outputs reject otherwise.

$\mathsf{EqCommProve}(params, m, open, open')$**.** Takes as input a message and two commitment opening values. It outputs a proof π that $comm = \mathsf{Commit}(m, open)$ is a commitment to the same value as $comm' = \mathsf{Commit}(m, open')$. This proof is used to bind the commitment of a P-signature proof to a more permanent commitment.

$\mathsf{VerEqComm}(params, comm, comm', \pi)$. Takes as input two commitments and a proof and accepts if π is a proof that $comm$, $comm'$ are commitments to the same value.

Definition 27 (Secure P-Signature Scheme)**.** *Let F be a efficiently computable bijection (possibly parameterized by public parameters). A P-signature scheme is secure if* $(\mathsf{Setup}, \mathsf{Keygen}, \mathsf{Sign}, \mathsf{VerifySig})$ *form an F-unforgeable signature scheme, if* $(\mathsf{Setup}, \mathsf{Commit})$ *is a perfectly binding, strongly computationally hiding commitment scheme, if* $(\mathsf{Setup}, \mathsf{EqCommProve}, \mathsf{VerEqComm})$ *is a non-interactive proof system, and if the* Signer privacy, User privacy, Correctness, Unforgeability, *and* Zero-knowledge *properties hold:*

Correctness. An honest user who obtains a P-signature from an honest issuer will be able to prove to an honest verifier that he has a valid signature.

$$\begin{aligned}\forall m \in \{0,1\}^* : \Pr[&params \leftarrow \mathsf{Setup}(1^k); (pk, sk) \leftarrow \mathsf{Keygen}(params);\\ &\sigma \leftarrow \mathsf{Sign}(params, sk, m); (comm, \pi) \leftarrow \mathsf{Prove}(params, pk, m, \sigma) :\\ &\mathsf{VerifyProof}(params, pk, comm, \pi) = 1] = 1\end{aligned}$$

Signer privacy. No PPTM adversary can tell if it is running IssueSig with an honest issuer or with a simulator who merely has access to a signing oracle. Formally, there exists a simulator SimIssue such that for all PPTM adversaries $(\mathcal{A}_1, \mathcal{A}_2)$, there exists a negligible function ν so that:

$$\begin{aligned}
\Big| \Pr[&params \leftarrow \mathsf{Setup}(1^k); (sk, pk) \leftarrow \mathsf{Keygen}(params);\\
&(m, open, state) \leftarrow \mathcal{A}_1(params, sk);\\
&comm \leftarrow \mathsf{Commit}(params, m, open);\\
&b \leftarrow \mathcal{A}_2(state) \leftrightarrow \mathsf{IssueSig}(params, sk, comm) : b = 1]\\
- \Pr[&params \leftarrow \mathsf{Setup}(1^k); (sk, pk) \leftarrow \mathsf{Keygen}(params);\\
&(m, open, state) \leftarrow \mathcal{A}_1(params, sk);\\
&comm \leftarrow \mathsf{Commit}(params, m, open); \sigma \leftarrow \mathsf{Sign}(params, sk, m);\\
&b \leftarrow \mathcal{A}_2(state) \leftrightarrow \mathsf{SimIssue}(params, comm, \sigma) : b = 1]\Big| < \nu(k)
\end{aligned}$$

Note that we ensure that IssueSig and SimIssue gets an honest commitment to whatever $m, open$ the adversary chooses.

Since the goal of signer privacy is to prevent the adversary from learning anything except a signature on the opening of the commitment, this is sufficient for our purposes. Note that our SimIssue will be allowed to rewind $\mathcal{A}$. to Also, we have defined Signer Privacy in terms of a single interaction between the adversary and the issuer. A simple hybrid argument can be used to show that this definition implies privacy over many sequential instances of the issue protocol.

User privacy. No PPTM adversary $(\mathcal{A}_1, \mathcal{A}_2)$ can tell if it is running ObtainSig with an honest user or with a simulator. Formally, there exists a simulator $\mathsf{Sim} = \mathsf{SimObtain}$ such that for all PPTM adversaries $\mathcal{A}_1, \mathcal{A}_2$, there exists negligible function ν so that:

$$\begin{aligned}
\Big| \Pr[&params \leftarrow \mathsf{Setup}(1^k); (pk, m, open, state) \leftarrow \mathcal{A}_1(params);\\
&comm = \mathsf{Commit}(params, m, open);\\
&b \leftarrow \mathcal{A}_2(state) \leftrightarrow \mathsf{ObtainSig}(params, pk, m, comm, open) : b = 1]\\
- \Pr[&(params, sim) \leftarrow \mathsf{Setup}(1^k); (pk, m, open, state) \leftarrow \mathcal{A}_1(params);\\
&comm = \mathsf{Commit}(params, m, open);\\
&b \leftarrow \mathcal{A}_2(state) \leftrightarrow \mathsf{SimObtain}(params, pk, comm) : b = 1]\Big| < \nu(k)
\end{aligned}$$

Here again SimObtain is allowed to rewind the adversary.

Note that we require that only the user's input m is hidden from the issuer, but not necessarily the user's output σ. The reason that this is sufficient is that in actual applications (for example, in anonymous credentials), a user would never show σ in the clear; instead, he would just prove that he knows σ. An alternative, stronger way to define signer privacy and user privacy together, would be to require that the pair of algorithms ObtainSig and IssueSig carry out a secure two-party computation. This alternative definition would ensure that σ is hidden from the issuer as well. However, as explained above, this feature is not necessary for our application, so we preferred to give a special definition which captures the minimum properties required.

Unforgeability. We require that no PPTM adversary can create a proof for any message m for which he has not previously obtained a signature or proof from the oracle.

A P-signature scheme is unforgeable if an extractor $(\mathsf{ExtractSetup}, \mathsf{Extract})$ and a bijection F exist such that (1) the output of $\mathsf{ExtractSetup}(1^k)$ is indistinguishable from the output of $\mathsf{Setup}(1^k)$, and (2) no PPTM adversary can output a proof π that $\mathsf{VerifyProof}$ accepts, but from which we extract $F(m), \sigma$ such that either (a) σ is not valid signature on m, or (b) $comm$ is not a commitment to m or (c) the adversary has never previously queried the signing oracle on m. Formally, for all PPTM adversaries $\mathcal{A}$, there exists a negligible function ν such that:

$$
\begin{aligned}
&\Pr[params_0 \leftarrow \mathsf{Setup}(1^k); (params_1, td) \leftarrow \mathsf{ExtractSetup}(1^k) : b \leftarrow \{0,1\} : \\
&\quad \mathcal{A}(params_b) = b] < 1/2 + \nu(k), \text{ and} \\
&\Pr[(params, td) \leftarrow \mathsf{ExtractSetup}(1^k); (pk, sk) \leftarrow \mathsf{Keygen}(params); \\
&\quad (Q_{\mathsf{Sign}}, comm, \pi) \leftarrow \mathcal{A}(params, pk)^{\mathcal{O}_{\mathsf{Sign}}(params, sk, \cdot)}; \\
&\quad (y, \sigma) \leftarrow \mathsf{Extract}(params, td, \pi, comm) : \\
&\quad \mathsf{VerifyProof}(params, pk, comm, \pi) = \mathsf{accept} \\
&\quad \wedge (\mathsf{VerifySig}(params, pk, F^{-1}(y), \sigma) = \mathsf{reject} \\
&\qquad \vee (\forall open, comm \neq \mathsf{Commit}(params, F^{-1}(y), open)) \\
&\qquad \vee (\mathsf{VerifySig}(params, pk, F^{-1}(y), \sigma) = \mathsf{accept} \wedge y \notin F(Q_{\mathsf{Sign}})))] < \nu(k).
\end{aligned}
$$

Oracle $\mathcal{O}_{\mathsf{Sign}}(params, sk, m)$ runs the function $\mathsf{Sign}(params, sk, m)$ and returns the resulting signature σ to the adversary. It records the queried message on query tape Q_{Sign}. By $F(Q_{\mathsf{Sign}})$ we mean F applied to every message in Q_{Sign}.

Zero-knowledge. There exists a simulator $\mathsf{Sim} = (\mathsf{SimSetup}, \mathsf{SimProve}, \mathsf{SimEqComm})$, such that for all PPTM adversaries $\mathcal{A}_1, \mathcal{A}_2$, there exists a negligible function ν such that under parameters output by $\mathsf{SimSetup}$, Commit is perfectly hiding and (1) the parameters output by $\mathsf{SimSetup}$ are indistinguishable from those output by Setup, but $\mathsf{SimSetup}$ also outputs a special auxiliary string sim; (2) when $params$ are generated by $\mathsf{SimSetup}$, the output of $\mathsf{SimProve}(params, sim, pk)$ is indistinguishable from that of $\mathsf{Prove}(params, pk, m, \sigma)$ for all (pk, m, σ) where $\sigma \in \sigma_{pk}(m)$; and (3) when $params$ are generated by $\mathsf{SimSetup}$, the output of $\mathsf{SimEqComm}(params, sim, comm, comm')$ is indistinguishable from that of $\mathsf{EqCommProve}(params, m, open, open')$ for all $(m, open, open')$ where $comm = \mathsf{Commit}(params, m, open)$ and $comm' = \mathsf{Commit}(params, m, open')$.

In GMR notation, this is formally defined as follows:

$$
\begin{aligned}
&|\Pr[params \leftarrow \mathsf{Setup}(1^k); b \leftarrow \mathcal{A}(params) : b = 1] \\
&\quad - \Pr[(params, sim) \leftarrow \mathsf{SimSetup}(1^k); b \leftarrow \mathcal{A}(params) : b = 1]| < \nu(k), \text{ and} \\
&|\Pr[(params, sim) \leftarrow \mathsf{SimSetup}(1^k); (pk, m, \sigma, state) \leftarrow \mathcal{A}_1(params, sim); \\
&\quad (comm, \pi, open) \leftarrow \mathsf{Prove}(params, pk, m, \sigma); b \leftarrow \mathcal{A}_2(state, comm, \pi) : b = 1] \\
&\quad - \Pr[(params, sim) \leftarrow \mathsf{SimSetup}(1^k); (pk, m, \sigma, state) \leftarrow \mathcal{A}_1(params, sim); \\
&\quad (comm, \pi) \leftarrow \mathsf{SimProve}(params, sim, pk); b \leftarrow \mathcal{A}_2(state, comm, \pi) \\
&\quad : b = 1]| < \nu(k), \text{ and}
\end{aligned}
$$

$$\begin{aligned}
&|\Pr[(params, sim) \leftarrow \mathsf{SimSetup}(1^k); (m, open, open') \leftarrow \mathcal{A}_1(params, sim);\\
&\quad \pi \leftarrow \mathsf{EqCommProve}(params, m, open, open'); b \leftarrow \mathcal{A}_2(state, \pi) : b = 1]\\
&\quad - \Pr[(params, sim) \leftarrow \mathsf{SimSetup}(1^k); (m, open, open') \leftarrow \mathcal{A}_1(params, sim);\\
&\quad \pi \leftarrow \mathsf{SimEqComm}(params, sim, \mathsf{Commit}(params, m, open),\\
&\qquad\qquad\qquad\qquad\qquad \mathsf{Commit}(params, m, open'));\\
&\quad b \leftarrow \mathcal{A}_2(state, \pi) : b = 1]| < \nu(k).
\end{aligned}$$

4.2 P-signatures: Constructions

4.2.1 Construction of P-Signature Scheme from IHSDH

An F-unforgeable signature scheme

Our first construction of a P-signature scheme uses the Weak Boneh-Boyen signature scheme (WBB) signature scheme [11] as a building block. The WBB scheme is as follows:

WBB-$\mathsf{SigSetup}(1^k)$ runs $\mathsf{BilinearSetup}(1^k)$ to get the pairing parameters $(p, G_1, G_2, G_T, e, g, h)$. In the sequel, by z we denote $z = e(g, h)$.

WBB-$\mathsf{Keygen}(params_{Sig})$ The secret key is $\alpha \leftarrow Z_p$. $pk = (v, \tilde{v})$, where $v = h^\alpha$, $\tilde{v} = g^\alpha$.[1] The correctness of the public key can be verified by checking that $e(g, v) = e(\tilde{v}, h)$.

WBB-$\mathsf{Sign}(params_{Sig}, sk, m)$ calculates $\sigma = g^{1/(\alpha+m)}$, where $sk = \alpha$.

WBB-$\mathsf{VerifySig}(params_{Sig}, pk, m, \sigma)$ outputs accept if the public key is correctly formed and if $e(\sigma, vh^m) = z$, where $pk = (v, \tilde{v})$. Outputs reject otherwise.

Boneh and Boyen proved that the Weak Boneh-Boyen signature is only weakly secure given SDH, which is insufficient for our purposes.

In Appendix 4.2.1, we show that the weak Boneh-Boyen signature scheme is F-secure given IHSDH (which implies standard [54] security).

Theorem 6. *Let $F(x) = (h^x, u^x)$, where $u \in G_1$ and $h \in G_2$ as given in the statement of the IHSDH assumption. The Weak Boneh-Boyen signature scheme is F-secure given IHSDH.*

Proof. The proof of security is trivial given the IHSDH assumption. Correctness is straightforward. To prove unforgeability, we create a reduction to the IHSDH assumption. The reduction gets as input $(p, G_1, G_2, G_T, e, g, G, h, H, u)$, where $G = g^x$ and $H = h^x$ for some secret x. The reduction sets up the public parameters of the Boneh Boyen signature scheme $params = ((p, G_1, G_2, G_T, e, g, h)$ and a public-key $pk = (H, G)$. To answer a signature query on message m_ℓ, the reduction sends a query to $\mathcal{O}_w(m_\ell)$ and sends $g^{1/(x+m_\ell)}$ back to the adversary. Eventually, the adversary will output a forgery (σ, y), where $\sigma = g^{1/(x+m)}$, $y = F(m) = (h^m, u^m)$, and $m \neq msg_\ell$ for all ℓ. The reduction can then output the IHSDH tuple (σ, h^m, u^m). □

[1]The shadow value $\tilde{v}$ does not exist in [11] and is needed to prove zero-knowledge of our P-signatures in pairing settings in which no efficient isomorphisms exist.

A P-signature scheme

We extend the WBB scheme to obtain our first P-signature scheme (Setup, Keygen, Sign, $\mathsf{VerifySig}$, Commit, $\mathsf{ObtainSig}$, $\mathsf{IssueSig}$, Prove, $\mathsf{VerifyProof}$, $\mathsf{EqCommProve}$, $\mathsf{VerEqComm}$), as follows:

$\mathsf{Setup}(1^k)$ First, obtain $params_{BM} = (p, G_1, G_2, G_T, e, g, h) \leftarrow \mathsf{BilinearSetup}(1^k)$. Next, obtain $params_{GS} = (params_{BM}, params_1, params_2, params_\pi) \leftarrow \mathsf{GSSetup}(params_{BM})$. Pick $u \leftarrow G_1$. Let $params = (params_{GS}, u)$. As before, z is defined as $z = e(g, h)$.

$\mathsf{Keygen}(params)$ Run $\mathsf{WBB\text{-}Keygen}(params_{BM})$ and outputs $sk = \alpha$, $pk = (h^\alpha, g^\alpha) = (v, \tilde{v})$.

$\mathsf{Sign}(params, sk, m)$ Run $\mathsf{WBB\text{-}Sign}(params_{BM}, sk, m)$ to obtain $\sigma = g^{1/(\alpha+m)}$ where $\alpha = sk$.

$\mathsf{VerifySig}(params, pk, m, \sigma)$ Run $\mathsf{WBB\text{-}VerifySig}(params_{BM}, pk, m, \sigma)$.

$\mathsf{Commit}(params, m, open)$ To commit to m, compute $C = \mathsf{GSExpCommit}(params_2, h, m, open)$. (Recall that $\mathsf{GSExpCommit}(params_2, h, m, open) = \mathsf{GSCommit}(params_2, h^m, open)$, and $params_2$ is part of $params_{GS}$.)

$\mathsf{ObtainSig}(params, pk, m, comm, open) \leftrightarrow \mathsf{IssueSig}(params, sk, comm)$**.** The user and the issuer run the following protocol:

1. The user chooses $\rho \leftarrow Z_p$.
2. The user and issuer engage in a secure two-party computation protocol [61][2], where the user's private input is $(\rho, m, open)$, and the issuer's private input is $sk = \alpha$. The issuer's private output is $x = (\alpha + m)\rho$ if $comm = \mathsf{Commit}(params, m, open)$, and $x = \perp$ otherwise.
3. If $x \neq \perp$, the issuer calculates $\sigma' = g^{1/x}$ and sends σ' to the user.
4. The user computes $\sigma = \sigma'^\rho = g^{1/(\alpha+m)}$. The user checks that the signature is valid.

$\mathsf{Prove}(params, pk, m, \sigma)$ Check if pk and σ are valid, and if they are not, output $\perp$. Else, pick appropriate $open_1$, $open_2$, $open_3$ and form the following three GS commitments: $M_h = \mathsf{GSExpCommit}(params_2, h, m, open_1)$, $M_u = \mathsf{GSExpCommit}(params_1, u, m, open_2)$, $\Sigma = \mathsf{GSCommit}(params_1, \sigma, open_3)$. Compute the following proof: $\pi = \mathsf{NIPK}\{((M_h : a), (M_u : b), (\Sigma : x)) : e(u, a) = e(b, h) \wedge e(x, va) = z\}$. Output $(comm, \pi) = (M_h, \pi)$.

$\mathsf{VerifyProof}(params, pk, comm, \pi)$ Outputs accept if the proof π is a valid proof of the statement described above for $M_h = comm$ and for properly formed $pk = (v, \tilde{v})$.

$\mathsf{EqCommProve}(params, m, open, open')$ Let commitment $comm = \mathsf{Commit}(params, m, open) = \mathsf{GSCommit}(params_2, h^m, open)$ and $comm' = \mathsf{Commit}(params, m, open') = \mathsf{GSCommit}(params_2, h^m, open')$. Use the GS proof system as described in Section 3.5 to compute $\pi \leftarrow \mathsf{NIZKPK}\{((comm : a), (comm' : b) : a = b\}$.

$\mathsf{VerEqComm}(params, comm, comm', \pi)$ Verify the proof π using the GS proof system as described in Chapter 3.

[2]Jarecki and Shmatikov give a protocol for secure two-party computation on committed inputs; their construction can be adapted here. In general using secure two-party computation is expensive, but here we only need to compute a relatively small circuit on the inputs.

Efficiency

The P-signature proof system generates 3 new commitments (2 in G_1 and 1 in G_2) and GS proofs for 2 pairing product equations, one with $Q = 2$, and one with $Q = 1$. Applying the efficiency formulas given in Section 3.4, we get the following lemma:

Lemma 2. *When instantiated using the SXDH instantiation given in Section 3.3 the P-signature proofs will have the following efficiency: Generating the proof will require* 36 *exponentiations in* G_1 *and* 32 *exponentiations in* G_2*. The resulting proof will consist of* 12 *elements of* G_1 *and* 10 *elements of* G_2*. Verifying the proof will involve computing* 44 *bilinear group pairings.*

When instantiated using the symmetric setting version of the DLIN instantiation given in Section 3.3, the above proof system will have the following efficiency: Generating the proof will require 135 *exponentiations in* G*. The resulting proof will consist of* 27 *elements in* G*. Verifying the proof will involve computing* 81 *bilinear group pairings.*

Note that obtaining a full zero knowledge proof system will require a combination of the Prove algorithm and the $\mathsf{EqCommProve}$, whose efficiency is discussed in Section 3.5.

Security

Theorem 7 (Security)**.** *Our first P-signature construction is secure for* $F_{params}(m) = (h^m, u^m)$ *given IHSDH and the security of the GS commitments and proofs.*

Proof. *Correctness* follows from correctness of GS proofs.

Signer Privacy. We must construct the $\mathsf{SimIssue}$ algorithm that is given as input $params$, a commitment $comm$ and a signature σ and must simulate the adversary's view. $\mathsf{SimIssue}$ will invoke the simulator for the two-party computation protocol. Recall that in two-party computation, the simulator can first extract the input of the adversary: in this case, some $(\rho, m, open)$. Then $\mathsf{SimIssue}$ checks that $comm = \mathsf{Commit}(params, m, open)$; if it isn't, it terminates. Otherwise, it sends to the adversary the value $\sigma' = \sigma^{1/\rho}$. Suppose the adversary can determine that it is talking with a simulator. Then it must be the case that the adversary's input to the protocol was incorrect which breaks the security properties of the two-party computation.

User privacy. The simulator will invoke the simulator for the two-party computation protocol. Recall that in two-party computation, the simulator can first extract the input of the adversary (in this case, some α', not necessarily the valid secret key). Then the simulator is given the target output of the computation (in this case, the value x which is just a random value that the simulator can pick itself), and proceeds to interact with the adversary such that if the adversary completes the protocol, its output is x. Suppose the adversary can determine that it is talking with a simulator. Then it breaks the security of the two-party computation protocol.

Zero knowledge. Consider the following algorithms. $\mathsf{SimSetup}$ runs $\mathsf{BilinearSetup}(1^k)$ to get $params_{BM} = (p, G_1, G_2, G_T, e, g, h)$ and $\mathsf{GSSimSetup}(params_{BM})$ to get $params_{GS}, sim_{GS}$. It then picks $t \leftarrow Z_p$ and sets up $u = g^t$. The final parameters are $params = (params_{GS}, u, z = e(g, h))$ and $sim = (t, sim_{GS})$. Note that the distribution of $params$ is indistinguishable from the distribution output by Setup. Also note that using these parameters, the commitments generated by $\mathsf{GSCommit}$ are perfectly hiding.

SimProve receives $params$, sim, and public key $(v, \tilde{v})$ and can use trapdoor $sim = t$ to create a random P-signature forgery as follows. Pick $s \leftarrow Z_p$ and compute $\sigma = g^{1/s}$. We implicitly set $m = s - \alpha$. Note that the simulator does not know m and α. However, he can compute $h^m = h^s/v$ and $u^m = (g^s/\tilde{v})^t$. Now he can use σ, h^m, and u^m to create commitments. The proof π is computed in the same way as in the real Prove protocol using σ, h^m, and u^m and the opening information of the commitments as witnesses. By the witness indistinguishability of the GS proof system, a proof using the faked witnesses is indistinguishable from a proof using a real witness, thus SimProve is indistinguishable from Prove.

Finally, we need to show that we can simulate proofs of EqCommProve given the trapdoor sim_{GS}. This follows from composable zero knowledge of EqCommProve. See Section 3.5.

Unforgeability. Consider the following algorithms: ExtractSetup(1^k) outputs the usual $params$, except that it invokes GSExtractSetup to get alternative $params_{GS}$ and the trapdoor $td = (td_1, td_2)$ for extracting from GS commitments in G_1 and G_2. The parameters generated by GSSetup are indistinguishable from those generated by GSExtractSetup, so we know that the parameters generated by ExtractSetup will be indistinguishable from those generated by Setup.

Extract$(params, td, comm, \pi)$ extracts the values from commitment $comm$ and the commitments M_h, M_u contained in the proof π using the GS commitment extractor. If VerifyProof accepts then $comm = M_h$. Let $F(m) = (h^m, u^m)$.

Now suppose we have an adversary that can break the unforgeability of our P-signature scheme for this extractor and this bijection. We create a reduction to break the IHSDH assumption. The reduction gets $(p, G_1, G_2, G_T, e, g, \tilde{X}, h, X, u)$, where $X = h^x, \tilde{X} = g^x$ for some unknown x. The reduction runs GSExtractSetup$(p, G_1, G_2, G_T, e, g, h)$ to get $params_{GS}$ and td. It otherwise creates $params$ in the same way as Setup (and ExtractSetup). Note that td lets it open all commitments. The reduction gives $(params, td, pk = (X, \tilde{X}))$ to the adversary. Whenever the adversary queries $\mathcal{O}_{\mathsf{Sign}}$ on m, the reduction returns $\sigma \leftarrow \mathcal{O}_x(m)$ and stores m in Q_{Sign}.

Eventually, the adversary outputs a proof π. Since π is f-extractable and perfectly sound, Extract $(params, td, comm, \pi)$ will return $a = h^m$, $b = u^m$, and $\sigma = g^{1/(x+m)}$. Thus we have a valid IHSDH tuple and $m = F^{-1}(a, b)$ will always fulfill VerifySig. We also know that since VerifyProof accepts, $comm = M_h =$ Commit$(params, m, open)$ for some $open$. Thus, since this is a forgery, it must be the case that $(a, b) = F(m) \notin F(Q_{\mathsf{Sign}})$. This means that we never queried $\mathcal{O}_x$ on m and the reduction has generated a fresh IHSDH tuple. □

4.2.2 Construction of P-Signature Scheme from TDH and HSDH

In this section, we present a new signature scheme and then build a P-signature scheme from it.

An F-unforgeable signature scheme

The new signature scheme is inspired by the full Boneh-Boyen signature scheme, and is as follows:

New-SigSetup(1^k) runs BilinearSetup(1^k) to get the pairing parameters $(p, G_1, G_2, G_T, e, g, h)$. In the sequel, by z we denote $z = e(g, h)$.

New-Keygen$(params)$ picks a random $\alpha, \beta \leftarrow Z_p$. The signer calculates $v = h^{\alpha}$, $w = h^{\beta}$, $\tilde{v} = g^{\alpha}$, $\tilde{w} = g^{\beta}$. The secret-key is $sk = (\alpha, \beta)$. The public-key is $pk = (v, w, \tilde{v}, \tilde{w})$. The public key can be verified by checking that $e(g, v) = e(\tilde{v}, h)$ and $e(g, w) = e(\tilde{w}, h)$.

New-Sign$(params, (\alpha, \beta), m)$ chooses $r \leftarrow Z_p - \{\frac{\alpha - m}{\beta}\}$ and calculates $C_1 = g^{1/(\alpha+m+\beta r)}$, $C_2 = w^r$, $C_3 = u^r$. The signature is (C_1, C_2, C_3).

New-VerifySig$(params, (v, w, \tilde{v}, \tilde{w}), m, (C_1, C_2, C_3))$ outputs accept if $e(C_1, vh^m C_2) = z$, $e(u, C_2) = e(C_3, w)$, and if the public key is correctly formed, i.e., $e(g, v) = e(\tilde{v}, h)$, and $e(g, w) = e(\tilde{w}, h)$.[3]

Theorem 8. *Let $F(x) = (h^x, u^x)$, where $u \in G_1$ and $h \in G_2$ as in the HSDH and TDH assumptions. Our new signature scheme is F-secure given HSDH and TDH. (*See full version for proof.*)*

Theorem 9. *Let $F(x) = (h^x, u^x)$, where $u \in G_1$ and $h \in G_2$ as in the HSDH and TDH assumptions. Our new signature scheme is F-secure given HSDH and TDH.*

Proof. Correctness is straightforward. *Unforgeability* is more difficult. Suppose we try to do a straightforward reduction to HSDH. The reduction will setup the parameters for the signature scheme. Whenever the adversary queries $\mathcal{O}_{\mathsf{Sign}}$, the reduction will use one of the provided tuples $(g^{1/(x+c_\ell)}, g^{c_\ell}, v^{c_\ell})$ to construct a signature for input message m_ℓ. We choose r_ℓ such that $c_\ell = m_\ell + \beta r_\ell$. Thus, $C_1 = g^{1/(x+c_\ell)}$, $C_2 = w^{r_\ell}$ and $C_3 = u^{r_\ell}$. (The actual proof will be more complicated, because we don't know c_ℓ and therefore cannot calculate r_ℓ directly).

Eventually, the adversary returns $F(m) = (h^m, u^m)$ and a valid signature (C_1, C_2, C_3). Since the signature is valid, we get that $C_1 = g^{1/(x+m+\beta r)}$, $C_2 = w^r = h^{\beta r}$, and $C_3 = u^r$. We can have two types of forgeries. In Type 1, the adversary returns a forgery such that $m + \beta r \neq m_\ell + \beta r_\ell$ for all of the adversary's previously queried messages m_ℓ, in which case we can easily create a new HSDH tuple. In Type 2, the adversary returns a forgery such that $m + \beta r = m_\ell + \beta r_\ell$. In this case, we cannot use the forgery to construct a new HSDH tuple. Therefore, we divide our proof into two categories. In Type 1, we reduce to the HSDH assumption. In Type 2, we reduce to the TDH assumption.

Type 1 forgeries: $\beta r + m \neq \beta r_\ell + m_\ell$ for any r_ℓ, m_ℓ from a previous query. The reduction gets an instance of the HSDH problem $(p, G_1, G_2, G_T, e, g, v, \tilde{v}, h, u, \{C_\ell, H_\ell, U_\ell\}_{\ell=1\ldots q})$, such that $v = h^x$ and $\tilde{v} = g^x$ for some unknown x, and for all ℓ, $C_\ell = g^{1/(x+c_\ell)}$, $H_\ell = h^{c_\ell}$, and $U_\ell = u^{c_\ell}$ for some unknown c_ℓ. The reduction sets up the parameters of the new signature scheme as $(p, G_1, G_2, e, g, h, u, z = e(g, h))$. Next, the reduction chooses $\beta \leftarrow Z_p$ and calculates $w = h^{\beta}, \tilde{w} = g^{\beta}$. The reduction gives the adversary the public parameters and the public-key $(v, w, \tilde{v}, \tilde{w})$.

Suppose the adversary's ℓth query is to Sign message m_ℓ. The reduction will implicitly set r_ℓ to be such that $c_\ell = m_\ell + \beta r_\ell$. This is an equation with two unknowns, so we do not know r_ℓ and c_ℓ. The reduction sets $C_1 = C_\ell$. It computes $C_2 = H_\ell / h^{m_\ell} = h^{c_\ell}/h^{m_\ell} = w^{r_\ell}$. Then it computes $C_3 = (U_\ell / u^{m_\ell})^{1/\beta} = (u^{c_\ell}/u^{m_\ell})^{1/\beta} = u^{(c_\ell - m_\ell)/\beta} = u^{r_\ell}$. The reduction returns the signature (C_1, C_2, C_3).

Eventually, the adversary returns $F(m) = (F_1, F_2)$ and a valid signature (C_1, C_2, C_3). Since this is a valid F-forger, we get that $F_1 = h^m$, $F_2 = u^m$ and $C_1 = g^{1/(x+m+\beta r)}$, $C_2 = w^r = h^{\beta r}$, and $C_3 = u^r$.

[3]The latter is needed only once per public key, and is meaningless in a symmetric pairing setting.

Since this is a Type 1 forger, we also have that $m + \beta r \neq m_\ell + \beta r_\ell$ for any of the adversary's previous queries. Therefore, $(C_1, F_1 C_2, F_2 C_3^\beta) = (g^{1/(x+m+\beta r)}, h^{m+\beta r}, u^{m+\beta r})$ is a new HSDH tuple.

Type 2 forgeries: $\beta r + m = \beta r_\ell + m_\ell$ for some r_ℓ, m_ℓ from a previous query. The reduction receives $(p, G_1, G_2, G_T, e, g, h, X, Z, Y, \{\sigma_\ell, c_\ell\})$, where $X = h^x$, $Z = g^x$, $Y = g^y$, and for all ℓ, $\sigma_\ell = g^{1/(x+c_\ell)}$. The reduction chooses $\gamma \leftarrow Z_p$ and sets $u = Y^\gamma$. The reduction sets up the parameters of the new signature scheme as $(p, G_1, G_2, e, g, h, u, z = e(g, h))$. Next the reduction chooses $\alpha \leftarrow Z_p$, and calculates $v = h^\alpha$, $w = X^\gamma$, $\tilde{v} = g^\alpha$, $\tilde{w} = Z^\gamma$. It gives the adversary the parameters and the public-key $(v, w, \tilde{v}, \tilde{w})$. Note that we set up our parameters and public-key so that $\beta = x\gamma$, for some unknown β and $u = g^{\gamma y}$.

Suppose the adversary's ℓth query is to Sign message m_ℓ. The reduction sets $r_\ell = (\alpha + m_\ell)/(c_\ell \gamma)$ (which it can compute). The reduction computes $C_1 = \sigma_\ell^{1/(\gamma r_\ell)} = (g^{1/(x+c_\ell)})^{1/(\gamma r_\ell)} = g^{1/(\gamma r_\ell (x+c_\ell))} = g^{1/(\alpha+m_\ell+\beta r_\ell)}$. Since the reduction knows r_ℓ, it computes $C_2 = w^{r_\ell}$, $C_3 = u^{r_\ell}$ and send (C_1, C_2, C_3) to $\mathcal{A}$.

Eventually, the adversary returns $F(m) = (F_1, F_2)$ and a valid signature (C_1, C_2, C_3). Since this is an F-forgery, we get that $F_1 = h^m$, $F_2 = u^m$ and that $C_1 = g^{1/(x+m+\beta r)}$, $C_2 = w^r = h^{\beta r}$, and $C_3 = u^r$. Since this is a Type 2 forger, we also have that $m + \beta r = m_\ell + \beta r_\ell$ for one of the adversary's previous queries. (We can learn m_ℓ and r_ℓ by comparing $F_1 C_2$ to $h^{m_\ell} w^{r_\ell}$ for all ℓ.) We define $\delta = m - m_\ell$. Since $m + \beta r = m_\ell + \beta r_\ell$, we also get that $\delta = \beta(r_\ell - r)$. Using $\beta = x\gamma$, we get that $\delta = x\gamma(r_\ell - r)$. We compute: $A = F_1/h^{m_\ell} = h^{m-m_\ell} = h^\delta$, $B = u^{r_\ell}/C_3 = u^{r_\ell - r} = u^{\delta/\gamma x} = g^{y\delta/x}$ and $C = (F_2/u^{m_\ell})^{1/\gamma} = u^{(m-m_\ell)/\gamma} = u^{\delta/\gamma} = g^{\delta y}$. We implicitly set $\mu = \delta/x$, thus $(A, B, C) = (h^{\mu x}, g^{\mu y}, g^{\mu x y})$ is a valid TDH tuple. $\square$

A P-signature Scheme

We extend the above signature scheme to obtain our second P-signature scheme (Setup, Keygen, Sign, $\mathsf{VerifySig}$, Commit, $\mathsf{ObtainSig}$, $\mathsf{IssueSig}$, Prove, $\mathsf{VerifyProof}$, $\mathsf{EqCommProve}$, $\mathsf{VerEqComm}$). The algorithms are as follows:

$\mathsf{Setup}(1^k)$ First, obtain $params_{BM} = (p, G_1, G_2, G_T, e, g, h) \leftarrow \mathsf{BilinearSetup}(1^k)$. Next, obtain $params_{GS} = (params_{BM}, params_1, params_2, params_\pi) \leftarrow \mathsf{GSSetup}(params_{BM})$. Pick $u \leftarrow G_1$. Let $params = (params_{GS}, u)$. As before, z is defined as $z = e(g, h)$.

$\mathsf{Keygen}(params)$ Run the New-$\mathsf{Keygen}(params_{BM})$ and output $sk = (\alpha, \beta)$, $pk = (h^\alpha, h^\beta, g^\alpha, g^\beta) = (v, w, \tilde{v}, \tilde{w})$.

$\mathsf{Sign}(params, sk, m)$ Run New-$\mathsf{Sign}(params_{BM}, sk, m)$ to obtain $\sigma = (C_1, C_2, C_3)$ where $C_1 = g^{1/(\alpha+m+\beta r)}$, $C_2 = w^r$, $C_3 = u^r$, and $sk = (\alpha, \beta)$

$\mathsf{VerifySig}(params, pk, m, \sigma)$ Run New-$\mathsf{VerifySig}(params_{BM}, pk, m, \sigma)$.

$\mathsf{Commit}(params, m, open)$ To commit to m, compute $C = \mathsf{GSExpCommit}(params_2, h, m, open)$. (Recall that $\mathsf{GSExpCommit}(params_2, h, m, open) = \mathsf{GSCommit}(params_2, h^m, open)$, and $params_2$ is part of $params_{GS}$.)

$\mathsf{ObtainSig}(params, pk, m, comm, open) \leftrightarrow \mathsf{IssueSig}(params, sk, comm)$**.** The user and the issuer run the following protocol:

1. The user chooses $\rho_1, \rho_2 \leftarrow Z_p$.

2. The issuer chooses $r' \leftarrow Z_p$.

3. The user and the issuer run a secure two-party computation protocol where the user's private inputs are $(\rho_1, \rho_2, m, open)$, and the issuer's private inputs are $sk = (\alpha, \beta)$ and r'.

   The issuer's private output is $x = (\alpha + m + \beta\rho_1 r')\rho_2$ if $comm = \mathsf{Commit}(params, m, open)$, and $x = \perp$ otherwise.

4. If $x \neq \perp$, the issuer calculates $C'_1 = g^{1/x}$, $C'_2 = w^{r'}$ and $C'_3 = u^{r'}$, and sends (C'_1, C'_2, C'_3) to the user.

5. The user computes $C_1 = C_1'^{\rho_2}$, $C_2 = C_2'^{\rho_1}$, and $C_3 = C_3'^{\rho_1}$ and then verifies that the signature (C_1, C_2, C_3) is valid.

$\mathsf{Prove}(params, pk, m, \sigma)$ Check if pk and σ are valid, and if they are not, output $\perp$. Then the user computes commitments $\Sigma = \mathsf{GSCommit}(params_1, C_1, open_1)$, $R_w = \mathsf{GSCommit}(params_1, C_2, open_2)$, $R_u = \mathsf{GSCommit}(params_1, C_3, open_3)$, $M_h = \mathsf{GSExpCommit}(params_2, h, m, open_4) = \mathsf{GSCommit}$ $(params_2, h^m, open_4)$ and $M_u = \mathsf{GSExpCommit}(params_1, u, m, open_5) = \mathsf{GSCommit}(params_1, u^m,$ $open_5)$.

The user outputs the commitment $comm = M_h$ and the proof

$$\pi = \mathsf{NIPK}\{((\Sigma : C_1), (R_w : C_2), (R_u : C_3)(M_h : a), (M_u : b)) : \\ e(C_1, vaC_2) = z \wedge e(u, C_2) = e(C_3, w) \wedge e(u, a) = e(b, h)\}.$$

$\mathsf{VerifyProof}(params, pk, comm, \pi)$ Outputs accept if the proof π is a valid proof of the statement described above for $M_h = comm$ and for properly formed pk.

$\mathsf{EqCommProve}(params, m, open, open')$ Let commitment $comm = \mathsf{Commit}(params, m, open) = \mathsf{GSCommit}(params_2, h^m, open)$ and $comm' = \mathsf{Commit}(params, m, open') = \mathsf{GSCommit}(params_2, h^m, open')$. Use the GS proof system as described in Section 3.5 to compute $\pi \leftarrow \mathsf{NIZKPK}\{((comm : a), (comm' : b) : a = b\}$.

$\mathsf{VerEqComm}(params, comm, comm', \pi)$ Verify the proof π using the GS proof system as described in Section 2.1.3.

Efficiency

The P-signature proof system generates 5 new commitments (3 in G_1 and 2 in G_2) and GS proofs for 3 pairing product equations, one with $Q = 1$, and two with $Q = 2$. Applying the efficiency formulas given in Section 3.4, we get the following lemma:

Lemma 3. *When instantiated using the SXDH instantiation given in Section 3.3 the P-signature proofs will have the following efficiency: Generating the proof will require* 56 *exponentiations in* G_1 *and* 52 *exponentiations in* G_2*. The resulting proof will consist of* 18 *elements of* G_1 *and* 16 *elements of* G_2*. Verifying the proof will involve computing* 68 *bilinear group pairings.*

When instantiated using the symmetric setting version of the DLIN instantiation given in Section 3.3, the above proof system will have the following efficiency: Generating the proof will require 216 *exponentiations* *in G. The resulting proof will consist of 42 elements in G. Verifying the proof will involve computing 126 bilinear group pairings.*

Security

Theorem 10 (Security)**.** *Our second P-signature construction is secure given HSDH and TDH and the security of the GS commitments and proofs.*

Proof. *Correctness.* VerifyProof will always accept properly formed proofs.

Signer Privacy. We must construct the SimIssue algorithm that is given as input $params$, a commitment $comm$ and a signature $\sigma = (C_1, C_2, C_3)$ and must simulate the adversary's view. SimIssue will invoke the simulator for the two-party computation protocol. Recall that in two-party computation, the simulator can first extract the input of the adversary: in this case, some $(\rho_1, \rho_2, m, open)$. Then SimIssue checks that $comm = \mathsf{Commit}(params, m, open)$; if it isn't, it terminates. Otherwise, it sends to the adversary the values $(C_1' = C_1^{1/\rho_2}, C_2' = C_2^{1/\rho_1}, C_3' = C_3^{1/\rho_1})$. Suppose the adversary can determine that it is talking with a simulator. Then it must be the case that the adversary's input to the protocol was incorrect which breaks the security properties of the two-party computation.

User Privacy. The simulator will invoke the simulator for the two-party computation protocol. Recall that in two-party computation, the simulator can first extract the input of the adversary (in this case, some (α', β'), not necessarily the valid secret key). Then the simulator is given the target output of the computation (in this case, the value x which is just a random value that the simulator can pick itself), and proceeds to interact with the adversary such that if the adversary completes the protocol, its output is x. Suppose the adversary can determine that it is talking with a simulator. Then it breaks the security of the two-party computation protocol.

Zero knowledge. Consider the following algorithms. SimSetup runs BilinearSetup to get $params_{BM} = (p, G_1, G_2, G_T, e, g, h)$. It then picks $t \leftarrow Z_p$ and sets up $u = g^a$. Next it calls $\mathsf{GSSimSetup}(params_{BM})$ to obtain $params_{GS}$ and sim. The final parameters are $params = (params_{GS}, u, z = e(g, h))$ and $sim = (a, sim)$. Note that the distribution of $params$ is indistinguishable from the distribution output by Setup. SimProve receives $params$, sim, and public key $(v, \tilde{v}, w, \tilde{w})$ and can use trapdoor sim to create a random P-signature forgery in SimProve as follows. Pick $s, r \leftarrow Z_p$ and compute $\sigma = g^{1/s}$. We implicitly set $m = s - \alpha - r\beta$. Note that the simulator does not know m and α. However, he can compute $h^m = h^s/(vw^r)$ and $u^m = u^s/(\tilde{v}^a \tilde{w}^{ar})$. Now he can use σ, h^m, u^m, w^r, u^r as a witness and construct the proof π in the same way as the real Prove protocol. By the witness indistinguishability of the GS proof system, a proof using the faked witnesses is indistinguishable from a proof using a real witness, thus SimProve is indistinguishable from Prove.

Finally, we need to show that we can simulate proofs of EqCommProve given the trapdoor sim_{GS}. This follows directly from composable zero knowledge of EqCommProve. (See Section 3.5.)

Unforgeability. Consider the following algorithms: $\mathsf{ExtractSetup}(1^k)$ outputs the usual $params$, except that it invokes GSExtractSetup to get alternative $params_{GS}$ and the trapdoor $td = (td_1, td_2)$ for extracting GS commitments in G_1 and G_2. The parameters generated by GSSetup are indistinguishable from those generated by $\mathsf{GSExtractSetup}$, so we know that the parameters generated by $\mathsf{ExtractSetup}$ will be indistinguishable from those generated by Setup.

$\mathsf{Extract}(params, td, comm, \pi)$ extracts the values from commitment $comm$ and the commitments M_h, M_u contained in the proof π using the GS commitment extractor. If $\mathsf{VerifyProof}$ accepts then $comm = M_h$. Let $F(m) = (h^m, u^m)$.

Now suppose we have an adversary that can break the unforgeability of our P-signature scheme for this extractor and this bijection.

A P-signature forger outputs a proof from which we extract $(F(m), \sigma)$ such that either (1) $\mathsf{VerifySig}$ $(params, pk, m, \sigma) = \mathsf{reject}$, or (2) $comm$ is not a commitment to m, or (3) the adversary never queried us on m. Since $\mathsf{VerifyProof}$ checks a set of pairing product equations, f-extractability of the GS proof system trivially ensures that (1) never happens. Since $\mathsf{VerifyProof}$ checks that $M_h = comm$, this ensures that (2) never happens. Therefore, we consider the third possibility. The extractor calculates $F(m) = (h^m, u^m)$ where m is fresh. Due to the randomness element r in the signature scheme, we have two types of forgeries. In a Type 1 forgery, the extractor can extract from the proof a tuple of the form $(g^{1/(\alpha+m+\beta r)}, w^r, u^r, h^m, u^m)$, where $m + r\beta \neq m_\ell + r_\ell\beta$ for any (m_ℓ, r_ℓ) used in answering the adversary's signing or proof queries. The second type of forgery is one where $m + r\beta = m_\ell + r_\ell\beta$ for (m_ℓ, r_ℓ) used in one of these previous queries. We show that a Type 1 forger can be used to break the HSDH assumption, and a Type 2 forger can be used to break the TDH assumption.

Type 1 forgeries: $\beta r + m \neq \beta r_\ell + m_\ell$ for any r_ℓ, m_ℓ from a previous query. The reduction gets an instance of the HSDH problem $(p, G_1, G_2, G_T, e, g, X, \tilde{X}, h, u, \{C_\ell, H_\ell, U_\ell\}_{\ell=1\ldots q})$, such that $X = h^x$ and $\tilde{X} = g^x$ for some unknown x, and for all ℓ, $C_\ell = g^{1/(x+c_\ell)}$, $H_\ell = h^{c_\ell}$, and $U_\ell = u^{c_\ell}$ for some unknown c_ℓ. The reduction sets up the parameters of the new signature scheme as $(p, G_1, G_2, e, g, h, u, z = e(g, h))$. Next, the reduction chooses $\beta \leftarrow Z_p$, sets $v = X, \tilde{v} = \tilde{X}$ and calculates $w = h^\beta, \tilde{w} = g^\beta$. The reduction gives the adversary the public parameters, the trapdoor, and the public-key $(v, w, \tilde{v}, \tilde{w})$.

Suppose the adversary's ℓth query is to Sign message m_ℓ. The reduction will implicitly set r_ℓ to be such that $c_\ell = m_\ell + \beta r_\ell$. This is an equation with two unknowns, so we do not know r_ℓ and c_ℓ. The reduction sets $C_1 = C_\ell$. It computes $C_2 = H_\ell / h^{m_\ell} = h^{c_\ell}/h^{m_\ell} = w^{r_\ell}$. Then it computes $C_3 = (U_\ell)^{1/\beta}/u^{m_\ell/\beta} = (u^{c_\ell})^{1/\beta}/u^{m_\ell/\beta} = u^{(c_\ell - m_\ell)/\beta} = u^{r_\ell}$ The reduction returns the signature (C_1, C_2, C_3).

Eventually, the adversary returns a proof π. Since π is f-extractable and perfectly sound, we extract $\sigma = g^{1/(x+m+\beta r)}$, $a = w^r$, $b = u^r$, $c = h^m$, and $d = u^m$. Since this is a P-signature forgery, $(c, d) = (h^m, u^m) \notin F(Q_{\mathsf{Sign}})$. Since this is a Type 1 forger, we also have that $m + \beta r \neq m_\ell + \beta r_\ell$ for any of the adversary's previous queries. Therefore, $(\sigma, ca, db^\beta) = (g^{1/(x+m+\beta r)}, h^{m+\beta r}, u^{m+\beta r})$ is a new HSDH tuple.

Type 2 forgeries: $\beta r + m = \beta r_\ell + m_\ell$ for some r_ℓ, m_ℓ from a previous query. The reduction receives $(p, G_1, G_2, G_T, e, g, h, X, Z, Y, \{\sigma_\ell, c_\ell\})$, where $X = h^x$, $Z = g^x$, $Y = g^y$, and for all ℓ, $\sigma_\ell = g^{1/(x+c_\ell)}$. The reduction chooses $\gamma \leftarrow Z_p$ and sets $u = Y^\gamma$. The reduction sets up the parameters of the new signature scheme as $(p, G_1, G_2, e, g, h, u, z = e(g, h))$. Next the reduction chooses $\alpha \leftarrow Z_p$, and calculates $v = h^\alpha$, $w = X^\gamma$, $\tilde{v} = g^\alpha$, $\tilde{w} = Z^\gamma$. It gives the adversary the parameters, the trapdoor, and the public-key $(v, w, \tilde{v}, \tilde{w})$. Note that we set up our parameters and public-key so that β is implicitly defined as $\beta = x\gamma$, and

$u = g^{\gamma y}$.

Suppose the adversary's ℓth query is to Sign message m_ℓ. The reduction sets $r_\ell = (\alpha + m_\ell)/(c_\ell \gamma)$ (which it can compute). The reduction computes $C_1 = \sigma_\ell^{1/(\gamma r_\ell)} = (g^{1/(x+c_\ell)})^{1/(\gamma r_\ell)} = g^{1/(\gamma r_\ell (x+c_\ell))} = g^{1/(\alpha + m_\ell + \beta r_\ell)}$. Since the reduction knows r_ℓ, it computes $C_2 = w^{r_\ell}$, $C_3 = u^{r_\ell}$ and send (C_1, C_2, C_3) to $\mathcal{A}$.

Eventually, the adversary returns a proof π. The proof π is f-extractable and perfectly sound, the reduction can extract $\sigma = g^{1/(x+m+\beta r)}$, $a = w^r$, $b = u^r$, $c = h^m$, and $d = u^m$. Therefore, VerifySig will always accept $m = F^{-1}(c, d)$, σ, a, b. We also know that if this is a forgery, then VerifyProof accepts, which means that $comm = M_h$, which is a commitment to m. Thus, since this is a P-signature forgery, it must be the case that $(c, d) = (h^m, u^m) \notin F(Q_{\mathsf{Sign}})$. However, since this is a Type 2 forger, we also have that $\exists \ell : m + \beta r = m_\ell + \beta r_\ell$, where m_ℓ is one of the adversary's previous Sign or Prove queries. We implicitly define $\delta = m - m_\ell$. Since $m + \beta r = m_\ell + \beta r_\ell$, we also get that $\delta = \beta(r_\ell - r)$. Using $\beta = x\gamma$, we get that $\delta = x\gamma(r_\ell - r)$. We compute: $A = c/h^{m_\ell} = h^{m - m_\ell} = h^\delta$, $B = u^{r_\ell}/b = u^{r_\ell - r} = u^{\delta/(\gamma x)} = g^{y\delta/x}$ and $C = (d/u^{m_\ell})^{1/\gamma} = u^{(m-m_\ell)/\gamma} = u^{\delta/\gamma} = g^{\delta y}$. We implicitly set $\mu = \delta/x$, thus $(A, B, C) = (h^{\mu x}, g^{\mu y}, g^{\mu x y})$ is a valid TDH tuple. □

4.2.3 P-signatures with Multiple Messages from HSDH and TDH

We will first present an F-unforgeable Multi-Block Signature Scheme, and then build a P-signature scheme from it.

An F-unforgeable signature scheme

$\mathsf{Setup}(1^k)$. Let G_1, G_2, G_T be groups of prime order q, such that $|q| = k$ and there exists a bilinear map $e : G_1 \times G_2 \to G_T$. Let g, u be generators for G_1, and h be a generator for G_2. Let $params_{GS}$ be the parameters for a Groth-Sahai NIZK proof system with either the XDH or DLIN setup. Output parameters $params_{PK} = (q, G_1, G_2, G_T, g, h, u, params_{GS})$. By z we denote $z = e(g, h)$.

$\mathsf{Keygen}(params)$ picks a random $\alpha, \beta_0, \ldots, \beta_n \leftarrow Z_p$. The signer calculates $v = h^\alpha$, $w_i = h^{\beta_i}$, $\tilde{v} = g^\alpha$, $\tilde{w}_i = g^{\beta_i}$. The secret-key is $sk = (\alpha, \vec{\beta})$. The public-key is $pk = (v, \vec{w}, \tilde{v}, \vec{\tilde{w}})$. The public key can be verified by checking that $e(g, v) = e(\tilde{v}, h)$ and $e(g, w_i) = e(\tilde{w}_i, h)$ for all i.

$\mathsf{Sign}(params, (\alpha, \beta), m)$ chooses $r \leftarrow Z_p - \{\frac{\alpha - (\beta_1 m_1 + \cdots + \beta_n m_n)}{\beta_0}\}$ and calculates $C_1 = g^{1/(\alpha + \beta_0 r + \beta_1 m_1 + \cdots + \beta_n m_n)}$, $C_2 = w_0^r$, $C_3 = u^r$. The signature is (C_1, C_2, C_3).

$\mathsf{VerifySig}(params, (v, w, \tilde{v}, \tilde{w}), \vec{m}, (C_1, C_2, C_3))$ outputs accept if $e(C_1, vC_2 \prod_{i=1}^n w_i^{m_i}) = z$ and $e(u, C_2) = e(C_3, w)$.

Theorem 11. *Let $F(m_1, \ldots, m_n) = (h^{m_1}, u^{m_1}, \ldots, h^{m_n}, u^{m_n})$. The above signature scheme is F-unforgeable given the HSDH and TDH assumptions.*

Proof. We distinguish two types of forgeries. In a Type 1 forgery the signature consists of a tuple (C_1, C_2, C_3) where $C_1 \neq C_1^{(q)}$ for any $C_1^{(q)}$ used in answering the forger's signature queries. We will show how a

Type 1 forger can be used to break the HSDH assumption. In a Type 2 forgery, $\exists q : C_1 = C_1^{(q)}$. We will show how a Type 2 forger can be used to break the TDH assumption.

Type 1 forgeries: The reduction gets as input $g, \tilde{v} = g^{\alpha}, u, h, v = h^{\alpha}, \{Z_q = g^{1/(\alpha+c_q)}, H_q = h^{c_q}, U_q = u^{c_q}\}_q = 1...Q$, as well as a description of the groups q, G_1, G_2, G_T. It needs to compute a new tuple $(g^{1/(\alpha+c)}, h^c, u^c)$ such that $\forall q : c \neq c_q$.

$\mathsf{Setup}(1^k)$. The reduction computes $params_{GS}$ and gives the adversary public parameters $params_{PK} = (q, G_1, G_2, G_T, g, h, u, params_{GS})$.

$\mathsf{Keygen}(params)$. The reduction picks a random $\beta_0, \ldots, \beta_n \leftarrow Z_p$. It computes, $\forall i \in [0, n] : w_i = h^{\beta_i}$ and $\tilde{w_i} = g_i^{\beta}$. The reduction gives the adversary the public key $pk = (v, \vec{w}, \tilde{v}, \vec{\tilde{w}})$. (The secret key is $sk = (\alpha, \vec{\beta})$, though the reduction does not know α.)

$\mathcal{O}_{\mathsf{Sign}}(params, (\alpha, \beta), \vec{m})$. At the first sign query, the reduction sets the counter $q = 1$, and increments it after responding to each sign query. The reduction will implicitly set $c_q = \beta_0 r + \sum_{i=1}^n \beta_i m_i$. Thus, $r = (c_q - \sum_{i=1}^n \beta_i m_i)/\beta_0$. However, the reduction does not know c_q and r. The reduction computes C_1, C_2, C_3 as follows:

$$\begin{aligned}
C_1 &= Z_q = g^{1/(\alpha+c_q)} \\
C_2 &= H_q / h^{\sum_{i=1}^n \beta_i m_i} = h^{c_q - \sum_{i=1}^n \beta_i m_i} = h^{\beta_0 r} = w_0^r \\
C_3 &= \left(U_q / h^{\sum_{i=1}^n \beta_i m_i}\right)^{1/\beta_0} = u^{(c_q - \sum_{i=1}^n \beta_i m_i)/\beta_0} = u^r
\end{aligned}$$

Forgery. Eventually, the adversary returns a Type 1 forgery $F(\vec{m}) = (h^{m_1}, u^{m_1}, \ldots, h^{m_n}, u^{m_n}) = (A_1, B_1, \ldots, A_n, B_n)$ and $C_1 = g^{1/(\alpha+\beta_0 r+\beta_1 m_1+\cdots+\beta_n m_n)}$, $C_2 = w_0^r = h^{\beta_0 r}$, and $C_3 = u^r$. We implicitly set $c = \beta_0 r + \beta_1 m_1 + \cdots + \beta_n m_n$. We now have $C_1 = g^{1/(\alpha+c)}$. We compute the rest of the HSDH tuple:

$$\begin{aligned}
A &= C_2 \prod_{i=1}^n A_i^{\beta_i} = h^{\beta_0 r} \prod_{i=1}^n h^{m_i \beta_i} = h^c \\
B &= C_3^{\beta_0} \prod_{i=1}^n B_i^{\beta_i} = u^{\beta_0 r} \prod_{i=1}^n u^{m_i \beta_i} = u^c
\end{aligned}$$

The tuple (C_1, A, B) is a fresh HSDH tuple because this is a Type 1 forgery ($\forall q : c \neq c_q$).

Type 2 forgeries: The reduction gets as input $g, G = g^x, u = g^y, h, H = h^x, \{c_q, Z_q = g^{1/(x+c_q)}\}_q = 1...Q$, as well as a description of the groups q, G_1, G_2, G_T. It needs to compute the tuple $(h^{\mu\alpha}, g^{\mu y}, g^{\mu x y})$ such that $\mu \neq 0$.

$\mathsf{Setup}(1^k)$. The reduction computes $params_{GS}$ and gives the adversary public parameters $params_{PK} = (q, G_1, G_2, G_T, g, h, u, params_{GS})$.

$\mathsf{Keygen}(params)$**.** The reduction picks a random $\alpha, \beta_0, \dots, \beta_n \leftarrow Z_p$. It computes, $\forall i \in [0, n] : w_i = h^{\beta_i}$ and $\tilde{w_i} = g_i^{\beta}$. Then, it chooses a random $t \leftarrow \{1, n\}$ and $\gamma \leftarrow Z_p$, and sets $w_t = H^\gamma = h^{x\gamma}$ and $\tilde{w}_t = G^\gamma = g^{x\gamma}$. The reduction gives the adversary the public key $pk = (v, \vec{w}, \tilde{v}, \vec{\tilde{w}})$. The secret key is $sk = (\alpha, \vec{\beta})$, though the reduction does not know $\beta_t = x\gamma$.)

$\mathcal{O}_{\mathsf{Sign}}(params, (\alpha, \beta), \vec{m})$**.** At the first sign query, the reduction sets the counter $q = 1$, and increments it after responding to each sign query. The reduction sets $c_q = (\alpha + \beta r + \sum_{i=1, i \neq t}^{n} \beta_i m_i)/\gamma m_t$, and solves for r. Then it is easy to verify that

$$(x + c_q)\gamma m_t = x\gamma m_t + \alpha + \beta_0 r + \sum_{i=1, i\neq t}^{n} \beta_i m_i = \alpha + \beta_0 r + \sum_{i=1}^{n} \beta_t m_t.$$

This is precisely the inverse of the exponent which forms the first part of the signature, C_1. The reduction sets $C_1 = Z_q^{1/\gamma m_t} = g^{1/(x+c_q)\gamma m_t}$. Since the reduction knows r, it computes $C_2 = w_0^r$ and $C_3 = u^r$.

Forgery. Eventually, the adversary returns a Type 2 forgery $F(\vec{m}) = (h^{m_1}, u^{m_1}, \dots, h^{m_n}, u^{m_n}) = (A_1, B_1, \dots, A_n, B_n)$ and $C_1 = g^{1/(\alpha + \beta_0 r + \sum_{i=1}^{n} \beta_i m_i)}$, $C_2 = w_0^r = h^{\beta_0 r}$, and $C_3 = u^r$.

Since this is a Type 2 forgery, there the reduction already has a message / signature pair such that $\beta_0 r + \sum_{i=1}^{n} \beta_i m_i = \beta_0 r^{(q)} + \sum_{i=1}^{n} \beta_i m_i^{(q)}$. Since it is a forgery, $\exists m_i \in \vec{m} : m_i \neq m_i^{(q)}$. With probability $1/n$, $i = t$.

That means that $m_t \neq m_t^{(q)}$ but $\beta_t m_t + \beta_0 r + \sum_{i=1, i\neq t}^{n} \beta_i m_i = \beta_t m_t^{(q)} + \beta_0 r^{(q)} + \sum_{i=1, i\neq t}^{n} \beta_i m_i^{(q)}$. Now, we will implicitly set $\mu = (m_t^{(q)} - m_t)\gamma$ (Note that if we have guessed t correctly, this will be nonzero). We cannot compute this value, however we will be able to compute $h^{\mu x}, g^{\mu y}, g^{\mu x y}$ as follows:

Compute the following values:

$$\begin{aligned}
W_1 &= (B_t/(u^{m_t^{(q)}}))^\gamma = (u^{m_t - m_t^{(q)}})^\gamma = u^\mu = g^{y\mu} \\
W_2 &= \prod_{i=1, i\neq t}^{n} (A_i/(h^{m_i^{(q)}}))^{\beta_i} C_2/h^{r^{(q)}\beta_0} = h^{\sum_{i=1,i\neq t}^{n} \beta_i (m_i - m_i^{(q)})} h^{\beta_0 (r - r^{(q)})} = h^{\beta_t (m_t^{(q)} - m_t)} \\
&= h^{x\gamma(m_t^{(q)} - m_t)} = h^{x\mu} \\
W_3 &= \prod_{i=1, i\neq t}^{n} (B_i/(u^{m_i^{(q)}}))^{\beta_i} (C_3/u^{r^{(q)}})^{\beta_0} = u^{\sum_{i=1,i\neq t}^{n} \beta_i (m_i - m_i^{(q)})} u^{\beta_0 (r - r^{(q)})} = u^{\beta_t (m_t^{(q)} - m_t)} \\
&= u^{x\gamma(m_t^{(q)} - m_t)} = u^{x\mu} = g^{xy\mu}
\end{aligned}$$

The reduction will output W_1, W_2, W_3, which will be a valid tuple iff this is a type 2 forgery such that $m_t \neq m_t^{(q)}$.

$\square$

A Multi-Block P-Signature Scheme

We will use the above signature scheme and the commitment scheme obtained by $\mathsf{Commit}(x, open_x) = \mathsf{GSCommit}(h^x, open_x)$ where h is the generator of bilinear group G_2.

We need to augment the multi-block signature scheme with the three P-Signature protocols.

1. $\mathsf{ProveSig}(params, (v, \vec{w}, \tilde{v}, \vec{\tilde{w}}), (C_1, C_2, C_3), \vec{m})$ is defined as follows: We use Groth-Sahai commitments to commit to $(h^{m_1}, u^{m_1}, \ldots, h^{m_n}, u^{m_n})$, resulting in commitments $(S_{H_1}, S_{U_1}, \ldots, S_{H_n}, S_{U_n})$. We also commit to $(w_1^{m_1}, \ldots w_n^{m_n})$ resulting in commitments $S_{W_1}, \ldots S_{W_n}$. Next, we commit to the signature (C_1, C_2, C_3) to get commitments (S_1, S_2, S_3). Finally, we form the Groth-Sahai witness indistinguishable proof:

$$\begin{aligned}\pi \leftarrow \mathsf{NIPK}[&(H_1, U_1, \ldots H_n, U_n)\,\mathrm{in}(C_{H_1}, C_{U_1}, \ldots, C_{H_n}, C_{U_n}), (W_1, \ldots W_n)\,\mathrm{in}(C_{W_1}, \ldots, C_{W_n}),\\ &(C_1, C_2, C_3)\,\mathrm{in}(S_1, S_2, S_3)]\\ &\{H_1, U_1, \ldots H_n, U_n, C_1, C_2, C_3 : e(C_1, vC_2 \prod_{i=1}^{n} w_i^{m_i}) = z \wedge e(u, C_2)e(C_3, w^{-1}) = 1\\ &\wedge \forall i, e(u, H_i) = e(U_i, h) \wedge \forall i, e(u, W_i) = e(w_i, U_i)\}\end{aligned}$$

   The final proof is $\Pi = (\pi, S_1, S_2, S_3, U_1, \cdots, U_n, W_1, \ldots W_n)$ and the resulting commitments are $H_1, \ldots H_n$. $\mathsf{VerifyProof}(params, pk, \Pi, (H_1, \ldots, H_n))$ simply verifies the proof Π.

2. The second protocol is a zero knowledge proof of equality of committed values. See Section 3.5 for details.

3. The third protocol is a secure two-party computation for signing a committed value. We use the same technique as in Section 4.2.2 to reduce computing a signature to computing an arithmetic circuit using the Jarecki and Shmatikov [61] secure two party computation protocol.

Efficiency

The P-signature proof system for a signature scheme which allows n messages generates $3 + 3n$ new commitments ($2 + n$ in G_1 and $1 + 2n$ in G_2) and Groth-Sahai proofs for $2 + 2n$ pairing product equations, 1 with $Q = 1$, and $1 + 2n$ with $Q = 2$. Applying the efficiency formulas given in Section 3.4, we get the following lemma:

Lemma 4. *When instantiated using the SXDH instantiation given in Section 3.3 the P-signature proofs will have the following efficiency: Generating the proof will require* $36 + 36n$ *exponentiations in* G_1 *and* $32 + 40n$ *exponentiations in* G_2*. The resulting proof will consist of* $12 + 10n$ *elements of* G_1 *and* $10 + 12n$ *elements of* G_2*. Verifying the proof will involve computing* $44 + 48n$ *bilinear group pairings.*

When instantiated using the symmetric setting version of the DLIN instantiation given in Section 3.3, the above proof system will have the following efficiency: Generating the proof will require $135 + 153n$ *exponentiations in* G*. The resulting proof will consist of* $27 + 27n$ *elements in* G*. Verifying the proof will involve computing* $81 + 90n$ *bilinear group pairings.*

Security

Theorem 12. *The above construction is a secure P-Signature scheme given the HSDH and TDH assumption and either the SXDH or DLIN assumption.*

The proof follows from the F-unforgeability of the multi-block signature scheme and the security of the Groth-Sahai proofs, which depend on either the SXDH or DLIN assumptions. It is very similar to that given Section 4.2.2, so we omit it here.

4.2.4 P-signatures with Multiple Messages and Small Keys from BB-HSDH and BB-CDH

Note that in what follows, we will describe an authentication scheme, however it can easily be converted to a signature scheme using $pk = h^{sk}$.

Our authentication construction is loosely inspired by the weak Boneh Boyen signature scheme [11], in which the signature under a secret key sk on a message m is computed as $\mathsf{BBsig}(sk, m) = g^{\frac{1}{sk+m}}$. Our scheme is more complicated for several reasons. First, we must have a scheme which allows us to authenticate several messages at once, without increasing the size of the secret key (the secret key space must be a subset of the message space).

Also, we want this scheme to be F-unforgeable, in the sense that it is also hard to produce $F(m)$ and an authenticator on m for an m which has not been signed. In this case our bijection F will be the information which we can extract from a commitment to a message, which will be in the group G_1, while if we emulate the BB scheme, m must be in the exponent space, Z_p. Thus the trivial bijection $F(m) = m$ will not work, and F unforgeability must be a stronger definition than standard unforgeability. Note, also that if $F(m) = g^m$, and if the adversary is given $v = F(sk)$ (as he is in our certification definition), the above authentication scheme is not F unforgeable – for any $f \in Z_p^*$, the value $g^{1/f}$ is a valid BB signature for $F(m) = g^f/v = g^{f-sk}$. Thus, our bijection F must be somewhat more complex.

Finally, the weak BB scheme is only secure for a previously determined polynomial sized message space. We want to be able to sign arbitrary messages, since we must be able to sign any randomly generated secret key. (This could be 'solved' by using an interactive assumption, however, as interactive assumptions are generally considered very strong, we would like to avoid them.)

The authentication scheme is as follows:

$\mathsf{AuthSetup}(1^k)$ generates groups G_1, G_2, G_T of prime order p (where $|p|$ is proportional to k), bilinear map $e : G_1 \times G_2 \to G_T$, and group elements $g, u, u^*, u_1, \ldots, u_n \in G_1$ and $h \in G_2$. It outputs $params_A = (G_1, G_2, G_T, e, p, g, u, u^*, u_1, \ldots, u_n)$. Note that the element u is only needed to define F; it is not used in creating the authenticator.

$\mathsf{AuthKg}(params_A)$ outputs a random $sk \leftarrow Z_p$.

$\mathsf{Auth}(params_A, sk, (m_1, \ldots m_n))$ chooses random $K^*, K_1, \ldots K_n \leftarrow Z_p$. It outputs

$$auth = (g^{\frac{1}{sk+K^*}}, h^{K^*}, u^{*K^*}, \{g^{\frac{1}{K^*+K_i}}, h^{K_i}, u_i^{K_i}, g^{\frac{1}{K_i+m_i}}\}_{1\le i\le n}) \ .$$

$\mathsf{VerifyAuth}(params_A, sk, (m_1, \ldots, m_n), auth)$ Parse $auth$ as $(A^*, B^*, C^*, \{A_i, B_i, C_i, D_i\}_{1 \leq i \leq n})$. Verify that $e(A^*, h^{sk}B^*) = e(g, h)$, and that $e(B^*, u^*) = e(h, C^*)$. For $1 \leq i \leq n$ verify that $e(A_i, B^*B_i) = e(g, h)$, that $e(B_i, u_i) = e(h, C_i)$, and that $e(D_i, B_i h^{m_i}) = e(g, h)$. Accept if and only if all verifications succeed.

Theorem 13. *The message authentication scheme* $(\mathsf{AuthSetup}, \mathsf{AuthKg}, \mathsf{Auth}, \mathsf{VerifyAuth})$ *is F-unforgeable for $F(m_i) = (h^{m_i}, u^{m_i})$ under the BB-HSDH and BB-CDH assumption.*

We defer the proof to section 4.3.4 where we will show that a stronger property holds.

Furthermore, in Section 4.3.3, we will show protocols for issuing and proving knowledge of the authenticator (or signature) which also satisfy stronger properties.

Efficiency

Let us briefly consider the efficiency of the resulting Prove protocol. We use Groth-Sahai commitments to commit to $F(\vec{m}) = (h^{m_1}, u^{m_1}, \ldots, h^{m_n}, u^{m_n})$, resulting in commitments $(S_{H_1}, S_{U_1}, \ldots, S_{H_n}, S_{U_n})$. Next, we commit to the signature $(A^*, B^*, C^*, \{A_i, B_i, C_i, D_i\}_{1 \leq i \leq n})$ to get commitments $(S_{A^*}, S_{B^*}, S_{C^*}, \{S_{A_i}, S_{B_i}, S_{C_i}, S_{D_i}\}_{1 \leq i \leq n})$. Then we form the following witness indistinguishable proof:

$$\begin{aligned}
\pi_1 \leftarrow \mathsf{NIPK}[&(H_1, U_1, \ldots H_n, U_n) \,\mathrm{in}(S_{H_1}, S_{U_1}, \ldots, S_{H_n}, S_{U_n}, A^* \,\mathrm{in}\, S_{A^*}, B^* \,\mathrm{in}\, S_{B^*}, C^* \,\mathrm{in}\, S_{C^*},\\
&(A_1, B_1, C_1, D_1, \ldots A_n, B_n, C_n, D_n) \,\mathrm{in}(S_{A_1}, S_{B_1}, S_{C_1}, S_{D_1}, \ldots S_{A_n}, S_{B_n}, S_{C_n}, S_{D_n})]\\
&\{(\{H_i, U_i\}_{1 \leq i \leq n}, A^*, B^*, C^*, \{A_i, B_i, C_i, D_i\}_{1 \leq i \leq n}) :\\
&e(A^*, pkB^*) = e(g, h) \wedge e(B^*, u^*) = e(h, C^*) \wedge \{e(H_i, u) = e(h, U_i)\}_{1 \leq i \leq n} \wedge\\
&\{e(A_i, B^*B_i) = e(g, h) \wedge e(B_i, u_i) = e(h, C_i) \ \wedge e(D_i, B_i H_i) = e(g, h)\}_{1 \leq i \leq n}\ \}.
\end{aligned}$$

Thus, the P-signature proof system for a signature scheme which allows n messages will generates $3 + 6n$ new commitments ($2 + 4n$ in G_1 and $1 + 2n$ in G_2) and Groth-Sahai proofs for $2 + 4n$ pairing product equations, $1 + 2n$ with $Q = 1$, and $1 + 2n$ with $Q = 2$. Applying the efficiency formulas given in Section 3.4, we get the following lemma:

Lemma 5. *When instantiated using the SXDH instantiation given in Section 3.3 the P-signature proofs will have the following efficiency: Generating the proof will require* $36 + 72n$ *exponentiations in* G_1 *and* $32 + 64n$ *exponentiations in* G_2*. The resulting proof will consist of* $12 + 24n$ *elements of* G_1 *and* $10 + 20n$ *elements of* G_2*. Verifying the proof will involve computing* $44 + 88n$ *bilinear group pairings.*

When instantiated using the symmetric setting version of the DLIN instantiation given in Section 3.3, the above proof system will have the following efficiency: Generating the proof will require $135 + 270n$ *exponentiations in* G*. The resulting proof will consist of* $27 + 54n$ *elements in* G*. Verifying the proof will involve computing* $81 + 162n$ *bilinear group pairings.*

4.3 P-signatures with Extra Properties

In some cases, we will need P-signatures with even stronger properties. Here we describe a stronger notion of unforgeability, then describe the extra properties that we want from the protocols for issuing and proving knowledge of signatures. We show that the construction presented in section 4.2.4 satisfies these properties. In Chapter 7, we show an application of P-signatures which requires these stronger properties.

4.3.1 Certification Unforgeability

A message authentication scheme is similar to a signature scheme. However, there is no public-key, and a user needs to use the *secret key* to verify that a message has been authenticated. We need an authentication scheme for a vector of messages $\vec{m}$. This is reminiscent of signatures for blocks of messages [24].

An authentication scheme in the common parameters model consists of four protocols: $\mathsf{AuthSetup}(1^k)$ outputs common parameters $params_A$. $\mathsf{AuthKg}(params_A)$ outputs a secret key sk. $\mathsf{Auth}(params_A, sk, \vec{m})$ outputs an authentication tag $auth$ that authenticates a vector of messages $\vec{m}$. $\mathsf{VerifyAuth}(params_A, sk, \vec{m}, auth)$ accepts if $auth$ is a proper authenticator for $\vec{m}$ under key sk.

The security properties of an authentication scheme are similar to those of a signature scheme. It must be complete and unforgeable. However, we need to strengthen the unforgeability property to fit our application: We need the authentication scheme to be unforgeable even if the adversary learns a signature on the secret-key. This is because an adversary in a delegatable credentials system might give a user a credential – i.e., sign the user's secret-key. For example, if $\mathsf{Auth}(params_A, sk, m) = \mathsf{Auth}(params_A, m, sk)$, then it is impossible to tell whether this is the adversary's signature on the user's secret-key or the user's signature on the adversary's secret-key. Such a scheme cannot be certification-secure, but might be secure under standard unforgeability definitions.
Formally, an authentication scheme is F-unforgeable and certification secure if for all PPTM adversaries $\mathcal{A}$:

$$\begin{aligned}&\Pr[params_A \leftarrow \mathsf{AuthSetup}(1^k); sk \leftarrow \mathsf{AuthKg}(params_A);\\&\quad(\vec{y}, auth) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Auth}}(params_A, sk, .), \mathcal{O}_{\mathsf{Certify}}(params_A, ., (sk, ., \ldots))}(params_A, F(sk)) :\\&\quad\mathsf{VerifyAuth}(params_A, sk, F^{-1}(\vec{y}), auth) = 1 \wedge F^{-1}(\vec{y}) \notin Q_{\mathsf{Auth}}] \leq \nu(k).\end{aligned}$$

Oracle $\mathcal{O}_{\mathsf{Auth}}(params_A, sk, \vec{m})$**.** Outputs $\mathsf{Auth}(params_A, sk, \vec{m})$ and stores $\vec{m}$ on Q_{Auth}.

Oracle $\mathcal{O}_{\mathsf{Certify}}(params_A, sk^*, (sk, m_2, \ldots, m_n))$**.** Outputs $\mathsf{Auth}(params_A, sk^*, (sk, m_2, \ldots, m_n))$.

4.3.2 Other Protocols

We also require proof protocols and issuing protocols with stronger properties.

First we require that there exist efficient F-extractable commitment scheme Commit, and algorithms $\mathsf{PKSetup}$, $\mathsf{PKProve}$, $\mathsf{PKVerify}$ which form an efficient, randomizable proof of knowledge system for proving knowledge of an authenticator under a committed secret key on a committed message.

Additionally, we require that there be a secure two-party protocol in which an issuer who possesses a secret key and a user who possesses a message can jointly compute a proof of knowledge of a signature under that secret key on that message. Here both parties are given as public input commitments to the secret key and to the message as well as the system parameters. We require a strong security property in which the protocol must be guaranteed secure even if the trapdoor (simulation or extraction) for those parameters is given to both participants.

4.3.3 Construction

Constructing a Certification-secure F-unforgeable Authentication Scheme

We will use the authentication scheme described in section 4.2.4

Zero-Knowledge Proof of Knowledge of an Authenticator

First, we describe the commitment scheme that we will use for the delegatable credentials. A commitment $comm_x = \mathsf{Commit}(x, open_x)$ to an exponent x consists of two GS commitments to group elements such that $\mathsf{Commit}(x, (o_1, o_2)) = (C^{(1)}, C^{(2)}) = (\mathsf{GSCommit}(h^x, o_1), \mathsf{GSCommit}(u^x, o_2))$ and a proof these commitments are correctly related. This can be done using a randomizable $\mathsf{NIPK}_{GS}[h^x \text{ in } \mathsf{Comm}^{(1)}; u^x \text{ in } \mathsf{Comm}^{(2)}]$ $\{(F(x)) : e(h^x, u) = e(h, u^x)\}$ and allows us to extract $F(x)$, where F is as defined in Section 4.2.4. As the GS proof system is randomizable for $\mathsf{GSCommit}$ and for $\mathsf{GSProve}$, it is also randomizable for Commit.

We need to create a zero-knowledge proof of knowledge of an unforgeable authenticator for messages $\vec{m} = (m_1, \ldots, m_n)$, where the first ℓ values $\vec{m_h} = (m_1, \ldots, m_\ell)$ are hidden in commitments $comm_{\vec{m_h}} = (comm_{m_1}, \ldots, comm_{m_\ell})$ and the values $\vec{m_o} = (m_{\ell+1}, \ldots, m_n)$ are publicly known, i.e., a proof

$$\begin{aligned}\pi \leftarrow \mathsf{NIZKPK}&[sk \text{ in } comm_{sk}; \vec{m_h} \text{ in } comm_{\vec{m_h}}]\\ &\{(F(sk), F(\vec{m_h}), auth) : \mathsf{VerifyAuth}(params_A, sk, \vec{m_h}, \vec{m_o}, auth) = 1\}.\end{aligned}$$

To generate this , we first prove knowledge of a valid authenticator with respect to freshly generated commitments $comm'_{sk}$ and $comm'_{\vec{m_h}}$, and then we prove that those commitments are to the same values as $comm_{sk}$ and $comm_{\vec{m_h}}$ respectively. The first is a witness indistinguishable proof of knowledge of a pairing product equation, which we can do using GS proofs (note here we describe the proof in terms of the component GS commitments as well):

$$\begin{gathered}\pi_1 \leftarrow \mathsf{NIPK}_{GS}[h^{sk} \text{ in } comm'^{(1)}_{sk}; u^{sk} \text{ in } comm'^{(2)}_{sk}; h^{m_1} \text{ in } comm'^{(1)}_{m_1}; u^{m_1} \text{ in } comm'^{(2)}_{m_1}; \ldots;\\ h^{m_\ell} \text{ in } comm'^{(1)}_{m_\ell}; u^{m_\ell} \text{ in } comm'^{(2)}_{m_\ell}]\\ \{(h^{sk}, u^{sk}), (\{h^{m_i}, u^{m_i}\}_{1 \le i \le \ell}), (A^*, B^*, C^*, \{A_i, B_i, C_i, D_i\}_{1 \le i \le n})) :\\ e(u, h^{sk}) = e(u^{sk}, h) \wedge e(A^*, h^{sk}B^*) = e(g, h) \wedge e(B^*, u^*) = e(h, C^*) \wedge \{e(h^{m_i}, u) = e(h, u^{m_i})\}_{1 \le i \le \ell} \wedge\\ \{e(A_i, B^*B_i) = e(g, h) \wedge e(B_i, u_i) = e(h, C_i) \ \wedge e(D_i, B_i h^{m_i}) = e(g, h)\}_{1 \le i \le n}\ \}.\end{gathered}$$

Next, we give a zero knowledge proof that these new commitments are to the same values as the originals. We can prove this for two GS commitments C, C' using the proof system given in Section 3.5.

We form proofs $\pi^{(1)}_{sk}$ and $\{\pi^{(1)}_{m_i}\}$ for commitments $C^{(1)}_{sk}, C'^{(1)}_{sk}$, and $\{C^{(1)}_{m_i}, C'^{(1)}_{m_i}\}$, respectively. The final proof of knowledge of the authenticator is $\pi = \pi_1 \circ \pi^{(1)}_{sk} \circ \pi^{(1)}_{m_1} \circ \ldots \circ \pi^{(1)}_{m_\ell}$. (Note that this implies that $C^{(2)}_{sk}, C'^{(2)}_{sk}$, and $\{C^{(2)}_{m_i}, C'^{(2)}_{m_i}\}$ are correct as well.)

Zero Knowledge. To see that this proof system is zero knowledge, consider the following simulator: The simulator first picks sk' and $\vec{m_h}'$ at random and uses them to generate an authentication tag. It uses the authentication tag as a witness for the witness indistinguishable proof and then fakes the proofs that the commitments C'_{sk}, and $C'_{\vec{m_h}}$ are to the same values as the original commitments C_{sk} and $C_{\vec{m_h}}$.

Efficiency. The proof π_1 will require $5 + 6\ell$ additional Groth-Sahai commitments ($3 + 4\ell$ in G_1 and $2 + 2\ell$ in G_2), and Groth-Sahai proofs for $3 + 4\ell$ pairing product equations , $1 + 2\ell$ with $Q = 1$ and $2 + 2\ell$ with $Q = 2$. Then we must also include the $1 + \ell$ zero-knowledge proofs of equality of committed values for elements of G_2 as described in Section 3.5. Applying the efficiency formulas given in Section 3.4 and Section 3.5, we get the following lemma:

Lemma 6. *When instantiated using the SXDH instantiation given in Section 3.3 the P-signature proofs will have the following efficiency: Generating the proof will require* $84+100\ell$ *exponentiations in* G_1 *and* $76+88\ell$ *exponentiations in* G_2*. The resulting proof will consist of* $28 + 34\ell$ *elements of* G_1 *and* $24 + 28\ell$ *elements of* G_2*. Verifying the proof will involve computing* $108 + 128\ell$ *bilinear group pairings.*

When instantiated using the symmetric setting version of the DLIN instantiation given in Section 3.3, the above proof system will have the following efficiency: Generating the proof will require $315 + 369\ell$ *exponentiations in G. The resulting proof will consist of* $63 + 75\ell$ *elements in G. Verifying the proof will involve computing* $198 + 234\ell$ *bilinear group pairings.*

2PC Protocol for Creating a NIZKPK of an Authenticator

An efficient two-party computation protocol for computing a non-interactive zero-knowledge proof of knowledge (NIZKPK) of an authentication tag is a protocol between a user and a issuer. The user's private input is the vector $\vec{m_h}$ of committed messages that are to be authenticated and opening vector $open_{\vec{m_h}}$. The issuer's private input is secret key sk and $open_{sk}$. Both parties agree on the public input: the public parameters $params_{DC}$ for the proof system and the authentication scheme, the commitments $comm_{\vec{m_h}}$, the public messages $\vec{m_o}$, and $comm_{sk}$. The commitments are double commitments as described in Section 4.3.3. We also define a commitment to a vector of messages to be the list of commitments to its elements. The user's output of the protocol is a proof $\pi \leftarrow \mathsf{NIZKPK}[sk \text{ in } comm_{sk}; \vec{m_h} \text{ in } \mathsf{Commit}(\vec{m_h}, \vec{0})]\{(F(sk), F(\vec{m_h}), auth) : \mathsf{VerifyAuth}(params_A, sk, \vec{m_h}, \vec{m_o}, auth) = 1\}$ as described in Section 4.3.3. The issuer only learns about success or failure of the protocol and outputs nothing. If $(\vec{m_h}, open_{\vec{m_h}})$, or $(sk, open_{sk})$ are inconsistent with $comm_{\vec{m_h}}, comm_{sk}$ respectively, the functionality reports failure; otherwise it returns a correctly formed random proof π.

1. The user proves that she knows $\vec{m_h}$ and $open_{\vec{m_h}}$ for $comm_{\vec{m_h}}$. The issuer aborts if the proof fails.
2. The issuer does a proof of knowledge of sk and $open_{sk}$ for $comm_{sk}$. The user aborts if the proof fails.
3. The issuer chooses random $K^*, K_1, \ldots K_n \leftarrow Z_p$. He computes a partial authentication tag $auth' = (g^{\frac{1}{sk+K^*}}, h^{K^*}, u^{*K^*}, \{g^{\frac{1}{K^*+K_i}}, h^{K_i}, u_i^{K_i}\}_{1 \le i \le n})$.
4. Then the issuer computes a fresh commitment $comm'_{sk}$ to sk and creates a NIZKPK π' for a partial authenticator $auth'$:

$$\begin{aligned}\pi' \leftarrow \mathsf{NIZKPK}_{GS}[&h^{sk} \text{ in } comm'^{(1)}_{sk}; u^{sk} \text{ in } comm'^{(2)}_{sk}; B_i \text{ in } \mathsf{GSCommit}(B_i, 0)]\\ &\{(F(sk), (A^*, B^*, C^*, \{A_i, B_i, C_i\}_{1\le i\le n}, \{D_i\}_{\ell+1\le i\le n}), h^\theta) :\\ &e(u^{sk}/u^{sk'}, h^\theta) = 1 \wedge e(g, h^\theta) = e(g, h) \wedge e(h^{sk}, u) = e(h, u^{sk})\wedge\\ &e(A^*, h^{sk}B^*) = e(g, h) \wedge e(B^*, u^*) = e(h, C^*)\wedge\\ &\{e(A_i, B^*B_i) = e(g, h) \wedge e(B_i, u_i) = e(h, C_i)\}_{1\le i\le n}\wedge\\ &\{e(D_i, B_i h^{m_i}) = e(g, h)\}_{\ell+1\le i\le n}\}.\end{aligned}$$

Note that all authenticator values use random commitments, except for the B_i values, which are committed using 0 openings. This means that $B_i = h^{K_i}$ can be learned from the proof.

5. The issuer sends π' to the user.

6. The user checks the proof and aborts if the verification fails. Otherwise, she runs ℓ instances of an efficient two-party computation protocol with the issuer. On public input $comm_{m_i}$ and secret input m_i by the user and K_i by the issuer ($1 \le i \le \ell$) the protocol computes the missing $D_i = g^{\frac{1}{K_i+m_i}}$. The output is obtained only by the user. We give an efficient implementation of this 2PC protocol using additively homomorphic encryption in Appendix 4.3.3.

7. The user checks that the D_i were computed for the correct K_i using the B_i from the proof π', computes commitments $comm^{(1)}_{m_i} = \mathsf{GSCommit}(h^{m_i}, 0)$, $comm^{(2)}_{m_i} = \mathsf{GSCommit}(u^{m_i}, 0)$ and fresh commitments $comm'^{(1)}_{m_i}$, $comm'^{(2)}_{m_i}$ for each m_i and computes the proof:

$$\begin{aligned}\pi'' \leftarrow \mathsf{NIPK}_{GS}[&h^{m_1} \text{ in } comm'^{(1)}_{m_1}; u^{m_1} \text{ in } comm'^{(2)}_{m_1}; \ldots; h^{m_\ell} \text{ in } comm'^{(1)}_{m_\ell}; u^{m_\ell} \text{ in } comm'^{(2)}_{m_\ell});\\ &B_i \text{ in } \mathsf{GSCommit}(B_i, 0)]\\ &\{(F(\vec{m_h}) = (\{F(m_i)\}_{1\le i\le\ell}), \{B_i, D_i\}_{1\le i\le\ell}) :\\ &\{e(D_i, B_i h^{m_i}) = e(g, h) \wedge e(h^{m_i}, u) = e(h, u^{m_i})\}_{1\le i\le\ell}\},\end{aligned}$$

as well as the zero-knowledge proofs of equality of committed values in $comm^{(1)}_{m_i}$ and $comm'^{(1)}_{m_i}$, $\pi_{m_1}, \ldots \pi_{m_\ell}$.

8. Finally the user computes $\pi = \pi' \circ \pi'' \circ \pi_{m_1} \circ \ldots \circ \pi_{m_\ell}$ and randomizes the combined proof with opening values $open_{sk} = 0$ and $open_{\vec{m_h}} = 0$, and all other openings chosen at random. The resulting proof π' is the user's output.

A Protocol for securely computing $g^{1/(sk+m)}$

We have a user whose input to the protocol is a secret m and the issuer whose input is a secret sk. Both parties have common input $comm_m$. As a result of the protocol, the user obtains the value $g^{1/(sk+m)}$, and the issuer obtains no information about m.

We will first describe the protocol for any homomorphic encryption scheme and then instantiate it with the Paillier encryption scheme (for the latter we refer to Appendix 4.3.4).

Let $\mathsf{Keygen}, \mathsf{Enc}, \mathsf{Dec}$ be an additively homomorphic semantically secure encryption scheme, let "$\oplus$" denote the homomorphic operation on ciphertexts; for e a ciphertext and r an integer, $e \otimes r$ denotes "adding" e to itself r times.

Let p be the size of the bilinear groups G_1, G_2, G_T given in $params_A$.

Now, to issue a signature, the user and the issuer run the following protocol:

1. The issuer generates $(sk_{hom}, pk_{hom}) \leftarrow \mathsf{Keygen}(1^k)$ in such a way that the message space is of size at least $2^k p^2$. He then computes $e_1 = \mathsf{Enc}(pk_{hom}, sk)$ and sends e_1, pk_{hom} to the user and engages with her in a proof that e_1 encrypts to a message in $[0, p]$.

2. The user chooses $r_1 \leftarrow Z_p$ and $r_2 \leftarrow \{0, \dots 2^k p\}$. The user then computes $e_2 = ((e_1 \oplus \mathsf{Enc}(pk_{hom}, m)) \otimes r_1) \oplus \mathsf{Enc}(pk_{hom}, r_2 p)$ and sends e_2 to the user.

3. The issuer and the user perform an interactive zero knowledge proof in which the user shows that e_2 has been computed correctly using the message in $comm_m$, and that r_1, r_2 are in the appropriate ranges.

4. The issuer decrypts $x = \mathsf{Dec}(sk_{hom}, e_2)$, computes $\sigma^* = g^{1/x}$ and sends it to the user.

5. The user computes $\sigma = {\sigma^*}^{r_1}$ and verifies that it is a correct weak BB signature on m.

Remarks. The above protocol can be realized efficiently when using the Paillier homomorphic encryption scheme [68]. In this case we can employ the well known discrete logarithm related protocols to prove all the statements the parties have to prove to each other, as is done, e.g., in [27]. Also note that as the issue protocol is interactive, this works without having to resort to the random oracle model to prove security. See Appendix 4.3.4.

4.3.4 Proof of Security

Proof of Certification F-unforgeability of Authentication Scheme

Suppose we have an adversary which, can break the certification F-unforgeability property, by outputting a forgery of the form:

$$(F(m_1), \dots, F(m_n)), (A^*, B^*, C^*, \{A_i, B_i, C_i, D_i\}_{1 \le i \le n})$$

If this is a valid forgery, then the verification equations hold, which implies that $\exists b^*, b_1, \dots b_n$ such that:

$$A^* = g^{\frac{1}{sk+b^*}}, B^* = h^{b^*}, C^* = {u^*}^{b^*}$$

and for all $i \in 1, \dots n$,

$$A_i = g^{\frac{1}{b^*+b_i}}, B_i = h^{b_i}, C_i = u_i^{b_i}, D_i = g^{\frac{1}{b_i+m_i}}$$

.

We consider the following cases:

Case 1:

Suppose that with nonnegligible probability, the adversary produces a forgery with $b^* = K^*$, where K^* was used in the response to a previous authentication query.

We divide this into three subcases:

Case 1a: Suppose that with nonnegligible probability, the adversary produces a forgery with $b^* = K^*$, where $K^*, K_1, \ldots, K_n$ were used together in the response to a previous authentication query, but there exists i such that for all $j \in \{1, \ldots, n\}$, $b_i \neq K_j$.

In this case we will show that the New SDH assumption with $q = n + 1$ does not hold.

Our reduction proceeds as follows: We are given the groups G_1, G_2, G_T of order p with bilinear map e, group elements $g, X_1 = g^x, v \in G_1$ and $h, X_2 = h^x \in G_2$ and the pairs $\{T_i = g^{\frac{1}{sk+c_\ell}}, c_\ell\}_{1 \leq \ell \leq n+1}$. We need to produce a new tuple $g^{\frac{1}{sk+c}}, h^c, v^c$.

We first make a guess i^* for which b_i will satisfy the above condition.

We will give the adversary parameters for the authentication scheme as follows: we choose random $z \leftarrow Z_p$, random $u \leftarrow G_1$, and random $u_j \leftarrow G_1$ for all $j \neq i^*$, and we set $u_{i^*} = v$ and $u^* = g^z$. $params = (G_1, G_2, G_T, e, p, g, h, u, u^*, u_1, \ldots u_n)$.

Next, we set secret key $sk = c_{n+1}$ and send $F(sk) = h^{sk}, u^{sk}$ to the adversary.

Then we must answer the adversary's signing and certification queries.

First, we make a guess λ for which query's K^* the adversary will attempt to reuse. Now, for signing queries other than query λ, the adversary sends message set $(m_1, \ldots m_n)$. We compute $\mathsf{Auth}(params,\ sk,\ (m_1, \ldots m_n))$ and send the result to the adversary.

For the λth signing query, we will implicitly set $K^* = x$. We set $K_j = c$ for all $j \in \{1, \ldots, n\}$. Then we can compute the authentication: $\hat{A}^* = T_{n+1}, \hat{B}^* = X_2, \hat{C}^* = X_1^z$, $\{\hat{A}_j = T_j, \hat{B}_j = h^{c_j}, \hat{C}_j = u_j^{c_j}, \hat{D}_j = g^{\frac{1}{c_j+m_j}}\}_{1 \leq j \leq n}$.

For certification queries, the adversary sends key sk_A, and messages $m_2, \ldots m_n$. We compute and return $\mathsf{Auth}(params, sk_A, (sk, m_2, \ldots m_n))$.

When the adversary produces a forgery, with non-negligible probability it will be of the form

$$F(m_1), \ldots, F(m_n), A* = g^{\frac{1}{sk+b^*}}, B^* = h^{b^*}, C^* = u^{*b^*},$$
$$\{A_i = g^{\frac{1}{b^*+b_i}}, B_i = h^{b_i}, C_i = u_i^{b_i}, D_i = g^{\frac{1}{b_i+m_i}}\}_{1 \leq i \leq n} \ .$$

with $b^* = \hat{K}^*$, where $\hat{K}^*, \hat{K}_1, \ldots \hat{K}_n$ were used together in the response to previous authentication query, but there exists i such that for all j, $b_i \neq \hat{K}_j$.

With nonnegligible probability, we will have correctly guessed λ such that $b^* = K^*$ for the K^* returned in the λth authentication query, and correctly guessed i^*. That means $b^* = x$, and A_i, B_i, C_i is a new HSDH triple.

Case 1b: Suppose that with nonnegligible probability, the adversary produces a forgery with $b* = K^*$, where $K^*, K_1, \ldots, K_n$ were used together in the response to a previous authentication query, but there exists $i, j, i \neq j$ such that $b_i = K_j$.

In this case we will show that the New CDH assumption with $q = 2$ does not hold.

Our reduction proceeds as follows: We are given the groups G_1, G_2, G_T of order p with bilinear map e, group elements $g, X_1 = g^x, Y = g^y \in G_1$ and $h, X_2 = h^x \in G_2$, integers $c_1, c_2 \in Z_p$, and values $T_1 = g^{\frac{1}{sk+c_1}}$ and $T_2 = g^{\frac{1}{sk+c_2}}$. We need to produce the value g^{xy}.

We first make a guesses j^* and i^* for which K_j and b_i will satisfy the above condition.

We will give the adversary parameters for the authentication scheme as follows: we choose random $z \leftarrow Z_p$, random $u, u^* \leftarrow G_1$ and random $u_j \leftarrow G_1$ for all $j \neq j^*, j \neq i^*$, and we set $u_{i^*} = Y$ and $u_{j^*} = g^z$. $params = (G_1, G_2, G_T, e, p, g, h, u^*, u_1, \dots u_n)$.

Next, we choose a random key sk and send $F(sk) = h^{sk}, u^{sk}$ to the adversary.

Then we must answer the adversary's signing and certification queries.

First, we make a guess λ for which query's K^* the adversary will attempt to reuse.

Now, for signing queries other than query λ, the adversary sends message set $(m_1, \dots m_n)$. We next compute $\mathsf{Auth}(params,\ sk, (m_1, \dots m_n))$ and send the result to the adversary.

Let $(m_1, \dots m_n)$ be the message set that the adversary gives in the λth signing query. We will set $K^* = c_2 - c_1 + m_{j^*}$ and implicitly set $K_{j^*} = x + c_1 - m_{j^*}$. (Thus, $K^* + K_{j^*} = x + c_2$ and $K_{j^*} + m_{j^*} = x + c_1$.)

We randomly choose $K_j = z_j$ for all $j \in \{1, \dots, n\}$. Then we can compute the authentication as follows: $\hat{A}^* = g^{\frac{1}{sk+c_2-c_1+m_{j^*}}}, \hat{B}^* = h^{sk}, \hat{C}^* = u^{*sk}$, $\{\hat{A}_j = g^{\frac{1}{c_2-c_1+m_{j^*}+z_j}}, \hat{B}_j = h^{z_j}, \hat{C}_j = u_j^{z_j}, \hat{D}_j = g^{\frac{1}{z_j+m_j}}\}_{1 \leq j \leq n, j \neq j^*}$, $\hat{A}_{j^*} = T_2, \hat{B}_{j^*} = X_2 h^{c_1 - m_{j^*}}, \hat{C}_{j^*} = X_1^z u_j^{c_1 - m_{j^*}}, \hat{D}_{j^*} = T_1$

For certification queries, the adversary sends key sk_A, and messages $m_2, \dots m_n$. We compute and return $\mathsf{Auth}(params, sk_A, (sk, m_2, \dots m_n))$.

When the adversary produces a forgery, with non-negligible probability it will be of the form

$$F(m_1), \dots, F(m_n)$$
$$A* = g^{\frac{1}{sk+b^*}}, B^* = h^{b^*}, C^* = u^{*b^*}, \{A_i = g^{\frac{1}{b^*+b_i}}, B_i = h^{b_i}, C_i = u_i^{b_i}, D_i = g^{\frac{1}{b_i+m_i}}\}_{1 \leq i \leq n}$$

.

with $b* = K^*$, where $K^*, K_1, \dots, K_n$ were used together in the response to a previous authentication query, but there exists $i, j, i \neq j$ such that $b_i = K_j$.

With non-negligible probability, we will have correctly guessed λ such that $b^* = K^*$ for the K^* returned in the λth authentication query, and correctly guessed i^*, j^*. Let M_{j^*} be the message that was signed by K_{j^*} in the λth authentication query.

That means $b^* = K^*$, and $b_{i^*} = K_{j^*}$ from the λth triple, so $b_i^* = x + c_1 - M_{j^*}$, which means we have $C_{i^*} = u_{i^*}^{x+c_1-M_{j^*}} = Y^{x+c_1-M_{j^*}}$. Finally, we compute and return $C_{i^*}/Y^{c_1-M_{j^*}} = Y^x = g^{xy}$.

Case 1c: Suppose that with non-negligible probability, the adversary produces a forgery with $b* = K^*$ where $K^*, K_1, \dots K_n$ were used together in the response to a previous authentication query, and for all $i \in \{1, \dots, n\}$, $b_i = K_i$. Suppose keys $K^*, K_1, \dots K_n$ were originally used to sign message set $(M_1, \dots M_n)$. Then since this is a forgery, there must exist at least one index j such that $m_j \neq M_j$.

We break this into two further cases:

Case 1ci: The above forgery is such that $m_j \neq K^*$.

In this case we will show that the New SDH assumption with $q = 2$ does not hold.

Our reduction proceeds as follows: We are given the groups G_1, G_2, G_T of order p with bilinear map e, group elements $g, X_1 = g^x, v \in G_1$ and $h, X_2 = h^x \in G_2$ and the pairs $\{T_\ell = g^{\frac{1}{sk+c_\ell}}, c_\ell\}_{\ell=1,2}$. We need to produce a new tuple $g^{\frac{1}{sk+c}}, h^c, v^c$.

We first make a guess j^* for which m_j will satisfy the above condition.

We will give the adversary parameters for the authentication scheme as follows: we choose random $z \leftarrow Z_p$, random $u^* \leftarrow G_1$, and random $u_j \leftarrow G_1$ for all $j \neq j^*$, and we set $u_{j^*} = g^z$ and $u = v$. $params = (G_1, G_2, G_T, e, p, g, h, u, u^*, u_1, \ldots u_n)$.

Next, we randomly choose secret key $sk \leftarrow Z_q$ and send $F(sk) = h^{sk}, u^{sk}$ to the adversary.

Then we must answer the adversary's signing and certification queries.

First, we make a guess λ for which query's K^* the adversary will attempt to reuse.

Now, for signing queries other than query λ, the adversary sends message set $(m_1, \ldots m_n)$. We compute $\mathsf{Auth}(params,\ sk, (m_1, \ldots m_n))$ and send the result to the adversary.

For the λth signing query, we will implicitly set $K_{j^*} = x + c_1 - m_{j^*}$. We set $K^* = c_2 - c_1 + m_{j^*}$ and randomly choose choose $K_j \leftarrow Z_p$ for all $j \in \{1, \ldots, n\}, j \neq j^*$. Then we can compute the authentication: $\hat{A}^* = g^{\frac{1}{sk+c_2-c_1+m_{j^*}}}, \hat{B}^* = h^{c_2-c_1+m_{j^*}}, \hat{C}^* = u^{*c_2-c_1+m_{j^*}}$, $\{\hat{A}_j = g^{\frac{1}{c_2-c_1+m_{j^*}+K_j}}, \hat{B}_j = h^{K_j}, \hat{C}_j = u_j^{K_j}, \hat{D}_j = g^{\frac{1}{K_j+m_j}}\}_{1\leq j\leq n, j\neq j^*}$, $\hat{A}_{j^*} = Y_2, \hat{B}_{j^*} = X_2 h^{c_1-m_{j^*}}, \hat{C}_{j^*} = X_1^z u_{j^*}^{c_1-m_{j^*}}, \hat{D}_{j^*} = Y_1$.

For certification queries, the adversary sends key sk_A, and messages $m_2, \ldots m_n$. We compute and return $\mathsf{Auth}(params, sk_A, (sk, m_2, \ldots m_n))$.

When the adversary produces a forgery, with non-negligible probability it will be of the form

$$F(m_1), \ldots, F(m_n)$$

$$A* = g^{\frac{1}{sk+b^*}}, B^* = h^{b^*}, C^* = u^{*b^*}, \{A_i = g^{\frac{1}{b^*+b_i}}, B_i = h^{b_i}, C_i = u_i^{b_i}, D_i = g^{\frac{1}{b_i+m_i}}\}_{1\leq i\leq n}$$

.

with $b^* = \hat{K}^*$, where $\hat{K}^*, \hat{K}_1, \ldots \hat{K}_n$ were used together in the response to previous authentication query to sign message set $M_1, \ldots M_n$, and for all i, $b_i = \hat{K}_i$, and there exists j such that $m_j \neq M_j$ and $m_j \neq \hat{K}^*$.

With non-negligible probability, we will have correctly guessed λ such that $b^* = K^*$ for the K^* returned in the λth authentication query, and correctly guessed j^*. That means $b_{j^*} = x + c_1 - m_{j^*}$, and $D_{j^*} = g^{\frac{1}{b_{j^*}+m_{j^*}}} = g^{\frac{1}{x+c_1-M_{j^*}+m_{j^*}}}$. Since we have said that $m_{j^*} \neq M_{j^*}$ and that $m_{j^*} \neq K^* = c_2 - c_1 + M_{j^*}$, we are guaranteed that $D_{j^*}, h^{c_1-M_{j^*}+m_{j^*}}, u_{j^*}^{c_1-M_{j^*}+m_{j^*}}$ is a new HSDH triple.

Case 1cii: The above forgery is such that $m_j = K^*$.

In this case we will show that the New CDH assumption does not hold for $q = n + 1$.

Our reduction proceeds as follows: We are given the groups G_1, G_2, G_T of order p with bilinear map e, group elements $g, X_1 = g^x, Y = g^y \in G_1$ and $h, X_2 = h^x \in G_2$, and $n+1$ pairs $\{c_\ell, T_1 = g^{\frac{1}{sk+c_\ell}}\}_{1\leq \ell\leq n+1}$. We need to produce the value g^{xy}.

We first make a guess j^* for which m_j will satisfy the above condition.

We will give the adversary parameters for the authentication scheme as follows: we choose random $z \leftarrow Z_p$, random $u \leftarrow G_1$ and random $u_j \leftarrow G_1$ for all $j \in \{1, \ldots, n\}$, and we set $u^* = X_1^z$ and $u = Y$. $params = (G_1, G_2, G_T, e, p, g, h, u, u^*, u_1, \ldots u_n)$.

Next, we set secret key $sk = c_{n+1}$ and send $F(sk) = h^{sk}, u^{sk}$ to the adversary.

Then we must answer the adversary's signing and certification queries.

First, we make a guess λ for which query's K^* the adversary will attempt to reuse.

Now, for signing queries other than query λ, the adversary sends message set $(m_1, \ldots m_n)$. We compute $\mathsf{Auth}(params,\ sk, (m_1, \ldots m_n))$ and send the result to the adversary.

Let $(m_1, \ldots m_n)$ be the message set that the adversary gives in the λth signing query, we will set $K_j = c_j$ for $j \in \{1, \ldots, n\}$ and implicitly set $K^* = x$.

Then we can compute the authentication: $\hat{A}^* = T_{n+1}, \hat{B}^* = X_2, \hat{C}^* = X_1^z, \{\hat{A}_j = T_j, \hat{B}_j = h^{c_j}, \hat{C}_j = u_j^{c_j}, \hat{D}_j = g^{\frac{1}{c_j+m_j}}\}_{1 \le j \le n}$.

For certification queries, the adversary sends key sk_A, and messages $m_2, \ldots m_n$. We compute and return $\mathsf{Auth}(params, sk_A, (sk, m_2, \ldots m_n))$.

When the adversary produces a forgery, with non-negligible probability it will be of the form

$$F(m_1) = (h^{m_1}, u^{m_1}), \ldots, F(m_n) = (h^{m_n}, u^{m_n})$$
$$A* = g^{\frac{1}{sk+b^*}}, B^* = h^{b^*}, C^* = u^{*b^*}, \{A_i = g^{\frac{1}{b^*+b_i}}, B_i = h^{b_i}, C_i = u_i^{b_i}, D_i = g^{\frac{1}{b_i+m_i}}\}_{1 \le i \le n}$$

.

with $b^* = \hat{K}^*$, where $\hat{K}^*, \hat{K}_1, \ldots \hat{K}_n$ were used together in the response to previous authentication query to sign message set $M_1, \ldots M_n$, and for all i, $b_i = \hat{K}_i$, and there exists j such that $m_j = \hat{K}^*$.

With non-negligible probability, we will have correctly guessed λ such that $b^* = K^*$ for the K^* returned in the λth authentication query, and correctly guessed i^*, j^*.

That means $m_j = K^* = x$, and so we can simply return $u^{m_j} = u^x = Y^x = g^{xy}$.

Case 2:

Suppose that with non-negligible probability, the adversary produces a forgery with $b* = K_i$, where K_i was returned by a previous certification query.

In this case we will show that the New CDH assumption does not hold for $q = 2$.

Our reduction proceeds as follows: We are given the groups G_1, G_2, G_T of order p with bilinear map e, group elements $g, X_1 = g^x, Y = g^y \in G_1$ and $h, X_2 = h^x \in G_2$, and $n+1$ pairs $\{c_\ell, T_1 = g^{\frac{1}{sk+c_\ell}}\}_{1 \le \ell \le n+1}$. We need to produce the value g^{xy}.

We first make a guess i^* for which K_i will satisfy the above condition.

We will give the adversary parameters for the authentication scheme as follows: we choose random $z \leftarrow Z_p$, random $u \leftarrow G_1$ and random $u_j \leftarrow G_1$ for all $j \in \{1, \ldots, n\}, j \neq i^*$, and we set $u^* = Y$ and $u_{i^*} = X^z$. $params = (G_1, G_2, G_T, e, p, g, h, u, u^*, u_1, \ldots u_n)$.

Next, we set secret key $sk = c_2$ and send $F(sk) = h^{sk}, u^{sk}$ to the adversary.

Then we must answer the adversary's signing and certification queries.

First, we make a guess λ for which certification query's K_{i^*} the adversary will attempt to reuse.

Now, for signing queries the adversary sends message set $(m_1, \ldots m_n)$. We compute $\mathsf{Auth}(params,\ sk, (m_1, \ldots m_n))$ and send the result to the adversary.

For certification queries other than λ, the adversary sends key sk_A, and messages $m_2, \dots m_n$. We compute and return $\mathsf{Auth}(params, sk_A, (sk, m_2, \dots m_n))$.

Let sk_A, $(m_2, \dots m_n)$ be the secret key and message set that the adversary gives in the λth certification query. We will implicitly set $K_{i^*} = x$, and we will set $K^* = c_1$. We randomly choose $K_j \leftarrow Z_p$ for $j \in \{1, \dots, n\}, j \neq i^*$.

Then we can compute the certification: $\hat{A}^* = g^{\frac{1}{sk_A + c_1}}, \hat{B}^* = h^{c_1}, \hat{C}^* = u^{*c_1}$, $\{\hat{A}_j = g^{\frac{1}{c_1 + K_j}}, \hat{B}_j = h^{K_j}, \hat{C}_j = u_j^{K_j}, \hat{D}_j = g^{\frac{1}{K_j + m_j}}\}_{1 \le j \le n, j \neq i^*}$, $\hat{A}_{i^*} = T_1, \hat{B}_{i^*} = X_2, \hat{C}_{i*} = X_1^z, \hat{D}_{i^*} = T_2$.

When the adversary produces a forgery, with non-negligible probability it will be of the form

$$F(m_1), \dots, F(m_n)$$

$$A* = g^{\frac{1}{sk + b^*}}, B^* = h^{b^*}, C^* = u^{*b^*}, \{A_i = g^{\frac{1}{b^* + b_i}}, B_i = h^{b_i}, C_i = u_i^{b_i}, D_i = g^{\frac{1}{b_i + m_i}}\}_{1 \le i \le n}$$

.

with $b* = K_i$, where K_i was returned by a previous certification query.

With non-negligible probability, we will have correctly guessed λ such that $b^* = K_i$ for the K_i returned in the λth certification query, and correctly guessed i^*.

That means $b^* = K_{i^*} = x$, and so we can simply return $B^* = u^{*x} = Y^x = g^{xy}$.

Case 3:

Suppose that with non-negligible probability, the adversary produces a forgery where $b^* =\neq K^*$ for any K^* returned by a previous authentication query, and where $b^* =\neq K_i$ for any K_i returned by a previous certification query.

Let q be an upper bound on the number of authentication or certification queries that the adversary makes. In this case we will show that the New SDH assumption does not hold for this q.

Our reduction proceeds as follows: We are given the groups G_1, G_2, G_T of order p with bilinear map e, group elements $g, X_1 = g^x, v \in G_1$ and $h, X_2 = h^x \in G_2$ and the pairs $\{T_i = g^{\frac{1}{sk + c_\ell}}, c_\ell\}_{1 \le \ell \le q}$. We need to produce a new tuple $g^{\frac{1}{sk+c}}, h^c, v^c$.

We will give the adversary parameters for the authentication scheme as follows: we choose random $z \leftarrow Z_p$, and random $u_j \leftarrow G_p$, and we set $u^* = v$ and $u = g^z$. $params = (G_1, G_2, G_T, e, p, g, h, u, u^*, u_1, \dots, u_n)$.

We implicitly secret key $sk = x$. We send $F(sk) = X_2, X_1^z$ to the adversary.

Then we must answer the adversary's signing and certification queries.

For the γth query:

If it is an authentication query: The adversary sends message set $(m_1, \dots m_n)$. We will set $K^* = c_\gamma$ We choose random $K_j \leftarrow Z_p$ for all $j \in \{1, \dots, n\}$. Then we can compute the authentication: $\hat{A}^* = T_i, \hat{B}^* = h^{c_\gamma}, \hat{C}^* = u^{*c_\gamma}$, $\{\hat{A}_j = g^{\frac{1}{c_\gamma + K_j}}, \hat{B}_j = h^{K_j}, \hat{C}_j = u_j^{K_j}, \hat{D}_j = g^{\frac{1}{K_j + m_j}}\}_{1 \le j \le n}$.

If it is a certification query: The adversary sends key sk_A, and messages $m_2, \dots m_n$. We will set $K_1 = c_\gamma$ We choose random $K_j \leftarrow Z_p$ for all $j \in \{1, \dots, n\}, j \neq 1$, and random $K^* \leftarrow Z_p$. Then we can compute the authentication: $\hat{A}^* = g^{\frac{1}{sk_A + K^*}}, \hat{B}^* = h^{K^*}, \hat{C}^* = u^{*K^*}$, $\{\hat{A}_j = g^{\frac{1}{K^* + K_j}}, \hat{B}_j = h^{K_j}, \hat{C}_j = u_j^{K_j}, \hat{D}_j = g^{\frac{1}{K_j + m_j}}\}_{2 \le j \le n}$, $\hat{A}_1 = g^{\frac{1}{K^* + c_\gamma}}, \hat{B}_1 = h^{c_\gamma}, \hat{C}_1 = u_1^{c_\gamma}, \hat{D}_1 = T_\gamma$.

When the adversary produces a forgery, with non-negligible probability it will be of the form

$$F(m_1), \ldots, F(m_n)$$

$$A* = g^{\frac{1}{sk+b*}}, B^* = h^{b^*}, C^* = u^{*b^*}, \{A_i = g^{\frac{1}{b^*+b_i}}, B_i = h^{b_i}, C_i = u_i^{b_i}, D_i = g^{\frac{1}{b_i+m_i}}\}_{i=1\ldots n}$$

.

where $b^* =\neq K^*$ for any K^* returned by a previous authentication query, and where $b^* =\neq K_i$ for any K_i returned by a previous certification query.

That means for all $\ell \in \{1, \ldots, q\}$, $b^* \neq c_\ell$, so A^*, B^*, C^* is a new HSDH triple.

Proof of Security for the Two-Party Protocol for Authentication Proofs

Theorem 14. *The above construction is a secure two-party computation for the parameters and the trapdoor generated by* ExtSetup *and* SimSetup *if the underlying proof systems are secure.*

Proof sketch. We need to do a simulation for both ExtSetup and SimSetup.
ExtSetup: A simulator that simulates a dishonest user $\tilde{\mathcal{U}}$ proceeds as follows:

- The simulator runs $\tilde{\mathcal{U}}$ as a blackbox. He uses the extractor of the proof of knowledge to obtain $\vec{m}$ and $open_{\vec{m}}$. If they are consistent with $comm_{\vec{m}}$ he sends them to the ideal functionality. He aborts otherwise.

- Next, the simulator uses the zero-knowledge simulator to simulate his own proof of knowledge.

  The ideal functionality returns a NIZKPK π of an authenticator.

- The simulator uses the trapdoor to extract the authenticator. He uses a subset of the values to build proof π'.

- Next the simulator uses the simulator of the 2PC subprotocol for weak BB signatures to finish the simulation. Note that an honest issuer never outputs anything, even in case of protocol failure.

- The simulator outputs whatever $\tilde{\mathcal{U}}$ outputs.

As the simulation of the environment for $\tilde{\mathcal{U}}$ up to the 2PC subprotocol is perfect, $\tilde{\mathcal{U}}$ will behave exactly in the same way as in the real world. The simulatability of the 2PC subprotocol completes the proof.

A simulator that simulates a dishonest issuer $\tilde{\mathcal{I}}$ precedes as follows:

- The simulator runs $\tilde{\mathcal{I}}$ as a black box. The simulator uses the zero-knowledge simulator to simulate the proof of knowledge.

- The simulator uses the extractor of the proof of knowledge. If they are inconsistent with $comm_{\vec{m}}$, he aborts.

- Now the malicious issuer provides additional values, that the simulator checks in the same way as the user would.

- Then it runs the simulator of the 2PC with the issuer. If the checks or the 2PC fail, he sends $\perp$ to the ideal functionality to force the ideal user to abort. If they pass, he sends $\vec{m}$ and $open_{\vec{m}}$ to the ideal functionality.
- The simulator outputs whatever $\tilde{\mathcal{I}}$ outputs.

The simulator aborts, when an honest user would abort, and both the ideal world user as well as the real world user output a random proof. This completes the proof for the extraction parameters.
SimSetup: A simulator that simulates a dishonest user $\tilde{\mathcal{U}}$ proceeds as follows:

- The simulator runs $\tilde{\mathcal{U}}$ as a blackbox. He uses the extractor of the proof of knowledge to obtain $\vec{m}$ and $open_{\vec{m}}$. If the values are consistent with $comm_{\vec{m}}$, he sends $\vec{m}$ and $open_{\vec{m}}$ to the ideal functionality; otherwise he aborts.
- Next, the simulator uses the zero-knowledge simulator to simulate his own proof of knowledge.

  The ideal functionality returns a NIZKPK π of an authenticator.
- Now, however, we cannot use the returned proof π for the simulation. Luckily, the simulation trapdoor allows us to fake the proof of equality between the commitments $comm_{sk}$ and $comm'_{sk}$, and the simulator can use a new random key sk' to compute the partial authenticator $auth'$ and the corresponding proof.
- The rest of the proof proceeds as for ExtSetup.

The simulation for the dishonest issuer $\tilde{\mathcal{I}}$ is the same as for ExtSetup.

Proof of Security for 2PC for computing WBB signature

Proof. We first simulate a malicious user. Recall that we must define a simulator $\mathcal{S}$ which gets as input parameters $params$, commitment $comm$ to the message to be signed, and a signature σ on that message under secret key sk from the ideal functionality, and must impersonate an honest issuer without knowing sk. Consider the following simulator:

1. $\mathcal{S}$ honestly generates a key pair $(sk_{hom}, pk_{hom}) \leftarrow \mathsf{Keygen}(1^k)$. It computes $e_1 = \mathsf{Enc}(pk_{hom}, 0)$. It sends pk_{hom}, e_1 to the adversary.
2. $\mathcal{S}$ receives e_2 from the adversary.
3. $\mathcal{S}$ acts as the verifier for the proof that e_2 was computed correctly. He runs the proof of knowledge extraction algorithm and extracts r_1 (note that this might include rewinding the adversary, but not farther than the beginning of this step). Finally it computes $\sigma^* = \sigma^{1/r_1}$ ands sends it to the adversary.

Now we show that this $\mathcal{S}$ does a successful simulation: Consider the following series of games:

- In the first game, $sk, pk, m, open, state, comm$ are generated as in the definition of the protocol, and then the adversary $\mathcal{A}_2(state)$ interacts with the real world party as defined above.

- In the second game, $sk, pk, m, open, state, comm$ are generated the same way, but now $\mathcal{A}_2(state)$ interacts with a $\mathcal{S}'$, which behaves as the real protocol for steps 1 and 2, but then behaves as $\mathcal{S}$ for step 3. The only difference then is that this simulator extracts r_1 from the proof, and uses r_1 and σ to form σ^*. Note that if the proof is sound, then this σ^* will be identical to that produced in the previous game. Thus this is indistinguishable from the previous game by the extraction property of the ZK proof system.

- In the last game, $sk, pk, m, open, state, comm$ are generated the same way, and then $\mathcal{A}_2(state)$ interacts with $\mathcal{S}$. This differs from the second game only in that the initial encryption e_1 is generated by encrypting 0. Thus, this is indistinguishable from the second game by the security of the encryption scheme.

Since the first game is indistinguishable from the third, the probability that the adversary $\mathcal{A}_2$ can output 1 in each game can differ only negligibly. Thus, the simulation is successful.

Next, we consider a malicious signer.

Recall that we must define a simulator $\mathcal{S}$ which gets as input parameters $params$, the public key pk of the signer, and a commitment $comm$ to the message to be signed, and must impersonate an honest user without knowing the message contained in the commitment m. Consider the following simulator:

1. $\mathcal{S}$ receives pk_{hom}, e_1 from the adversary.

2. $\mathcal{S}$ chooses a random value $t \leftarrow [0, 2^k p^2]$. It computes $e_2 = e_1 \oplus \mathsf{Enc}(pk_{hom}, t)$, and sends e_2 to the adversary.

3. $\mathcal{S}$ uses the simulator for the zero knowledge proof to interact with the adversary. (details?)

4. $\mathcal{S}$ receives σ^* from the adversary and checks it is a valid signature on $m' = t \bmod p$.

Now we show that this $\mathcal{S}$ does a successful simulation: Consider the following series of games:

- In the first game, $pk, m, open, state, comm$ are generated as in the real protocol, and then the adversary $\mathcal{A}_2(state)$ interacts with an honest user as defined above.

- In the second game, $sk, pk, m, open, state, comm$ are generated the same way, but now $\mathcal{A}_2(state)$ interacts with a $\mathcal{S}'$, which behaves as in the real protocol for steps 1 and 2, but then behaves as $\mathcal{S}$ for step 3. The only difference then is that here we use the zero-knowledge simulator to do the interactive proof. Thus this is indistinguishable from the previous game by the zero knowledge property of the ZK proof system.

- In the last game, $sk, pk, m, open, state, comm$ are generated the same way, and then $\mathcal{A}_2(state)$ interacts with $\mathcal{S}$. This differs from the second game only in that e_2 is generated by computing $e_1 \oplus enc(t)$ rather than by computing $e_2 = ((e_1 \oplus enc(m)) * r_1) \oplus \mathsf{Enc}(r_2 p)$. As t is chosen from $[0, 2^k p]$ and e_1 encrypts a value from $\mathbb{Z}_p$, the value encrypted in $e_1 \oplus enc(t)$ will be distributed statistically close to the uniform distribution over $[0, 2^k p]$. Similarly, the value encrypted in $e_2 = ((e_1 \oplus enc(m)) * r_1) \oplus$

$\mathsf{Enc}(r_2 p)$ will be distributed statistically close to the uniform distribution over $[0, 2^k p]$ and hence this game is indistinguishable from the previous one.

Since the first game is indistinguishable from the third, the probability that the adversary $\mathcal{A}_2$ can output 1 in each game can differ only negligibly. Thus, User Privacy holds.

$\square$

Chapter 5

NIZK Proofs for Pseudorandom Functions

Secure pseudorandomness was first introduced by Blum and Micali with the notion of pseudorandom generators [9]. Pseudorandom generators are used to generate a long random string from a short random seed. The definition guarantees that as long as the seed is chosen at random and kept secret, the resulting long string will be indistinguishable from a truly random string.

Pseudorandom functions introduced by Goldreich, Goldwasser and Micali [50] are an extension of this idea in which a fixed seed is used to compute values of a function on arbitrary inputs. As long as the seed is chosen at random, any polynomial time adversary who is not given the seed, but who is allowed to query the function on arbitrarily many inputs of his choice, will not be able to tell that he is not interacting with a completely random function.

Verifiable random functions are a more recent variation on this idea introduced by Micali, Rabin, and Vadhan [66]. In this case, each secret seed is associated with a public key. Thus, the functions can be publicly indexed by this public key. The owner of the seed can not only compute the value of the function on arbitrary inputs; he can also choose to provide a proof that the given value is indeed the output of the function indexed by the corresponding public key. The definition requires that even when we allow a polynomial-time adversary to request function values and proofs for arbitrary inputs, if he then requests the function value for an input of his choice without a proof, he should not be able to distinguish the correct output from a completely random value. At the same time, the functions must be verifiable: for a given public key and a given input, there should be exactly one output value for which a user can compute a valid proof.

We define another class of functions with stronger pseudorandomness properties. Intuitively, we want to require that the output of these functions be indistinguishable from truly random values even when the proofs are given. We capture this by saying that there should be a simulator who can compute the system parameters in such a way that, for any public key it can choose any random output value and fake a proof that it is the correct output of the function indexed by this public key. We require that a polynomial-time adversary who is allowed to query the function on arbitrary inputs should be unable to determine whether he is interacting with simulated parameters, random output values and simulated proofs, or whether he is interacting with honestly generated parameters, output values computed according to the correct function, and honestly generated proofs. At the same time, when the system parameters are generated honestly, we have the same verifiability guarantees as in a verifiable random function. That is, no adversary should be able to find valid proofs of two different output values for the same input and public key. We call functions which satisfy these requirements *simulatable verifiable random functions*(sVRFs).

As we define them, the sVRF proofs are not actually zero knowledge. We could define a stronger notion, where we simply require a PRF where we can efficiently commit to the seed and then prove that a given value is the correct output for a given input when using the seed contained within a given commitment. In our definition of sVRFs, we chose instead to give the minimum requirements necessary to capture the above notion that a given value remains pseudorandom even after a corresponding proof has been given. In particular, it is possible that our proofs leak some information about the secret seed, as long as this information does not compromise the pseudorandomness of any of the output values. We will see that this is strong enough for the application given in Chapter 8. However, in some cases, we will see that the stronger notion is necessary. Thus in Section 5.3, we will show a construction with fully secure NIZK proofs, and in Chapter 9 we will show an application which makes use of this stronger property.

5.1 On Defining sVRFs

We begin by adapting the definition of Micali, Rabin and Vadhan [66] in the public parameters model.

Definition 28 (VRF in the public parameters model)**.** *Let* $params(\cdot)$ *be an algorithm generating public parameters* $\mathtt{p}$ *on input security parameter* 1^k. *Let* $D(\mathtt{p})$ *and* $R(\mathtt{p})$ *be families of efficiently samplable domains for all* $\mathtt{p} \in params$. *The set of algorithms* $(G, \mathtt{Eval}, \mathtt{Prove}, \mathtt{Verify})$ *constitutes a verifiable random function (VRF) for parameter model* $params$, *input domain* $D(\cdot)$ *and output range* $R(\cdot)$ *if*

Correctness *Informally, correctness means that the verification algorithm* $\mathtt{Verify}$ *will always accept* $(\mathtt{p}, pk, x, y, \pi)$ *when* $y = F_{pk}(x)$, *and* π *is the proof of this fact generated using* $\mathtt{Prove}$. *More formally,* $\forall k, \mathtt{p} \in params(1^k), x \in D(\mathtt{p})$,

$$\Pr[(pk, sk) \leftarrow G(\mathtt{p}); y = \mathtt{Eval}(\mathtt{p}, sk, x); \pi \leftarrow \mathtt{Prove}(\mathtt{p}, sk, x); \\ b \leftarrow \mathtt{Verify}(\mathtt{p}, pk, x, y, \pi) \ : \ b = 1] \quad = \quad 1$$

Pseudorandomness *Informally, pseudorandomness means that, on input* $(\mathtt{p}, pk)$, *even with oracle access to* $\mathtt{Eval}(\mathtt{p}, sk, \cdot)$ *and* $\mathtt{Prove}(\mathtt{p}, sk, \cdot)$, *no adversary can distinguish* $F_{pk}(x)$ *from a random element of* $R(\mathtt{p})$ *without explicitly querying for it. More formally,* $\forall$ *PPT* $\mathcal{A}$, $\exists$

negligible ν such that

$$\begin{aligned} &\Pr[\mathtt{p} \leftarrow params(1^k); (pk, sk) \leftarrow G(\mathtt{p}); \\ &(Q_e, Q_p, x, state) \leftarrow \mathcal{A}^{\mathtt{Eval}(\mathtt{p}, sk, \cdot), \mathtt{Prove}(\mathtt{p}, sk, \cdot)}(\mathtt{p}, pk); \\ &y_0 = \mathtt{Eval}(\mathtt{p}, sk, x); y_1 \leftarrow R(\mathtt{p}); b \leftarrow \{0,1\}; \\ &(Q'_e, Q'_p, b') \leftarrow \mathcal{A}^{\mathtt{Eval}(\mathtt{p}, sk, \cdot), \mathtt{Prove}(\mathtt{p}, sk, \cdot)}(state, y_b) \\ &: b' = b \wedge x \notin (Q_e \cup Q_p \cup Q'_e \cup Q'_p)] \quad \leq \quad 1/2 + \nu(k) \end{aligned}$$

where Q_e and Q_p denote, respectively, the contents of the query tape that records $\mathcal{A}$'s queries to its `Eval` *and* `Prove` *oracles in the first query phase, and Q'_e and Q'_p denote the query tapes in the second query phase.*

Verifiability *For all k, for all $\mathtt{p} \in params(1^k)$, there do not exist $(pk, x, y_1, \pi_1, y_2, \pi_2)$ such that $y_1 \neq y_2$, but $\mathtt{Verify}(\mathtt{p}, pk, x, y_1, \pi_1) = \mathtt{Verify}(\mathtt{p}, pk, x, y_2, \pi_2) =$ ACCEPT.*

Note that verifiability in the definition above can be relaxed so as to only hold computationally (as opposed to unconditionally).

Simulatability, as defined below, is the novel aspect of sVRFs, setting them apart from VRFs as previously defined. First, we give the definition, and then we discuss variations.

Definition 29 (Simulatable VRF)**.** *Let $(params, G, \mathtt{Eval}, \mathtt{Prove}, \mathtt{Verify})$ be a VRF (according to Definition 28). They constitute a simulatable VRF if there exist algorithms (SimParams, SimG, SimProve) such that for all PPT $\mathcal{A}$, $\mathcal{A}$'s views in the following two games are indistinguishable:*

Game Real $\mathtt{p} \leftarrow params(1^k)$ *and then $\mathcal{A}(\mathtt{p})$ gets access to the following oracle $\mathcal{R}$: On query NewPK, $\mathcal{R}$ obtains and stores $(pk, sk) \leftarrow G(\mathtt{p})$, and returns pk to $\mathcal{A}$. On query (pk, x), $\mathcal{R}$ verifies that (pk, sk) has been stored for some sk. If not it returns "error". If so, it returns $y = \mathtt{Eval}(\mathtt{p}, sk, x)$ and $\pi \leftarrow \mathtt{Prove}(\mathtt{p}, sk, x)$.*

Game Simulated $(\mathtt{p}, t) \leftarrow$ *SimParams(1^k), and then $\mathcal{A}(\mathtt{p})$ gets access to the following oracle* Sim*: On query NewPK,* Sim *obtains and stores $(pk, sk) \leftarrow$ SimG$(\mathtt{p}, t)$, and returns pk to $\mathcal{A}$. On query (pk, x),* Sim *verifies if (pk, sk) has been stored for some sk. If not, it returns "error". If so,* Sim *(1) checks if x has previously been queried, and if so, returns the answer stored; (2) otherwise,* Sim *obtains $y \leftarrow R(\mathtt{p})$ and $\pi \leftarrow$ SimProve$(\mathtt{p}, sk, x, y, t)$, and returns and stores (y, π).*

5.1.1 Simplifying the Definition

The games in the above definition need to store multiple public keys and secret keys, as well as responses to all the queries issued so far, and consistently respond to multiple queries corresponding to all these various keys. It is clear that this level of security is desirable: we want an sVRF to retain its security properties under composition with other instances within the same system. A natural question is whether we can simplify the games by restricting the adversary to just one NewPK query or just one (pk, x) query per pk without weakening the security guarantees. In fact, the four possible combinations of such restrictions yield four distinct security notions, as we show in the full version of this paper.

Although we cannot simplify Definition 29 in this way, we can give a seemingly simpler definition (one that only allows one NewPK query from the adversary) that is strictly stronger than Definition 29 in that it requires that the adversary cannot distinguish the real game from the simulated one, *even with the knowledge of the trapdoor* t.

Definition 30 (Trapdoor-indistinguishable sVRF). *Let* $(params, G, \mathtt{Eval}, \mathtt{Prove}, \mathtt{Verify})$ *be a VRF (as in Definition 28). They constitute a trapdoor-indistinguishable (TI) sVRF if there exist algorithms* $(\mathit{SimParams}, \mathit{SimG}, \mathit{SimProve})$ *such that the distribution* $params(1^k)$ *is computationally indistinguishable from the distribution* $\mathit{SimParams}(1^k)$ *and for all PPT* $\mathcal{A}$*,* $\mathcal{A}$*'s views in the following two games are indistinguishable:*

Game Real Proofs $(\mathtt{p}, t) \leftarrow \mathit{SimParams}(1^k)$, $(pk, sk) \leftarrow G(\mathtt{p})$ *and then* $\mathcal{A}(\mathtt{p}, t, pk)$ *gets access to the following oracle* $\mathcal{R}$*: On query* x*,* $\mathcal{R}$ *returns* $y = \mathtt{Eval}(\mathtt{p}, sk, x)$ *and* $\pi \leftarrow \mathtt{Prove}(\mathtt{p}, sk, x)$.

Game Simulated Proofs $(\mathtt{p}, t) \leftarrow \mathit{SimParams}(1^k)$, $(pk, sk) \leftarrow \mathit{SimG}(\mathtt{p}, t)$*, and then* $\mathcal{A}(\mathtt{p}, t, pk)$ *gets access to the following oracle* Sim*: On query* x*,* Sim *(1) checks if* x *has previously been queried, and if so, returns the answer stored; (2) otherwise, obtains* $y \leftarrow R(\mathtt{p})$ *and* $\pi \leftarrow \mathit{SimProve}(\mathtt{p}, sk, x, y, t)$*, and returns and stores* (y, π).

By a fairly standard hybrid argument, we have the following lemma (see the full version for the proof):

Lemma 7. *If* $(params, G, \mathtt{Eval}, \mathtt{Prove}, \mathtt{Verify})$ *is a TI-sVRF, it is an sVRF.*

5.1.2 Weak Trapdoor-Indistinguishable sVRF

We now define a somewhat weaker notion of TI sVRFs, in which a simulator can only give fake proofs for those values of the output range that it has sampled itself in some special way.

Definition 31 (Weak TI-sVRF). *Let* $(G, \mathtt{Eval}, \mathtt{Prove}, \mathtt{Verify})$ *be a VRF in the* $params(1^k)$ *model with domain* $D(\cdot)$ *and range* $R(\cdot)$*. They constitute a weak trapdoor-indistinguishable (TI) sVRF if there exist algorithms* $(\mathit{SimParams}, \mathit{SimG}, \mathit{SimProve}, \mathit{SimSample})$ *such that the distribution* $params(1^k)$ *is computationally indistinguishable from the distribution* $\mathit{SimParams}(1^k)$ *and for all PPT* $\mathcal{A}$*,* $\mathcal{A}$*'s views in the following two games are indistinguishable:*

Game Real Proofs $(\mathtt{p}, t) \leftarrow \mathit{SimParams}(1^k)$, $(pk, sk) \leftarrow G(\mathtt{p})$ *and then* $\mathcal{A}(\mathtt{p}, t, pk)$ *gets access to the following oracle: On query* x*, the oracle returns* $y = \mathtt{Eval}(\mathtt{p}, sk, x)$ *and* $\pi \leftarrow \mathtt{Prove}(\mathtt{p}, sk, x)$.

Game Simulated Proofs $(\mathtt{p}, t) \leftarrow \mathit{SimParams}(1^k)$, $(pk, sk) \leftarrow \mathit{SimG}(\mathtt{p}, t)$*, and then* $\mathcal{A}(\mathtt{p}, t, pk)$ *gets access to the following oracle: On query* x*, the oracle (1) checks if* x *has previously been queried, and if so, returns the answer stored; (2) otherwise, obtains* $(y, w) \leftarrow \mathit{SimSample}(\mathtt{p}, t, sk, x)$ *and* $\pi \leftarrow \mathit{SimProve}(\mathtt{p}, sk, x, y, t, w)$*, and returns and stores* (y, π).

We now show that a weak TI-sVRF where *SimSample* outputs a uniformly random element of a sufficiently large set can be converted to a TI-sVRF with binary range. Let $(G, \texttt{Eval}, \texttt{Prove}, \texttt{Verify})$ be a weak TI-sVRF in the $params$ model with domain $D(\texttt{p})$, and range $R(\texttt{p}) \subseteq \{0,1\}^{m(k)}$ for some polynomial m for all $\texttt{p} \in params(1^k)$. Consider the following algorithms:

$params^*(1^k)$ Pick $r \leftarrow \{0,1\}^{m(k)}$, $\texttt{p} \leftarrow params(1^k)$; return $\texttt{p}^* = (r, \texttt{p})$.

G^* On input $\texttt{p}^* = (r, \texttt{p})$, output $(pk^*, sk^*) \leftarrow G(\texttt{p})$.

$\texttt{Eval}^*$ **and** $\texttt{Prove}^*$ On input $\texttt{p}^* = (r, \texttt{p})$, sk^*, and $x \in D(\texttt{p})$, compute $y = \texttt{Eval}(\texttt{p}, sk^*, x)$. Let $y^* = y \cdot r$, where by "$\cdot$", we denote the inner product, i.e. $y \cdot r = \bigoplus_{i=1}^{|y|} y_i r_i$. $\texttt{Eval}^*$ outputs y^*. $\texttt{Prove}^*$ picks $\pi \leftarrow \texttt{Prove}(\texttt{p}, sk^*, x)$ and outputs $\pi^* = (\pi, y)$.

$\texttt{Verify}^*$ On input $\texttt{p}^* = (r, \texttt{p})$, pk^*, $x \in D(\texttt{p})$, $y^* \in \{0,1\}$, $\pi^* = (\pi, y)$: accept iff $\texttt{Verify}(\texttt{p}, pk, x, y, \pi)$ accepts and $y^* = r \cdot y$.

Lemma 8. *Suppose* $(G, \texttt{Eval}, \texttt{Prove}, \texttt{Verify})$ *is a weak TI-sVRF with* $(\mathit{SimParams}, \mathit{SimSample}, \mathit{SimG}, \mathit{SimProve})$ *as in Definition 31. Let* ρ *be such that for all* $(\texttt{p}, t) \in \mathit{SimParams}(1^k)$*, for all* $x \in D(\texttt{p})$*, for all* $(sk, pk) \in \mathit{SimG}(\texttt{p}, t)$*,* $|\mathit{SimSample}(\texttt{p}, t, sk, x)| \geq \rho(k)$*, and SimSample is a uniform distribution over its support. Let* μ *be such that for all* $\texttt{p} \in params(1^k)$*,* $|D(\texttt{p})| \leq \mu(k)$*. If there exists a negligible function* ν *such that* $\mu(k)\rho(k)^{-\frac{1}{3}} = \nu(k)$ *then* $(G^*, \texttt{Eval}^*, \texttt{Prove}^*, \texttt{Verify}^*)$ *as constructed above are a TI-sVRF in the* $params^*$ *model with domain* $D(\texttt{p})$*, and range* $\{0,1\}$*.*

Proof. Correctness, verifiability and pseudorandomness follow easily from the respective properties of the weak TI-sVRF (recall that a weak TI-sVRF is still a VRF – the "weak" part refers to simulatability only). In particular, pseudorandomness follows by standard techniques such as the leftover hash lemma.

We must show TI-simulatability. We first prove a useful claim. Consider specific values $(\texttt{p}, t) \in \mathit{SimParams}(1^k)$, $(pk, sk) \in \mathit{SimG}(\texttt{p}, t)$. Since t and sk are fixed, the distributions $R'(x) = \mathit{SimSample}(\texttt{p}, t, sk, x)$ and $\mathit{Bad}(x) = \{r \in \{0,1\}^{m(k)} : |\Pr[y \leftarrow R'(x) : y \cdot r = 1] - .5| \geq |R'(x)|^{-\frac{1}{3}}\}$ are well-defined. In English, $\mathit{Bad}(x)$ is the set of those r's for which the random variable $y \cdot r$ (where y is sampled uniformly at random from $R'(x)$, i.e. sampled according to $\mathit{SimSample}(\texttt{p}, t, sk, x)$) is biased by at least $|R'(x)|^{-\frac{1}{3}}$ from a random bit.

Claim. $\forall x \in D(\texttt{p})$*,* $Pr[r \leftarrow \{0,1\}^{m(k)} : r \in \mathit{Bad}(x)] \leq |R'(x)|^{-\frac{1}{3}}$.

Proof. (Of claim.) Suppose $x \in D(\texttt{p})$ is fixed. Let $\mathit{Weight}(r) = \sum_{y \in R'(x)} y \cdot r$. By definition of $\mathit{Bad}(x)$, $r \in \mathit{Bad}(x)$ if and only if $|\mathit{Weight}(r)/|R'(x)| - .5| \geq |R'(x)|^{-\frac{1}{3}}$. It is easy to see that, if the probability is taken over the choice of r, then $\mathit{Exp}[\mathit{Weight}(r)/|R'(x)|] = .5$. On the other hand, for any pair $y_1 \neq y_2 \in R'(x)$, $y_1 \cdot r$ is independent from $y_2 \cdot r$, and so $\mathit{Weight}(r) = \sum_{y \in R'(x)} y \cdot r$ is a sum of pairwise independent random variables. Thus, $\mathit{Var}[\mathit{Weight}(r)] = \sum_{y \in R'(x)} \mathit{Var}[y \cdot r] = |R'(x)|/4$, and $\mathit{Var}[\mathit{Weight}(r)/|R'(x)|] = 1/4|R'(x)|$. Plugging Exp and Var for $\mathit{Weight}(r)/|R'(x)|$ into Chebyshev's inequality, we get $\Pr[|\mathit{Weight}(r)/|R'(x)| - .5| \geq |R'(x)|^{-\frac{1}{3}}\}] \leq |R'(x)|^{-\frac{1}{3}}$ which completes the proof. □

Now we will show that the simulatability property holds. Consider the following algorithms:

SimParams* On input 1^k, obtain $(\mathtt{p}, t) \leftarrow \mathit{SimParams}(1^k)$, $r \leftarrow \{0,1\}^{m(k)}$. Output $\mathtt{p}^* = (r, \mathtt{p})$, $t^* = t$.

SimG* On input $(\mathtt{p}^*, t^*)$, where $\mathtt{p}^* = (r, \mathtt{p})$ obtain $(pk, sk) \leftarrow \mathit{SimG}(\mathtt{p}, t^*)$. Output $pk^* = pk$, $sk^* = sk$.

SimProve* On input $(\mathtt{p}^*, sk^*, x, y^*, t^*)$ where $\mathtt{p}^* = (r, \mathtt{p})$, repeat the following up to k times until $y \cdot r = y^*$: $(y, w) \leftarrow \mathit{SimSample}(\mathtt{p}, t, sk, x)$. If after k calls to *SimSample*, $y \cdot r \neq y^*$, output "fail". Else obtain $\pi \leftarrow \mathit{SimProve}(\mathtt{p}, t, sk, x, (y, w))$. Output $\pi^* = (\pi, y)$.

We define two intermediate games in which the adversary is given an oracle that is similar to Game Simulated Proofs from the TI-sVRF definition in that it does not use `Eval` and `Prove`; instead of `Eval`, it uses *SimSample* (from the weak TI-sVRF definition) to obtain (y, w), and then outputs $y^* = y \cdot r$. The two games generate the proofs in different ways: Game Intermediate Real Proof just uses w and *SimProve* of the weak TI-sVRF definition to generate π, while Game Intermediate Simulated Proof uses *SimProve** defined above. More precisely:

Game Intermediate Real Proofs $(\mathtt{p}^*, t^*) \leftarrow \mathit{SimParams}^*(1^k)$, $(pk^*, sk^*) \leftarrow \mathit{SimG}^*(\mathtt{p}^*, t^*)$, and then $\mathcal{A}$ $(\mathtt{p}^*, t^*, pk^*)$ gets access to the following oracle: On query x, the oracle (1) checks if x has previously been queried, and if so, returns the answer stored; (2) otherwise, obtains $(y, w) \leftarrow \mathit{SimSample}(\mathtt{p}, t, sk, x)$, $y^* = y \cdot r$, and $\pi \leftarrow \mathit{SimProve}(\mathtt{p}, sk, x, y, t, w)$, $\pi^* = (\pi, y)$, and returns and stores (y^*, π^*).

Game Intermediate Simulated Proofs $(\mathtt{p}^*, t^*) \leftarrow \mathit{SimParams}^*(1^k)$, $(pk^*, sk^*) \leftarrow \mathit{SimG}^*(\mathtt{p}^*, t^*)$, and then $\mathcal{A}(\mathtt{p}^*, t^*, pk^*)$ gets access to the following oracle: On query x, the oracle (1) checks if x has previously been queried, and if so, returns the answer stored; (2) otherwise, obtains $(y', w') \leftarrow \mathit{SimSample}(\mathtt{p}, t, sk, x)$, $y^* = y' \cdot r$, and $\pi^* \leftarrow \mathit{SimProve}^*(\mathtt{p}, sk, x, y^*, t)$, and returns and stores (y^*, π^*).

We now argue that these intermediate games are indistinguishable from Game Real Proofs and Game Simulated Proofs as specified by the definition of TI-sVRF, instantiated with $(\mathit{SimParams}, \mathit{SimG}, \mathit{SimSample}, \mathit{SimProve})$ that follow from simulatability of our weak TI-sVRF, and with $(\mathit{SimParams}^*, \mathit{SimG}^*, \mathit{SimProve}^*)$ defined above. First, it is straightforward to see that an adversary distinguishing between Game Real Proofs and Game Intermediate Real Proofs directly contradicts the simulatability property of weak TI-sVRFs.

The only difference between Game Intermediate Simulated Proofs and Game Simulated Proofs, is the choice of the bit y^*: in the former, it is chosen using *SimSample*, i.e. indistinguishably from the way it is chosen in the real game. In the latter, it is chosen at random. If we condition on the event that for all x, $r \notin \mathit{Bad}(x)$, these two distributions are statistically close.

The only thing left to show is that the two intermediate games defined above are indistinguishable. If we condition on the event that we never fail, then the two games are identical. Note that if for all x, $r \notin \mathit{Bad}(x)$, then the probability that we fail on a particular query is $\leq (1/2 + |R'(x)|^{-\frac{1}{3}})^k$ which is negligible.

Thus we have shown that if the probability that $r \in \mathit{Bad}(x)$ for some x is negligible, then Game Real Proofs is indistinguishable from Game Simulated Proofs. By the union bound, combined with the claim, $\Pr[r \leftarrow \{0,1\}^{m(k)} : \exists x \in D(\mathtt{p})$ such that $r \in \mathit{Bad}(x)] \leq |D(\mathtt{p})||R'(x)|^{-\frac{1}{3}}$, which is equal to $\nu(k)$ by the premise of the lemma. $\square$

From Lemmas 7 and 8, we see that from a weak TI-sVRF satisfying the conditions of Lemma 8, we can construct an equally efficient sVRF with range $\{0, 1\}$.

Remark. Note that, even though the support of *SimSample*$(\mathtt{p}, t, sk, x)$ is quite large, the construction above only extracts one bit of randomness from it. Although it can be easily extended to extract a logarithmic number of random bits, there does not seem to be a black-box construction extracting a superlogarithmic number of bits. Suppose ext is a procedure that extracts ℓ bits from y, so $y^* = ext(y)$ is of length ℓ. Then how would *SimProve** work to generate a proof that y^* is correct? It needs to call *SimProve*$(\mathtt{p}, sk, x, y, t, w)$ for some y such that $y^* = ext(y)$ and w is an appropriate witness. It seems that the only way to obtain such a pair (y, w) is by calling *SimSample*$(\mathtt{p}, t, sk, x)$; in expectation, 2^ℓ calls to *SimSample* are needed to obtain an appropriate pair (y, w); if ℓ is superlogarithmic, this is prohibitively inefficient.

5.2 Simulatable Verifiable Random Functions: Constructions

CONSTRUCTING AN SVRF. Our first result is a direct construction of a simulatable VRF based on the Subgroup Decision assumption (SDA) [13], and an assumption related to the Q-BDHI assumption [11]. Dodis and Yampolskiy [43] used the Q-BDHI assumption to extend the Boneh-Boyen short signature scheme [10] and derive a VRF. The Dodis-Yampolskiy VRF is of the form $F_s(x) = e(g, g)^{1/(s+x)}$, where g is a generator of some group G_1 of prime order q, and $e : G_1 \times G_1 \mapsto G_2$ is a bilinear map. The secret key is s while the public key is g^s. The DY proof that $y = F_s(x)$ is the value $\pi = g^{1/(s+x)}$ whose correctness can be verified using the bilinear map.

Our sVRF is quite similar, only it is in a composite-order group with a bilinear map: the order of G_1 is an RSA modulus $n = pq$. This is what makes simulatability possible. In our construction, the public parameters consist of (g, A, D, H), all generators of G_1. As before, the secret key is s, but now the public key is A^s. $F_s(x) = e(H, g)^{1/(s+x)}$, and the proof is a randomized version of the DY proof: $\pi = (\pi_1, \pi_2, \pi_3)$, where $\pi_1 = H^{r/(s+x)}/D^r$, $\pi_2 = g^{1/r}$ and $\pi_3 = A^{(s+x)/r}$. It turns out that, when A generates the entire G_1, there is a unique $y = F_s(x)$ for which a proof exists. However, when A belongs to the order-p subgroup of G_1 (as is going to be the case when the system parameters are picked by the simulator), the verification tests correctness only as far as the order-p subgroup is concerned, and so the order-q component of $F_s(x)$ is unconstrained. The proof of security requires that a strengthening of Q-BDHI hold for the prime-order subgroups of G_1, and that the SDA assumption holds so that A picked by the simulator is indistinguishable from the correct A.

Next, we give a second construction of sVRFs which gives efficient proofs for a pseudorandom function whose range is the bilinear group G_1. In fact, it can be shown that it satisfies a stronger property in that the proofs are fully zero knowledge. It is based on the Groth-Sahai proofs system combined with the Dodis-Yampolskiy VRF, and thus, it can be instantiated using SXDH or the decisional linear assumption together with the assumption that BDHI holds in G_1.

Finally, we also give, as proof of concept, a construction under general assumptions, based on general multi-theorem NIZK.

5.2.1 Construction from Composite Order Bilinear Groups

We first present a construction for a weak TI-sVRF with a large output range. As we have shown, this can then be transformed into an sVRF with range $\{0, 1\}$. The security relies on the BDHI and BDHBI assumptions (see section2.3 for details).

The assumption in Definition 23 is a new assumption which can be shown to imply Q-BDHI. We will assume that it holds for the prime order subgroup of composite order bilinear groups that can be efficiently instantiated [13].

Definition 32 (SDA [13]). *A family $\mathcal{G}$ of groups satisfies the subgroup decision assumption if no PPT $\mathcal{A}$, on input $(instance, challenge)$ can distinguish if its challenge is of type 1 or type 2, where $instance$ and $challenge$ are defined as follows: $instance = (G_1, G_2, n, e, h)$ where $n = pq$ is a product of two primes of length $poly(k)$ (for k a sec. param.), G_1, G_2 are groups of order n returned by $\mathcal{G}(q, p)$, $e : G_1 \times G_1 \to G_2$ is a bilinear map, h is a random generator of G_1, $challenge$ of type 1 is g, a random generator of G_1, while $challenge$ of type 2 is g_p, a random order-p element of G_1.*

The weak TI-sVRF construction is as follows:

$params$ On input 1^k, choose groups G_1, G_2 of order $n = pq$ for primes p, q, where $|p|$ and $|q|$ are polynomial in k, with bilinear map $e : G_1 \times G_1 \to G_2$. Choose random generators g, H, A, D for G_1. $params$ will output $\mathtt{p} = (G_1, G_2, n, e, g, H, A, D)$.

Domain and range The input domain $\mathcal{D}(\mathtt{p})$ consists of integers $1 \leq x \leq l(k)$ where $l(k) < 2^{|q|-1}$ (We will later see the connection between $l(k)$ and $Q(k)$ by which our assumption is parameterized.) Note that $\mathcal{D}(\mathtt{p})$ depends only on k, not on $\mathtt{p}$. $R(\mathtt{p}) = G_2$.

G On input $\mathtt{p}$, pick $s \leftarrow \mathbb{Z}_n^*$, output $sk = s$, $pk = A^s$.

`Eval` On input $(\mathtt{p}, sk, x)$, output $e(H, g)^{\frac{1}{s+x}}$.

`Prove` On input $(\mathtt{p}, sk, x)$, pick $r \leftarrow \mathbb{Z}_n^*$, and output $\pi = (\pi_1, \pi_2, \pi_3)$, where $\pi_1 = H^{\frac{r}{s+x}}/D^r$, $\pi_2 = g^{\frac{1}{r}}$, $\pi_3 = A^{\frac{x+s}{r}}$.

`Verify` On input $(\mathtt{p}, sk, x, y, \pi)$, parse $\pi = (\pi_1, \pi_2, \pi_3)$ and verify that $e(\pi_1, \pi_2)e(D, g) = y$, $e(\pi_3, g) = e(A^x pk, \pi_2)$, $e(\pi_1, \pi_3)e(D, A^x pk) = e(H, A)$.

Lemma 9 (Efficiency). *The construction above has the following efficiency: Generating a proof for a given value y requires 4 exponentiations in the composite order group G_1. The resulting proof consists of 3 elements of the composite order group G_1. Verifying the proof requires 5 composite order bilinear pairings.* [1]

Theorem 15. $(G, \mathtt{Eval}, \mathtt{Prove}, \mathtt{Verify})$ *as described above constitute a weak TI-sVRF for parameter model $params$, input domain $\mathcal{D}$ of size l, and output range G_2 (where G_2 is as output by $params$) under the SDA*

[1]Note that the resulting construction is only a weak sVRF, and conversion to full sVRF with output range $\{0, 1\}^{\ell(k)}$ will decrease efficiency by a factor of $\ell(k)$ as described in Section 5.1.2. Also note that here we require composite order bilinear groups, which are generally considered less secure than prime order bilinear groups of comparable size. This means that in order to get the same security guarantees, we must use a somewhat larger groups than those considered in our other constructions.

assumption combined with the $(l(k), \nu(k)/l^2(k))$-BDHBI, where ν is an upper bound on the asymptotic advantage that any probabilistic polynomial-time algorithm has in breaking the simulatability game of Definition 31.

Proof. **Correctness** follows from construction.

Verifiability: Suppose there exists an adversary who, given parameters $\mathtt{p} = (G_1, G_2, n, e, g, H = g^h, A = g^a, D = g^d)$ generated by $params$ can produce $pk, y, y', \pi = (\pi_1, \pi_2, \pi_3), \pi' = (\pi_1', \pi_2', \pi_3')$ such that $\mathtt{Verify}(\mathtt{p}, pk, y, \pi) = \mathtt{Verify}(\mathtt{p}, pk, y', \pi') = 1$. Then we will show that $y = y'$.

Let $\lambda, \mu, \mu', \sigma, \phi, \theta, \sigma', \phi', \theta' \in \mathbb{Z}_n$ be the exponents such that $pk = g^\lambda, y = g^\mu, y' = g^{\mu'}, \pi_1 = g^\sigma, \pi_2 = g^\phi, \pi_3 = g^\theta, \pi_1' = g^{\sigma'}, \pi_2' = g^{\phi'}, \pi_3' = g^{\theta'}$.

If the verifications succeed, then we get that the following equations hold in $\mathbb{Z}_n$: $\sigma\phi + d = \mu, \quad \theta = (ax + \lambda)\phi, \quad \theta\sigma + d(ax + \lambda) = ha$.

Solving this system of equations gives us: $ha = \mu(ax + \lambda)$. Similarly, if (y', π') satisfy the verification equations, then we know that $ha = \mu'(ax + \lambda)$. H, A are generators for G_1, so $h, a \in \mathbb{Z}_n^*$, and therefore, $\mu'(ax + \lambda) \in \mathbb{Z}_n^*$, and $\mu(ax + \lambda) \in \mathbb{Z}_n^*$. This in turn means that $\mu', \mu, (ax + \lambda) \in \mathbb{Z}_n^*$.

From the solutions to the above equations, we know $\mu(ax + \lambda) = \mu'(ax + \lambda)$. Since $(ax + \lambda) \in \mathbb{Z}_n^*$, we can compute a unique inverse $(ax + \lambda)^{-1}$, and conclude that $\mu = \mu'$, and $y = y'$.

Note that this argument relies crucially on the fact that $h, a \in \mathbb{Z}_n^*$. In our simulation, we will instead choose $a = 0 \mod q$, which will allow us to avoid this binding property.

Pseudorandomness follows under the Q-BDHI Assumption from pseudorandomnesss of the Dodis-Yampolskiy VRF [43].

Simulatability: Consider the following simulator algorithms:

SimParams(1^k) Choose groups G_1, G_2 of order $n = pq$ for prime p, q, where $|p|$ and $|q|$ are polynomial in k, with bilinear map $e : G_1 \times G_1 \to G_2$. Let G_p be the order p subgroup of G_1, and let G_q be the order q subgroup of G_1. Let $(A, g_p, H_p, D_p) \leftarrow G_p^4$ and $(g_q, H_q, D_q) \leftarrow G_q^3$. Let $g = g_p g_q$, $H = H_p H_q$, and $D = D_p D_q$. Output $\mathtt{p} = (G_1, G_2, n, e, g, H, A, D), t = (g_p, g_q, H_p, H_q, D_p, D_q)$.

This is identical to $params$ except that $A \in G_p$, so that the verification algorithm cannot properly verify the G_q components of y and π.

SimG$(\mathtt{p}, t)$ $(sk, pk) \leftarrow G(\mathtt{p})$.

SimSample On input $(\mathtt{p}, t, sk, x)$, pick $w \leftarrow \mathbb{Z}_q^*$.
Let $y = e(H_p, g_p)^{\frac{1}{s+x}} e(g_q, g_q)^w$. Output (y, w). (Note y's G_p component will be correct, while its G_q component will be random.)

SimProve On input $(\mathtt{p}, sk, x, y, t, w)$, pick $r \leftarrow \mathbb{Z}_n^*$;
let $\pi_1 = (H_p^{\frac{r}{s+x}} / D_p^r)(g_q^{wr} / D_q^r)$, $\pi_2 = g^{\frac{1}{r}}$, $\pi_3 = A^{\frac{x+s}{r}}$. Output $\pi = (\pi_1, \pi_2, \pi_3)$. (Note that π's G_p components are correct, while its G_q components are chosen so as to allow us to fake the proof.)

Lemma 10. *The distribution $params(1^k)$ is indistinguishable from the distribution SimParams(1^k) by the Subgroup Decision Assumption.*

Proof. The only difference between these two distributions is that in $params$, A is chosen at random from G_1, and in *SimParams*, A is chosen at random from G_p. Thus, these two distributions are indistinguishable by the Subgroup Decision assumption by a straightforward reduction. $\square$

Lemma 11. *For the algorithms described above, Game Real Proofs and Game Simulated Proofs (as in Definition 31) are indistinguishable with advantage more that $\nu(k)$ by the $(l(k), \nu(k)/l^2(k))$-BDHBI assumption.*

Before we prove this lemma, we will describe and prove an intermediate assumption that follows from the assumptions that we have already made. We state this assumption in terms of any prime order bilinear group. However, we will later assume that this assumption (and the Q-BDHBI assumption) holds over the prime order subgroup of a composite order bilinear group.

Definition 33 ((Q, ν)-Intermediate assumption)**.** *A family $\mathcal{G}$ of groups satisfies the $(Q(k), \nu(k))$-intermediate assumption if for all subsets X of $\mathbb{Z}_{2^{a(k)-1}}$ (where $a(k)$ is a polynomial), of size $Q(k) - 1$ for all $x^* \in \mathbb{Z}_{2^{a(k)-1}} \setminus X$, no PPT $\mathcal{A}$, on input $(instance, challenge)$ can distinguish if its challenge is of type 1 or type 2 with advantage asymptotically higher than $\nu(k)$, for instance and challenge defined as follows: $instance = (G_1, G_2, q, e, g, H, D, \{(H^{r_x \frac{1}{s+x}}/D^{r_x}, g^{\frac{1}{r_x}})\}_{\forall x \in X})$ where q is an $a(k)$-bit prime, G_1, G_2 are groups of order q returned by $\mathcal{G}(q)$, $e : G_1 \times G_1 \to G_2$ is a bilinear map, $(g, H, D) \leftarrow G_1^3$, and $\{r_x\}_{x \in X}$ and s were all picked at random from $\mathbb{Z}_q^*$; challenge of type 1 is $(H^{r^* \frac{1}{s+x^*}}/D^{r*}, g^{\frac{1}{r^*}})$ where $r^* \leftarrow \mathbb{Z}_q^*$, while challenge of type 2 is (g^{R_1}, g^{R_2}) for R_1 and R_2 random from $\mathbb{Z}_q^*$.*

Lemma 12. *(l, ν)-BDHBI assumption implies (l, ν)-intermediate assumption.*

Proof. Suppose there exists an adversary $\mathcal{A}$ who breaks the intermediate assumption for set X of cardinality $l - 1$, and $x^* \notin X$. Then we show an algorithm $\mathcal{B}$ that can break l-BDHBI Assumption.

Algorithm $\mathcal{B}$ will behave as follows: Receive $G, q, e, g, g^{\alpha}, \dots g^{\alpha^l}, g^{\beta}$, and $Z = g^{\frac{1}{\alpha\beta}}$ or $Z = g^R$ for random $R \in Z_q^*$.

Choose random values $\Delta_1, \Delta_2 \leftarrow \mathbb{Z}_q^*$. Implicitly, let $\gamma = \gamma(\alpha) = \Delta_1(\alpha - \Delta_2)\prod_{x \in X}(\alpha + (x - x^*))$. Compute $H = g^{\gamma}$. Note that since this exponent is just an l degree polynomial in α, we can compute this value using $g, \dots g^{\alpha^l}$. If we implicitly define $s = \alpha - x^*$, we will get $H = g^{\Delta_1(\alpha-\Delta_2)\prod_{x \in X}(s+x)}$. (Note that now we know neither s, nor α explicitly.) Note that because of Δ_1, H is uniformly distributed over G_1, and is independent of g. Now we want to provide D. Implicitly we will define $d = \frac{\gamma - \delta}{\alpha}$, where $\delta = \Delta_1\Delta_2\prod_{x \in X}(x - x^*)$ is the constant term of the polynomial in α (represented by $\gamma(\alpha)$). Note now that δ is a quantity $\mathcal{B}$ can compute, while d is only defined implicitly. Since d is a polynomial expression in α, $D = g^d$ can be expressed as a sum of terms $g, g^{\alpha}, \dots, g^{\alpha^{l-1}}$, and computed using the given values. Finally, note that, because of Δ_2, D is uniformly distributed over G_1, and is independent of (g, H).

For all $\hat{x} \in X$: Let $\gamma'(\hat{x}) = \Delta_1(\alpha - \Delta_2)\prod_{x \in X, x \neq \hat{x}}(\alpha + (x - x^*)) = \frac{\gamma}{\hat{x}+s}$. Compute $v = g^{\gamma'(\hat{x})} = g^{\frac{\gamma}{s+\hat{x}}} = H^{\frac{1}{s+\hat{x}}}$. We then choose a random $r_{\hat{x}} \leftarrow \mathbb{Z}_n^*$. We compute and output $(v^{r_{\hat{x}}}/D^{r_{\hat{x}}}, g^{\frac{1}{r_{\hat{x}}}})$.

For x^*: Implicitly define $r^* = \frac{1}{\beta}$. Compute $u_1 = Z^{\delta}$. If $Z = g^{\frac{1}{\alpha\beta}}$, then this is equal to $g^{\frac{\delta}{\alpha\beta}} = g^{\frac{\gamma}{\alpha\beta} - \frac{\gamma-\delta}{\alpha\beta}} = g^{\frac{\gamma}{\alpha\beta}}/g^{\frac{\gamma-\delta}{\alpha\beta}} = H^{r^* \frac{1}{s+x^*}}/D^{r^*}$. Otherwise, this is equal to g^{R_1} for random R_1. Compute $u_2 = (g^{\beta}) = (g^{\frac{1}{r^*}})$. Output (u_1, u_2).

Finally, if $\mathcal{A}$ guesses that he received $(H^{r^{*}\frac{1}{s+x^{*}}}/D^{r^{*}}, g^{\frac{1}{r^{*}}})$, $\mathcal{B}$ guesses that $Z = g^{\frac{1}{\alpha\beta}}$, else that $Z = g_q^R$. If $\mathcal{A}$'s guess is correct, then $\mathcal{B}$'s guess is correct. $\square$

Proof. (of Lemma 11) We first define a series of hybrid games:

Game Hybrid i: Obtain $(\mathtt{p}, t) \leftarrow \mathit{SimParams}(1^k)$, and $(pk, sk) \leftarrow$

$\mathit{SimG}(\mathtt{p}, t)$ and then $\mathcal{A}(\mathtt{p}, t, pk)$ gets access to the following oracle: The oracle begins by storing $j = 0$. On query x, the oracle (1) checks if x has previously been queried, and if so, returns the answer stored. Otherwise, (2) if $j < i$ the oracle obtains $(y, w) \leftarrow \mathit{SimSample}(\mathtt{p}, t, sk, x)$ and $\pi \leftarrow \mathit{SimProve}(\mathtt{p}, sk, x, y, w, t)$, returns and stores (y, π), and increments j. (3) Or if $j \geq i$, the oracle computes $y = \mathtt{Eval}(\mathtt{p}, sk, x)$ and $\pi \leftarrow \mathtt{Prove}(\mathtt{p}, sk, x)$, returns and stores (y, π) and increments j.

Note that in this case, $G(\mathtt{p})$ is identical to $\mathit{SimG}(\mathtt{p}, t)$ for all $\mathtt{p}, t$, so Game Hybrid 0 is identical to Game Real Proofs. Game Hybrid Q, where Q is the maximum number of distinct oracle queries (not including repeated queries) that the adversary is allowed to make, is identical to Game Simulated Proofs. Thus, we have only to show the following lemma:

Lemma 13. *Suppose the (l, ν)-BDHBI Assumption holds in one of the two subgroups of a composite bilinear group. Then, when the size of the domain is at most l, no PPT adversary can distinguish Game Hybrid $i-1$ from Game Hybrid i with advantage higher than νl.*

Proof. Suppose there exists an adversary $\mathcal{A}$ who can distinguish Game Hybrid $i-1$ from Game Hybrid i when the domain $\mathcal{D}$ is of size l. Then we show an algorithm $\mathcal{B}$ that can break the l-intermediate assumption with advantage ϵ.

First we make a guess x^* about which input $\mathcal{A}$ will give in its ith distinct oracle query. Since $|\mathcal{D}| = l$, and all values given to $\mathcal{A}$ will be independent of x^*, we will be correct with probability $1/l$.

Now we will show an algorithm $\mathcal{B}$, which can, with nonnegligible probability, break the intermediate assumption for set $X = \mathcal{D} \setminus \{x^*\}$ and the x^* chosen above. $\mathcal{B}$ will receive $G, p, q, e, g_p, g_q, H_q, D_q$, $\{(H_q^{\frac{r_x}{s_q+x}}/D_q^{r_x}, g_q^{\frac{1}{r_x}})\}_{\forall x \in X}, (Z_1, Z_2)$ for $g_q, H_q, D_q \leftarrow G_q$, and randomly chosen (but unknown) $\{r_x\}_{x \in X}$, $s_q \leftarrow \mathbb{Z}_q^*$. Here, either $(Z_1, Z_2) = (H_q^{r^{*}\frac{1}{s_q+x^{*}}}/D_q^{r*}, g_q^{\frac{1}{r^{*}}})$ or $(Z_1, Z_2) = (g_q^{R_1}, g_q^{R_2})$ for random $R_1, R_2 \leftarrow \mathbb{Z}_q^*$.

First, $\mathcal{B}$ prepares the parameters as follows: Choose $H_p, A, D_p \leftarrow G_p$ and compute $g = g_p g_q$, $H = H_p H_q$, $D = D_p D_q$. Set $\mathtt{p} = (G_1, G_2, n, e, g, H, A, D)$. Let $s_p \leftarrow \mathbb{Z}_p^*$, and $pk = A^{s_p}$. Implicitly, set $s \in \mathbb{Z}_n^*$ to the the element such that $s \mod p = s_p$, and $s \mod q = s_q$. $\mathcal{B}$ sends $\mathtt{p}$ and trapdoor $t = (g_p, g_q, H_p, H_q, D_p, D_q)$ to $\mathcal{A}$.

Now $\mathcal{B}$ must answer $\mathcal{A}$'s queries. We assume (WLOG) that $\mathcal{A}$ does not repeat queries.

When $\mathcal{A}$ sends its j^{th} query, $\hat{x}$, $\mathcal{B}$ proceeds as follows:

If $j < i$: if $\hat{x} = x^*$, then $\mathcal{B}$ has guessed wrong about which value $\mathcal{A}$ will choose in his ith distinct query (if it is used again later, it will be repeated and thus not distinct), so $\mathcal{B}$ aborts. Otherwise, $\mathcal{B}$ chooses a random $w' \in \mathbb{Z}_q^*$. Let $y = e(H_p^{\frac{1}{s_p+\hat{x}}}, g_p)e(H_q, g_q)^{w'}$. Choose a random $r \leftarrow \mathbb{Z}_n^*$. Let $\pi_1 = (H_p^{r\frac{1}{s_p+\hat{x}}}/D_p^r)(H_q^{w'r}/D_q^r)$. Let $\pi_2 = g^{\frac{1}{r}}$ and $\pi_3 = A^{\frac{\hat{x}+s_p}{r}}$. If we implicitly set $w = w'h_q$, (where $H_q = g_q^{h_q}$) then these value will be distributed as in the output of *SimSample* and *SimProve*. Output $(y, \pi = (\pi_1, \pi_2, \pi_3))$.

If $j = i$: If $\hat{x} \neq x^*$, then $\mathcal{B}$ has guessed wrong, so it aborts. Otherwise, choose random $r_p \leftarrow \mathbb{Z}_p^*$. Implicitly set $r \in \mathbb{Z}_n^*$ to be the element such that $r \mod q = r^*$ and $r \mod p = r_p$. Compute $\pi_1 = H_p^{r_p \frac{1}{x^*+s_p}} / D_p^{r_p} Z_1$. Note that, if $Z_1 = H_q^{r^* \frac{1}{s_q+x^*}} / D_q^{r^*}$, then this is equal to $H^{\frac{r}{s+x^*}} / D^r$. Otherwise, this is equal to $H_p^{r_p \frac{1}{x^*+s_p}} / D_p^{r_p} g_q^{R_1}$. Now compute $\pi_2 = g_p^{\frac{1}{r_p}} Z_2$, if $Z_2 = g_q^{\frac{1}{r^*}}$, then this value will be $g^{\frac{1}{r}}$. Otherwise it will be $g_p^{\frac{1}{r_p}} g_q^{R_2}$ Compute $\pi_3 = A^{\frac{s_p+x^*}{r_p}} = A^{\frac{s+x^*}{r}}$. Finally, compute $y = e(\pi_1, \pi_2)e(D, g)$. Output $(y, \pi = (\pi_1, \pi_2, \pi_3))$ to the adversary.

If $j > i$, we know $\hat{x} \neq x^*$, and $\hat{x} \in X$. Let $V_1 = H_q^{r_{\hat{x}} \frac{1}{s_q+\hat{x}}} / D_q^{r_{\hat{x}}}$, and $V_2 = g^{\frac{1}{r_{\hat{x}}}}$, as provided in $\mathcal{B}$'s input. $\mathcal{B}$ chooses a random $r_p \leftarrow \mathbb{Z}_p^*$. Implicitly, set $r \in \mathbb{Z}_n^*$ for this query to be the element such that $r \mod p = r_p$, and $r \mod q = r_{\hat{x}}$. $\mathcal{B}$ computes $\pi_1 = (H_p^{r_p \frac{1}{s_p+\hat{x}}} / D_p^{r_p}) V_1 = H^{r \frac{1}{s+\hat{x}}} / D^r$, $\pi_2 = g_p^{\frac{1}{r_p}} V_2 = g^{\frac{1}{r}}$, and $\pi_3 = A^{\frac{s_p+\hat{x}}{r_p}} = A^{\frac{\hat{x}+s}{r}}$. Finally, $\mathcal{B}$ computes $y = e(\pi_1, \pi_2)e(D, g)$ and outputs $(y, \pi = (\pi_1, \pi_2, \pi_3))$ to $\mathcal{A}$.

Finally, $\mathcal{B}$ gets $\mathcal{A}$'s guess bit b. If $\mathcal{A}$ guesses that this is Game Hybrid $i-1$, $\mathcal{B}$ guesses that $(Z_1, Z_2) = (H_q^{r^* \frac{1}{s_q+x^*}} / D_q^{r_*}, g_q^{\frac{1}{r^*}})$; otherwise $\mathcal{B}$ guesses that $(Z_1, Z_2) = (g_q^{R_1}, g_q^{R_2})$. If $\mathcal{A}$ guesses correctly, $\mathcal{B}$'s guess will also be correct.

$\mathcal{B}$ has a $\frac{1}{l}$ probability of not aborting. Suppose that when $\mathcal{B}$ aborts, it returns a random bit. Then $\mathcal{B}$'s guess is correct with probability $(1 - \frac{1}{l}) * \frac{1}{2} + \frac{1}{l} * (\frac{1}{2} + \epsilon) = \frac{1}{2} + \frac{\epsilon}{l}$, where ϵ is $\mathcal{A}$'s advantage. Thus, if $\mathcal{A}$'s advantage is $\epsilon > \nu l$ then $\mathcal{B}$'s advantage is higher than ν, contradicting the assumption. □

□

For the theorem to follow, we observe that the overall reduction from breaking the simulatability game to breaking the BDHBI assumption uses at most $(l+1)$ hybrids, and so the adversary's advantage ϵ translates into the reduction's advantage ϵ / l^2 in breaking BDHBI. □

Remark. Since the construction above satisfies the premise of Lemma 8, it can be converted to an sVRF with binary range using the construction in Section 5.1.2.

5.2.2 Construction from DDHI and GS proofs

Here we present our new construction for sVRFs. Later, we will show that an extension of this construction (as described in sections 5.3 and 9.2) can be used to construct provably secure e-cash.

Our construction will be in the bilinear group setting where we are given p, g, G_1, G_2, G_T, e such that G_1, G_2, G_T are multiplicative groups of prime order p, g is a generator of G_1, h is a generator of G_2, and e is a bilinear map $e : G_1 \times G_2 \rightarrow G_T$. We will use the function $\mathsf{PRF}_s(x) = g^{\frac{1}{s+x}}$ to build an efficient Simulatable VRF [2] .

We will show that if $\mathsf{PRF}_s(x)$ is a pseudorandom function in this setting, and if the Groth-Sahai proof system is secure for these groups, then we can build an efficient t-sVRF with output range G_1. Note that the base function is similar to the Dodis-Yampolskiy VRF, which uses the function $\mathsf{PRF}_s(x) = e(g, h)^{\frac{1}{s+x}}$ and

[2]This function is also known as a Weak Boneh-Boyen signature [11]

thus gives output in G_T. Moving our function to output elements in G_1 is the crucial step which allows us to use the Groth-Sahai proof techniques.

Theorem 16. *Let $D_k \subset Z$ denote a family of domains of size polynomial in k. Let p, g, e, G_1, G_2, G_T be as described above where $|p| = k$. If the DDHI assumption holds in G_1, then the set $\{g^{\frac{1}{s+x}}\}_{x \in D_k}$ is indistinguishable from the set $\{g^{r_x}\}_{x \in D_k}$ where $s, \{r_x\}_{x \in D_k}$ are chosen at random from Z_p.* The proof is very similar to that in [43].

We will build an sVRF based on this function as follows:

$\mathsf{Setup}(1^k)$. Let $e : G_1 \times G_2 \rightarrow G_T$ be a bilinear map of order q. Let g be a generator for G_1, and h be a generator for G_2. Let $params_{GS}$ be the parameters for a Groth-Sahai NIZK proof system. These parameters define a perfectly binding commitment scheme $\mathsf{GSCommit}$ for committing to elements of G_1 and G_2. Output parameters $params_{VRF} = (q, G_1, G_2, G_T, g, h, params_{GS})$.

$\mathsf{Keygen}(params_{VRF})$. Pick a random seed $seed \leftarrow Z_p$ and random opening information $open_{seed}$ and output $sk = (seed, open_{seed})$ and public key $pk = \mathsf{GSCommit}(h^{seed}, open_{seed})$.

$\mathtt{Eval}(params_{VRF}, sk = (seed, open_{seed}), x)$. Compute $y = g^{1/(seed+x)}$.

$\mathsf{Prove}(params_{VRF}, sk = (seed, open_{seed}), x)$. Compute $y = g^{1/(seed+x)}$ and a corresponding commitment $C = \mathsf{GSCommit}(y, aux(y))$ from random opening $aux(y)$. Next create the following two proofs: π_1, a composable NIZK proof that C is a commitment to y using the Groth-Sahai techniques (see Section 3.5 for details); π_2, a GS composable witness indistinguishable proof that C is a commitment to Y and pk is a commitment to S such that $e(Y, Sh^x) = e(g, h)$. Output $\pi = (C, \pi_1, \pi_2)$.

$\mathsf{VerifyProof}(params, pk, x, y, \pi = (C, \pi_1, \pi_2))$. Use the Groth-Sahai verification to Verify π_1, π_2 with respect to C, x, pk, y.

Efficiency

The above proof protocol generates 1 new commitment in G_1, then produces one Groth-Sahai proof for a pairing product equation with $Q = 1$, and finally produces a zero-knowledge proof of equality of committed values in G_1 using the proof system in Section 3.5. Applying the efficiency formulas given in Sections 3.4 and 3.5, we get the following lemma:

Theorem 17. *When instantiated using the SXDH instantiation given in Section 3.3 the sVRF proofs will have the following efficiency: Generating the proof will require* 40 *exponentiations in* G_1 *and* 40 *exponentiations in* G_2*. The resulting proof will consist of* 14 *elements of* G_1 *and* 14 *elements of* G_2*. Verifying the proof will involve computing* 60 *bilinear group pairings.*

When instantiated using the symmetric setting version of the DLIN instantiation given in Section 3.3, the above proof system will have the following efficiency: Generating the proof will require 153 *exponentiations in* G*. The resulting proof will consist of* 33 *elements in* G*. Verifying the proof will involve computing* 108 *bilinear group pairings.*

Security

Theorem 18. *This construction with domain size q is a strong sVRF under the q-DDHI for G_1 and under the assumption that the Groth-Sahai proof system is secure.*

Proof. Correctness and Verifiability follow from the corresponding properties of the WI and NIZK GS proof systems.

Pseudorandomness can be shown via an approach very similar to our proof below for Simulatability, so we will not show it here.

Trapdoor-Indistinguishable Simulatability We define the following simulator algorithms:

$\mathsf{SimSetup}(1^k)$. Let $e : G_1 \times G_2 \to G_T$ be a bilinear map of order q. Let g be a generator for G_1, and h be a generator for G_2. Let $(params_{GS}, auxsim) \leftarrow \mathsf{GSSimSetup}(q, G_1, G_2, G_2, G_T, g, h)$ be simulated parameters for a Groth-Sahai NIZK proof system. Output parameters $params_{VRF} = (q, G_1, G_2, G_T, g, h, params_{GS})$ and trapdoor $t = sim$.

SimG$(params_{VRF})$. Pick a random seed $seed \leftarrow Z_p$ and random opening information $open_{seed}$ and output $sk = (seed, open_{seed})$ and public key $pk = \mathsf{GSCommit}(h^{seed}, open_{seed})$.

$\mathsf{SimProve}(params_{VRF}, sk = (seed, open_{seed}), x, y, t)$. Compute $y' = g^{\frac{1}{s+x}}$ and corresponding commitment $C = \mathsf{GSCommit}(y')$. Use the NIZK simulator to compute simulated proof π_1 that C is a commitment to y.

Create π_2 as an honest GS witness indistinguishable proof that C is a commitment to y' and pk is a commitment to S such that $e(y', Sh^x) = e(g, h)$

Output $\pi = (\pi_1, \pi_2)$.

Now we need to show that Game Simulated Proofs, when instantiated using this simulator, is indistinguishable from Game Real Proofs. We will do this by considering a series of intermediate games.

First consider the following intermediate simulator algorithm:

$\mathsf{HybSimProve}(params_{VRF}, sk = (seed, open_{seed}), x, t)$. Computes $y = g^{\frac{1}{s+x}}$, and then proceeds as $\mathsf{SimProve}$ does: It computes $y' = g^{\frac{1}{s+x}}$ and commitment $C = \mathsf{GSCommit}(y')$. It uses the NIZK simulator to compute simulated proof π_1 that C is a commitment to y.

It creates π_2 honestly: Create π_2 as an honest GS witness indistinguishable proof that C is a commitment to y' and pk is a commitment to S such that $e(y', Sh^x) = e(g, h)$

It outputs $\pi = (\pi_1, \pi_2)$.

Game Hybrid Sim 1. Runs as Game Real Proofs except that it uses $\mathsf{HybSimProve}$ instead of Prove:

$(\mathtt{p}, t) \leftarrow \mathit{SimParams}(1^k)$, $(pk, sk) \leftarrow G(\mathtt{p})$ and then $\mathcal{A}(\mathtt{p}, t, pk)$ gets access to the following oracle $\mathcal{R}$: On query x, $\mathcal{R}$ returns $y = \mathtt{Eval}(\mathtt{p}, sk, x)$ and $\pi \leftarrow \mathsf{HybSimProve}(\mathtt{p}, sk, x, t)$.

Game Hybrid Sim 1 is indistinguishable from Game Real Proofs by the zero knowledge property of the GS NIZK and by the perfect WI property of the GS WI proofs.

Now consider a second intermediate game:

Game Hybrid Sim 2. Runs as Game Simulated Proofs except that it computes y correctly:

$(\mathtt{p}, t) \leftarrow \textit{SimParams}(1^k)$, $(pk, sk = (seed, open_{seed})) \leftarrow \textit{SimG}(\mathtt{p}, t)$, and then $\mathcal{A}(\mathtt{p}, t, pk)$ gets access to the following oracle Sim: On query x, Sim (1) checks if x has previously been queried, and if so, computes $\pi \leftarrow \textit{SimProve}(\mathtt{p}, sk, x, y, t)$ for the stored y and returns (y, π); (2) otherwise, it computes $y = g^{\frac{1}{s+x}}$ and $\pi \leftarrow \textit{SimProve}(\mathtt{p}, sk, x, y, t)$, returns (y, π), and stores y.

First note that Game Hybrid Sim 2 is identical to Game Hybrid Sim 1. Next, we can see that by theorem 16, Game Hybrid Sim 2 is indistinguishable from Game Simulated Proofs. Thus, Game Simulated Proofs is indistinguishable from Game Real Proofs, and the simulatability property holds. $\square$

5.2.3 Construction Based on General Assumptions

In the common-random-string (CRS) model, sVRFs can be constructed from any one-way function and an unconditionally sound multi-theorem non-interactive zero-knowledge proof system ($\mathtt{NIZKProve}, \mathtt{NIZKVerify}$) for NP (we review the notion of NIZK in Section 8.1). Pseudorandom functions (PRFs) can be obtained from one-way functions [58, 50] (in the sequel, by $F_s(x)$ we denote a PRF with seed s and input x). In the CRS model, one-way functions also imply unconditionally binding computationally hiding non-interactive commitment [67] (in the sequel, denoted as $comm(x, q, r)$, where x is the value to which one commits, q is the public parameter, and r is the randomness). We describe the construction below. In the full version, we prove it is an sVRF.

$params$ Corresponding to the security parameter k, choose a common random string σ of length $\ell(k)$, where $\ell(k)$ bits suffice for multi-theorem NIZK
[7, 46, 56]. Choose a random $2k$-bit string q as the public parameter for the Naor commitment scheme. The parameters are $\mathtt{p} = (\sigma, q)$.

Domain and range The function has domain $D(\mathtt{p}) = \{0,1\}^{p_1(k)}$, and range $R(\mathtt{p}) = \{0,1\}^{p_2(k)}$, where p_1 and p_2 are functions bounded by a polynomial.

G Pick a random seed s for a pseudorandom function $F_s : \{0,1\}^{p_1(k)} \mapsto \{0,1\}^{p_2(k)}$. Let $pk = comm(s, q, r)$, $sk = (s, r)$, where r is the randomness needed for the commitment.

`Eval` On input x, output $y = F_s(x)$.

`Prove` On input x, run `NIZKProve` using CRS σ to output a NIZK proof π of the following statement: $\exists(s, r) \mid pk = comm(s, q, r) \wedge y = F_s(x)$.

`Verify` On input (pk, y, π), verify the proof π using the `NIZKVerify` algorithm.

5.3 NIZK Proofs of Pseudorandom Functions

In some applications, we need something stronger than an sVRF. The simulatability property of an sVRF guarantees that the public key and corresponding proofs will not compromise the pseudorandomness of the output values. However, they are not guaranteed to hide all information about the seed corresponding to the public key. In our ecash application, we need to be certain that the proofs will reveal no information about which wallet was used, which means that they should completely hide the seed used. Furthermore, we do not want to reveal which coin in the wallet is being spent, thus we also want to hide the input x.

What we really need is a pseudorandom function for which we can commit to a seed, and commit to an input, and then issue NIZK proofs that a particular value is the correct output for the given commitments. It turns out that the construction above can easily be extended to give these stronger properties.

As above, we will use the pseudorandom function $\mathsf{PRF}_s(x) = g^{\frac{1}{s+x}}$, where s is the seed, x is the input, and g is a group element. Thus, we need to be able to prove statements about the following language.

Let $\mathcal{L}_S(params)$ be the set of tuples C_s, C_x, y such that y is the correct output with respect to the x, s contained in C_s, C_x. I.e.

$$\begin{aligned}\mathcal{L}_S = \{&C_s, C_x, y \mid \exists x, s, aux(x), aux(s) \text{ such that}\\ &C_s = \mathsf{Commit}(s, aux(s)) \wedge C_x = \mathsf{Commit}(x, aux(x)) \wedge y = \mathsf{PRF}_s(x)\}\end{aligned}$$

Note that the commitments here are the same commitments to elements of Z_p that are also used by our P-signature constructions in Chapter 4. We use the same trick to turn a commitment to a group element into a commitment to $a \in Z_p$: $\mathsf{Commit}(a, aux(a)) = \mathsf{GSCommit}(h^a, aux(a))$ as described in Section 4.2.3.

Here our proof system will also use the Groth-Sahai Setup and $\mathsf{SimSetup}$ algorithms, and will satisfy the requirements for composable zero knowledge (see Section 8.1.) Thus, it can securely be combined with any other Groth-Sahai based proofs, and in particular, it can share its parameters with the P-signature scheme given in Section 4.2.3.

We build a NIZK proof system for this language. The construction is as follows:

$\mathsf{Setup}(1^k)$. Compute the parameters $params$ for the GS proof system.

$\mathsf{Prove}(params, C_s, C_x, y, s, aux(s), x, aux(x))$. We first form new commitments $C'_s = \mathsf{GSCommit}(h^s, aux(s)')$ and $C'_x = \mathsf{GSCommit}(h^x, aux(x)')$. Then we compute zero knowledge proof π_1 that C_s and C'_s are commitments to the same value and proof π_2 that C_x and C'_x are commitments to the same value using the techniques described in Section 3.5.

Next, we compute a commitment $C'_y = \mathsf{GSCommit}(y, aux(y))$ to y and a zero knowledge proof π_3 that C'_y is a commitment to y as in Section 3.5.

Finally, we compute a GS witness indistinguishable proof π_4 that the value committed to in C'_y is the correct output given the seed in C'_s and the input in C'_x, i.e. that C'_y is a commitment to Y, C'_s is a commitment to S, and C'_x is a commitment to X such that $e(Y, SX) = e(g, h)$. The final proof is $\pi = (C'_s, C'_x, C'_y, \pi_1, \pi_2, \pi_3, \pi_4)$.

$\mathsf{VerifyProof}(params, C_s, C_x, y, \pi = (C'_s, C'_x, C'_y, \pi_1, \pi_2, \pi_3, \pi_4))$. Uses the Groth-Sahai verification techniques to verify $\pi_1, \pi_2, \pi_3, \pi_4$ with respect to $C_s, C_x, y, C'_s, C'_x, C'_y$.

Efficiency

The proof system above generates 3 new commitments (1 in G_1 and 2 in G_2), Groth-Sahai proofs for 1 pairing product equations with $Q = 1$, and 3 zero-knowledge proofs of equality of committed exponents (see Section 3.5), 1 for a pair of commitments to an element of G_1, and 2 for commitments to elements of G_2. Applying the efficiency formulas given in Sections 3.4 and 3.5, we get the following lemma:

Theorem 19. *When instantiated using the SXDH instantiation given in Section 3.3 the above proof system will have the following efficiency: Generating the proof will require* 96 *exponentiations in* G_1 *and* 96 *exponentiations in* G_2*. The resulting proof will consist of* 34 *elements of* G_1 *and* 34 *elements of* G_2*. Verifying the proof will involve computing* 140 *bilinear group pairings.*

When instantiated using the symmetric setting version of the DLIN instantiation given in Section 3.3, the above proof system will have the following efficiency: Generating the proof will require 369 *exponentiations in G. The resulting proof will consist of* 81 *elements in G. Verifying the proof will involve computing* 252 *bilinear group pairings.*

Security

Theorem 20. *The proof system* Setup, Prove, VerifyProof *is a secure composable zero knowledge proof system for the language* $\mathcal{L}_S(params)$ *described above, where* $params$ *is output by* Setup.

Proof. Correctness and Soundness follow from the corresponding properties of the underlying proof systems. Thus, we need only show zero knowledge. Consider the following simulator:

$\mathsf{SimSetup}(1^k)$. runs the GS simulation setup to generate simulated parameters $params$ and trapdoor sim.

$\mathsf{SimProve}(params, sim, Com_x, Com_s, y)$. We first choose random $s', x' \leftarrow Z_p$, random opening information $aux(s)', aux(x)'$ and form new commitments $C'_s = \mathsf{GSCommit}(h^{s'}, aux(s)')$ and $C'_x = \mathsf{GSCommit}(h^{x'}, aux(x)')$.

Then we use the GS NIZK simulator to compute simulated zero knowledge proof π_1 that C_s and C'_s are commitments to the same value and simulated proof π_2 that C_x and C'_x are commitments to the same value using the techniques described in Section 3.5.

Next, we compute a commitment C'_y to $\mathsf{PRF}_{s'}(x')$ and use the GS NIZK simulator to generate simulated proof π_3 that C'_y is a commitment to y as in Section 3.5.

Finally, we compute a GS witness indistinguishable proof π_4 that the value committed to in C'_y is the correct output given the seed in C'_s and the input in C'_x. (Note that this statement is true given our choice of C'_y, C'_s, C'_x.)

The final proof is $\pi = (C'_s, C'_x, C'_y, \pi_1, \pi_2, \pi_3, \pi_4)$.

Note that when parameters are generated by SimSetup, the proof π_4 and the commitments C'_y, C'_s, C'_x generated by SimProve are distributed identically to those generated by Prove. Further, by the composable zero knowledge properties of the GS NIZK for equality of committed values, the simulated proofs π_1, π_2, π_3 will also be distributed identically to those generated by the honest Prove algorithm. Thus, SimSetup, SimProve as described here satisfy the definition of zero knowledge for Setup, Prove, VerifyProof.

$\square$

Chapter 6

Application to Non-Interactive Anonymous Credentials

Anonymous credentials [35, 41, 15, 65, 23, 24, 25] let Alice prove to Bob that Carol has given her a certificate. Anonymity means that Bob and Carol cannot link Alice's request for a certificate to Alice's proof that she possesses a certificate. In addition, if Alice proves possession of a certificate multiple times, these proofs cannot be linked to each other. Anonymous credentials are an example of a privacy-preserving authentication mechanism, which is an important theme in modern cryptographic research.

Anonymous credentials are an immediate consequence of P-signatures (and of CL-signatures [64]) as follows; Suppose there is a public-key infrastructure that lets each user register a public key. Alice registers unlinkable pseudonyms A_B and A_C with Bob and Carol. A_B and A_C are commitments to her secret key, and so they are unlinkable by the security properties of the commitment scheme. Suppose Alice wishes to obtain a certificate from Carol and show it to Bob. Alice goes to Carol and identifies herself as the owner of pseudonym A_C. They run the P-signature Issue protocol as a result of which Alice gets Carol's signature on her secret key. Now Alice uses the P-signature Prove protocol to construct a non-interactive proof that she has Carol's signature on the opening of A_B.

6.1 Anonymous Credentials Based on P-Signatures

Recall that the participants in any credential system are users (who obtain credentials), organizations (who grant credentials) and a certification authority (CA). The existence of a CA allows users and organizations to register public keys. This is necessary, even if all other transactions are anonymous. Instead of saying "Alice has a credential" which, in the digital world, is not a well-formed statement, one needs to be able to say "The owner of pk_{Alice} (i.e. whoever knows sk_{Alice}has a credential." Then, so long as we believe that Alice does not reveal her secret key to other entities, there is a reason to believe that indeed it is Alice who has the credential. Here, it does not matter if we are talking about anonymous credentials or non-anonymous ones: even when we don't care about Alice's anonymity, unless users take steps to protect her secrets, a digital credentials system cannot be very meaningful.

An anonymous credential system consists of the following protocols:

Setup System parameters $params$ are generated, users and organizations generate their public and secret keys (pk, sk) and register their public keys with the CA. We will refer to PKI as the collection of all the public keys, and to the *identity* of the user as pk, his public key. As a result of this registration step, a user (whose private input is his secret key) obtains his root credential C_{CA}.

Pseudonym registration As a result of this protocol, a user and an organization agree on a pseudonym (nym) N for the user. The user's private input is his (sk, pk) and his C_{CA}; the organization does not have any private input. Their common output is N. The user's private output is $aux(N)$, some auxiliary information that may be needed later.

Credential issue As a result of this protocol, a user obtains a credential from an organization without revealing his identity, just based on his pseudonym N. The user U's private input to the protocol is his (sk_U, pk_U, aux_N), the organization's private input is its secret key sk_O, the user's private output is the credential C.

Proof of possession of a credential Here, a user who is known to one organization, O_1 under pseudonym N_1, and to another, O_2, under pseudonym N_2, and a credential C_1 from O_1, proves to O_2 that he has a credential from O_1. The user's private input to this protocol consists of $(sk_U, pk_U, P_1, aux_{N_1}, aux_{N_2}, C_1)$, while the values N_2 and pk_{O_1} are public. The organization verifies the proof.

An anonymous credential system should satisfy unforgeability and anonymity.

Informally, unforgeability requires that (1) corresponding to each pseudonym there is a well-defined identity and (2) if a user with pseudonym P successfully convinces an honest organization that she possesses a credential from another honest organization O', then it must be the case that organization O' has issued a credential to some pseudonym P' such that the identity of P' is the same as that of P.

Anonymity, informally, requires that, even an adversary that corrupts the CA and any subset of the organizations and users cannot distinguish the following two situations (1) it receives honestly generated public parameters, and is interfacing with honest users who obtain and show credentials as directed by the adversary; (2) it receives a different set of parameters, and is interfacing with users who obtain and show credentials as directed by the adversary, but instead of using the correct protocol for showing their credentials, they use a simulator algorithm that does not receive any inputs whose distribution depends on the identity of the user.

We now proceed to describe how an anonymous credential scheme can be constructed from P-signatures. Note that the reason that this scheme can be preferable to known schemes is that the proof of possession of a credential is non-interactive.

Suppose we are given a P-signature scheme. Then consider the following construction for an anonymous credential system:

Setup The system parameters $params$ are the parameters for the P-signature scheme. Note that they also include the parameters for Commit.

A user U's secret key sk_U will be chosen from the message space of the signature scheme (which coincides with the message space of the commitment scheme). The user's public key will be $pk_U = \mathsf{PublicKey}(sk_U)$ for an appropriately defined function PublicKey.

Organizations (including the CA) will generate their key pairs using the key generation algorithm of the P-signature scheme.

The CA credential will be issued as follows:

1. The user forms his pseudonym with the CA, $N_{CA} = \mathsf{Commit}(params, sk_U, open)$ for an appropriately chosen $open$. (Note that, since the commitment scheme is perfectly binding, this automatically guarantees that the identity associated with this pseudonym is well-defined.)
2. The user proves that he has committed to a sk such that his $pk_U = \mathsf{PublicKey}(sk)$ using an appropriate designated verifier [59] non-malleable [62] interactive proof.
3. The user and the CA run the protocol for obtaining a signature on a committed value (i.e. they run the ObtainSig and IssueSig protocols, respectively).

Pseudonym registration The user forms his pseudonym by forming a commitment to his secret key: for an appropriately chosen $open$, $N = \mathsf{Commit}(params, sk_U, open)$. (Again, since the commitment scheme is perfectly binding, this automatically guarantees that the identity associated with this pseudonym is well-defined.) The user proves that he has a credential from the CA for this pseudonym (as described below). The user then sends N to the organization and proves knowledge of $(sk, open)$ using an appropriate designated verifier non-malleable interactive proof.[1]

The user's private output $aux(N) = open$.

Credential issue The user U and the organization O run ObtainSig and IssueSig, respectively. The user's input is $(params, pk_O, sk_U, N, aux(N))$, while the organization's input is $(params, sk_O, N)$. As a result, the user obtains a signature σ on his sk_U, and so his credential is $C = \sigma$.

Proof of possession of a credential The user has a credential $C = \sigma_{O_1}(sk_U)$. He is known to organization O_2 as the owner of the pseudonym N. He needs to issue a non-interactive proof that a credential has been issued to the owner of N. This is done as follows:

1. Compute $(comm, \pi_1, open) \leftarrow \mathsf{Prove}(params, pk_{O_1}, sk_U, C)$.
2. Compute $\pi_2 \leftarrow \mathsf{EqCommProve}(params, sk_U, open, aux(N))$. (Where EqCommProve is explained in Section 3.5. It is a non-interactive proof that the two commitments $comm$ and N are to the same value.)
3. Output $(N, comm, \pi_1, \pi_2)$.

[1]This ensures that, at registration time, the entity registering the pseudonym knows the secret key associated with this pseudonym, so that, for example, Alice could not get Bob to commit to his secret key and prove to her that he knows it, only to then have Alice use this commitment as her own pseudonym with another organization.

6.2 Efficiency

We now consider the efficiency of the protocol for proving possession of a credential. The construction above is very simple – it generates one P-signature proof for signature on a single message, and then it generates one zero-knowledge proof of equality of committed values in G_2 as in Section 3.5.

Since we are not concerned with small key size or with signature on multiple messages, the two logical P-signature constructions would be those described in Sections 4.2.1 and 4.2.2. The construction in Section 4.2.1 is more efficient, while the construction in Section 4.2.2 is based on a seemingly much weaker assumption. Thus, we will consider both options, in the efficiency theorems below:

Theorem 21. *Consider the construction resulting from using the P-signature scheme given in Section 4.2.1.*

When instantiated using the SXDH instantiation given in Section 3.3 the above proof system will have the following efficiency: Generating the proof will require 64 *exponentiations in* G_1 *and* 56 *exponentiations in* G_2*. The resulting proof will consist of* 22 *elements of* G_1 *and* 18 *elements of* G_2*. Verifying the proof will involve computing* 84 *bilinear group pairings.*

When instantiated using the symmetric setting version of the DLIN instantiation given in Section 3.3, the above proof system will have the following efficiency: Generating the proof will require 234 *exponentiations in G. The resulting proof will consist of* 48 *elements in G. Verifying the proof will involve computing* 153 *bilinear group pairings.*

Theorem 22. *Consider the construction resulting from using the P-signature scheme given in Section 4.2.2.*

When instantiated using the SXDH instantiation given in Section 3.3 the above proof system will have the following efficiency: Generating the proof will require 84 *exponentiations in* G_1 *and* 76 *exponentiations in* G_2*. The resulting proof will consist of* 28 *elements of* G_1 *and* 24 *elements of* G_2*. Verifying the proof will involve computing* 108 *bilinear group pairings.*

When instantiated using the symmetric setting version of the DLIN instantiation given in Section 3.3, the above proof system will have the following efficiency: Generating the proof will require 315 *exponentiations in G. The resulting proof will consist of* 63 *elements in G. Verifying the proof will involve computing* 198 *bilinear group pairings.*

6.3 Proof of Security

We must now show that the resulting anonymous credentials scheme is secure.

6.3.1 Unforgeability

Theorem 23. *The credentials scheme described above is unforgeable given the security of the P-signatures.*

Proof. (Sketch) Recall that the commitment scheme is perfectly binding. Therefore, corresponding to any setting of $params$, and any commitment N, there is exactly one value sk and opening $open$ such that $N = \mathsf{Commit}(params, sk, open)$, and exactly one corresponding value $pk = \mathsf{PublicKey}(sk)$. Therefore, with the pseudonym N, we can associate the identity pk, and (1) is satisfied. To satisfy (2), first suppose that the credential system is not unforgeable. Then we set up a reduction that breaks unforgeability of the P-signature scheme. Let F be the bijection that satisfies the unforgeability definition, and let $\mathsf{ExtractSetup}, \mathsf{Extract}$ be the corresponding extractor. The reduction will be given $params$ as output by $\mathsf{ExtractSetup}$, a public key pk, and access to a signing oracle. The reduction will make pk the public signing key of an arbitrary organization O under it's control. It will generate the keys for all other entities under its control correctly. Finally, it will make a random guess i that the adversary's ith proof of a credential O will be a forgery. Since the pseudonym registration protocol includes an (interactive) proof of knowledge of the opening to a commitment, every times the adversary wishes to register a pseudonym N with O, the values $(sk_A, open)$ such that $N = \mathsf{Commit}(params, sk_A, open)$ can be extracted using the knowledge extractor. Every time the adversary wishes to obtain a credential from an organization other than O, the reduction interacts with the adversary using the correct protocol. When the adversary, using pseudonym N, wishes to obtain a credential from O, the reduction already knows the values $(sk_A, open)$ such that $N = \mathsf{Commit}(params, sk_A, open)$. So it queries its signing oracle to obtain $\sigma \leftarrow \mathsf{Sign}(params, sk, sk_A)$, and then invokes $\mathsf{SimIssue}$ instead of $\mathsf{IssueSig}$. (Note that $\mathsf{SimIssue}$ does not take any additional values, its simulation is based on rewinding the adversary.) The ith time the adversary produces a proof of possession of a credential from organization O consisting of $(N', comm, \pi_1, \pi_2)$, the reduction outputs π_1.

Now we analyze the reduction's probability of success. Note that the adversary's view is independent of i, O. If the reduction has guessed i, O correctly, and if the adversary's credential forgery is successful, then the identity defined by N' has not been granted a credential by O, but the credential proof will verify successfully. This means that $\mathsf{VerEqComm}(params, comm, N', \pi_2) = 1$ and $\mathsf{VerifyProof}(params, pk, comm, \pi_1) = 1$. Since $(\mathsf{EqCommProve}, \mathsf{VerEqComm})$ is perfectly sound, we know that $comm, N'$ are both commitments to the same value x. Since this is a forgery, we know O never issued a credential to the identity represented by $comm, N$, which means we have never queried our signing oracle on the committed value x. This means that when extractor extracts y, σ from π, $comm$, either $F^{-1}(y) \neq x$, or $\mathsf{VerifySig}(params, pk, F^{-1}(y), \sigma) =$ reject, or $\mathsf{VerifySig}(params, pk, F^{-1}(y), \sigma) =$ accept and $F^{-1}(y) = x$ and $x \notin Q_{\mathsf{Sign}}$. In all cases, we break the unforgeability property.

Note that the reduction above implies unforgeability even as the adversarially controlled users talk to multiple organizations. However, each organization may only talk to one user at a time, because the reduction must extract the opening of the commitment (pseudonym) of the user wishing to obtain a credential from O, and it needs to rewind the adversary for that to happen. Similarly each organization must execute issue protocols sequentially. This is OK only if the adversary is never rewound to a point in time that happened before the last query to the signing oracle (because the signing oracle cannot be rewound). □

6.3.2 Anonymity

Theorem 24. *The credentials scheme described above is anonymous given the security of the P-signatures.*

Proof. (Sketch) Recall that we must show that no adversary can distinguish a real execution from one in which it is interfacing with users who, when obtaining and showing credentials do not use the correct protocols, but instead use a simulator algorithm that does not receive any inputs whose distribution depends on the identity of the user.

We now describe a series of hybrid experiments.

In hybrid experiment H_0, the adversary is interfacing with users and organizations carrying out the real protocols.

In hybrid experiment H_1, the parameters $params$ are generated using $\mathsf{SimSetup}(1^k)$. (Recall that SimSetup generates parameters for the commitment scheme that result in an information theoretically hiding commitment scheme.) Other than that, the adversary is interfacing with users and organizations carrying out the real protocols. The adversary's view in H_1 is indistinguishable from his view in H_0 because otherwise we could distinguish $params$ generated using Setup from those generated by SimSetup.

In hybrid experiment H_2, the parameters $params$ and the value sim are generated using SimSetup. The honest organizations with which the adversary is interfacing are carrying out the real protocols. The honest users will always form their pseudonyms correctly, but in the zero-knowledge proof of knowledge protocol that accompanies the registration (both the registration with the CA and the registration with other adversarial organizations), the users use the zero-knowledge simulator for that proof and not the actual proof protocol. Hybrid H_2 gives the adversary an indistinguishable view as that in hybrid H_1 because otherwise we contradict the zero-knowledge property of the zero-knowledge proof system.

Recall that, in addition to $params$, SimSetup also generates sim. The knowledge of sim is empowering in several important ways. The knowledge of sim allows one to (1) compute simulated proofs of equality of committed values (i.e. simulate EqCommProve), and (2) simulate a proof that the committed value has been signed (recall the zero-knowledge part of our definition of P-signatures).

In hybrid experiment H_3, the only difference from H_2 is that honest users prove equality of committed values using the SimEqComm instead of using EqCommProve. This should be indistinguishable from H_2 by the zero knowledge property of the P-signature.

In hybrid experiment H_4, the only difference from H_3 is that honest users generate proofs that committed values have been signed using SimProve instead of Prove. Note that this means they no longer have the opening of the resulting commitment $comm$. However, as we are now using SimEqComm, we no longer need this opening. If this makes any difference to the adversary's view, then we again break the zero-knowledge property of the P-signature.

In hybrid experiment H_5, the only difference from H_4 is that honest users obtain signatures from adversarial organizations using SimObtain instead of ObtainSig. (Note that they do not need to obtain the real signatures because they never use them, since their proofs that a commitment has been signed are always simulated.) If this makes any difference to the adversary's view, then it is easy to show that the user privacy part of the definition of security for P-signatures is broken.

In hybrid experiment H_6, the only difference from H_5 is that when honest users register pseudonyms, then commit to 1 instead of committing to their secret keys. Note that the view that the adversary gets as a result is the same as the view he gets in H_5, because the commitments are information-theoretically hiding, and all the proofs are simulated. Also note that in this experiment, the honest users run only protocols that never take users' identities as input. Therefore, we have obtained the desired simulator. □

Chapter 7

Application to Delegatable Anonymous Credentials

One of the most common uses of cryptography today is access control: does the person requesting access to a resource possess the required credentials? A credential typically consists of a certification chain rooted at some authority responsible for managing access to the resource and ending at the public key of a user in question, who then needs to demonstrate that he knows the corresponding secret key. The simplest case, when the trusted authority issues certificates *directly* to each user (so the length of each certification chain is 1), is inconvenient, because it requires the authority to do too much work. A system in which the authority delegates responsibility to other entities is more convenient: an entity with a certification chain of length ℓ can issue certification chains of length $\ell + 1$. A conventional signature scheme immediately allows delegatable credentials: Alice, who has a public signing key pk_A and a certification chain of length ℓ, can sign Bob's public key pk_B, and now Bob has a certification chain of length $\ell + 1$.

The design of an anonymous delegatable credential scheme in which participants can obtain, delegate and demonstrate possession of credential chains without revealing any additional information about themselves, is a natural and desirable goal. Our main contribution is a solution to this problem which, until now, has proved elusive: no fully delegatable anonymous credential schemes were previously known. The only known construction of delegatable anonymous credentials, due to Chase and Lysyanskaya [31], needed $k^{\Omega(\ell)}$ space to store a certification chain of length ℓ (for security parameter k), and therefore could not tolerate non-constant ℓ. In contrast, our solution is *practical*: all operations on chains of length ℓ will need $\Theta(k\ell)$ time and space.

Let us now explain why this was a challenging problem. There is no straightforward transformation of anonymous credential schemes without delegation [35, 41, 15, 65, 23, 24, 25, 5] into delegatable schemes. The main building block in research [64, 24, 25, 5] on anonymous credentials was signature schemes that lend themselves to the design of efficient protocols for (1) obtaining a signature on a committed value (so a user can obtain a credential from the authority on his committed secret key); and (2) proving that a committed value has been signed (so a user can prove that his committed secret key has been signed by the authority).

To form a proof of possession of a credential, a user needs to know a signature on his committed secret key. Generalizing this building block to delegation chains, a user would need to know an entire delegation chain authorizing his committed secret key. A conventional delegation chain is just a chain of signatures, and so knowing it means that the user also knows the identity of all the intermediate signers — the delegators cannot be anonymous.

Thus, the old approach does not yield itself to delegation, and we must try something very different. Our main tool will be non-interactive zero-knowledge proofs of knowledge with certain desirable properties. Let's say Oliver is the authority originating a particular type of credential with public key pk_O and secret key sk_O; let's say Alice is a user with secret key sk_A, and she wants to obtain the credential directly from Oliver (so her certification chain will be of length 1). Under the old approach, they would run a secure two-party protocol as a result of which Alice obtains a signature $\sigma_{pk_O}(sk_A)$ on sk_A, while Oliver gets no output. Under the new approach, she does not get such a signature — rather, her output is $(comm_A, \pi_A)$ where $comm_A$ is a commitment to her secret key sk_A, and π_A is a *proof of knowledge* of Oliver's signature on the contents of $comm_A$.

How can Alice use this credential anonymously? If the underlying proof system is *malleable* in just the right way, then given $(comm_A, \pi_A)$ and the opening to $comm_A$, Alice can compute $(comm'_A, \pi'_A)$ such that $comm'_A$ is another commitment to her sk_A that she can successfully open, while π'_A is a proof of knowledge of Oliver's signature on the contents of $comm'_A$. Malleability is usually considered a bug rather than a feature. But in combination with the correct extraction properties, we still manage to guarantee that these randomizable proofs give us a useful building block for the construction. The bottom line is that $(comm'_A, \pi'_A)$ should not be linkable to $(comm_A, \pi_A)$, and also it should not be possible to obtain such a tuple without Oliver's assistance.

Next, how can Alice delegate her credential to Bob? First, we need the commitment $comm'_A$ to essentially serve a double purpose as a signature public key. Alice and Bob can run a secure protocol as a result of which Bob obtains $(comm_B, \pi_B)$ where $comm_B$ is a commitment to Bob's secret key sk_B and π_B is a proof of knowledge of a signature issued by the owner of $comm'_A$ on the contents of $comm_B$. Now, essentially, the set of values $(comm'_A, comm_B, \pi'_A, \pi_B)$ together indicate that the owner of $comm'_A$ got a credential from Oliver and delegated to the owner of $comm_B$, and so it constitutes a proof of possession of a certification chain. Moreover, it hides the identity of the delegator Alice! Now Bob can, in turn, use the randomization properties of the underlying proof system to randomize this set of values so that it becomes unlinkable to his original pseudonym $comm_B$; he can also, in turn, delegate to Carol.

It may be somewhat counter-intuitive that despite the fact that the proofs are malleable no adversary can forge a proof of possession of a certification chain. The explanation here is that the proof system is perfectly *extractable*, and so in fact a certification chain can be extracted from a proof of possession of a certification chain. Therefore an adversary that succeeds in faking a proof of possession of a certification chain would in fact yield an attack on the unforgeability properties of the underlying signature scheme.

(Note that in this informal introduction we are omitting important details that have to do with securely realizing certification chains of this type, such as (1) how to make it impossible for adversarial users to mix and match pieces of different certification chains to create unauthorized certification chains; (2) how to define and construct a signature/authentication scheme that remains secure even when an adversary can not only query honest users for signatures on chosen messages, but also cause honest users to query for adversary's signature on their secret keys.)

Our main contributions are that we (1) define and construct extractable and randomizable proofs of knowledge of a witness to a certain class of relations based on the recent proof system for pairing product equations due to Groth and Sahai [57]; and (2) define and construct a delegatable anonymous credential system based on this and other building blocks. Our construction is efficient whenever the building blocks can be realized efficiently. We also give efficient instantiations of the building blocks under appropriate assumptions about bilinear groups.

7.1 Definition of Delegatable Credentials

An anonymous delegatable credential system has only one type of participant: users. An originator O of a certain type of credential can register a pseudonym Nym_O as its public key to act as credential authority. Users interact with other users, including authorities, using many different pseudonyms. Thus a user A can be known to authority O as $Nym_A^{(O)}$ and to user B as $Nym_A^{(B)}$. If authority O issues user A a credential for $Nym_A^{(O)}$, then user A can prove to user B that $Nym_A^{(B)}$ has a credential from authority O. We say that credentials received directly from the authority are level 1 credentials or basic credentials, credentials that have been delegated once are level 2 credentials, and so on. Thus user A can also delegate its credential to user B, and user B can then prove that he has a level 2 credential from authority O. Thus, a delegatable credential system consists of the following algorithms:

$\mathsf{Setup}(1^k)$. Outputs the trusted public parameters of the system, $params_{DC}$.

$\mathsf{Keygen}(params_{DC})$. Creates the secret key of a party in in the system.

$\mathsf{Nymgen}(params_{DC}, sk)$. Outputs Nym and auxiliary info $aux(Nym)$ for secret key sk.

$\mathsf{VerifyAux}(params_{DC}, Nym, sk, aux(Nym))$. Outputs accept iff Nym is a valid pseudonym for sk, $aux(Nym)$.

$\mathsf{NymProve}(params_{DC}, sk, Nym, aux(Nym)) \leftrightarrow \mathsf{NymVerify}(params_{DC}, Nym)$. Interactive algorithms for proof of pseudonym possession. The prover inputs $(params_{DC}, sk, Nym, aux(Nym))$, where Nym is a pseudonym for sk with auxiliary info $aux(Nym)$. The prover runs NymProve and gets no output. The verifier inputs $(params_{DC}, Nym)$, runs NymVerify and outputs accept or reject.

$\mathsf{Issue}(params_{DC}, Nym_O, sk_I, Nym_I, aux(Nym_I), cred, Nym_U, L)$

$\leftrightarrow \mathsf{Obtain}(params_{DC}, Nym_O, sk_U, Nym_U, aux(Nym_U), Nym_I, L)$. Interactive algorithms for an issuing protocol between an issuer and a user. The issuer inputs $(params_{DC}, Nym_O, sk_I, Nym_I, aux(Nym_I), cred, Nym_U, L)$, where Nym_O is the authorities public key, sk_I is the issuer's secret key, Nym_I is the issuer's pseudonym with auxiliary information $aux(Nym_I)$, $cred$ is the issuer's level L credential rooted at Nym_O, and Nym_U is the user's pseudonym. If $Nym_I = Nym_O$, then the issuer is the authority responsible for this credential, so $L = 0$ and $cred = \epsilon$. The issuer runs Issue and gets no output. The user inputs $(params_{DC}, Nym_O, sk_U, Nym_U, aux(Nym_U), Nym_I, L)$, where Nym_O identifies the authority responsible for this type of credential, sk_U is the user's secret key, $aux(Nym_U)$ is the auxiliary information corresponding to the user's pseudonym Nym_U, and L is the level of the issuer's credential. The user runs Obtain and gets a credential $cred_U$ as output.

CredProve$(params_{DC}, Nym_O, cred, sk, Nym, aux(Nym), L)$**.** Takes as input a level L credential $cred$ from authority Nym_O, outputs a value $credproof$.

CredVerify$(params_{DC}, Nym_O, credproof, Nym, L)$**.** Outputs accept if $credproof$ is a valid proof that the owner of pseudonym Nym possesses a level L credential from authority Nym_O. Outputs reject otherwise.

Discussion of the Anonymity Property. We require that the adversary's interactions with the honest parties in the real game should be indistinguishable from some ideal game in which pseudonyms, credentials and proofs are truly anonymous. Specifically, there should be an alternative setup algorithm SimSetup which produces parameters indistinguishable from those output by Setup, along with some simulation trapdoor sim. Under the simulated parameters, for any secret key sk, the distributions of pseudonyms, credentials and proofs for this secret key should be identical so the distribution output by simulators (SimIssue, SimObtain, SimProve) that take sim as input but do not take sk as input. Our definition, somewhat in the spirit of the composable zero knowledge definition given in [57], requires that each individual protocol (SimIssue, SimObtain, SimProve), when run on a single adversarially chosen input (and not only in the oracle described above), produces output indistinguishable from the corresponding real protocol, even when the adversary is given the *simulation trapdoor*. A simple hybrid argument shows that this implies the more complex but weaker definition described above. This stronger definition is much easier to work with as we need only consider one protocol at a time, and only a single execution of each protocol, thus it is the one we will present below.

Discussion of the Unforgeability Property. On a high level, we want to define an ideal world, indistinguishable from the real world, in which the adversary clearly cannot generate a forgery. Here we will have all of the honest parties controlled by a single oracle, and we will keep track of all honestly issued credentials. Then we will require that an adversary given access to this oracle should have only negligible probability of outputting a forged credential.

DEFINING A FORGERY. First note that, in order for unforgeability of credentials to make sense, we have to define it in a setting where pseudonyms are completely binding, i.e. for each pseudonym there is exactly one valid corresponding secret key. The question is, what exactly constitutes a forgery? As a first attempt, we might say that A succeeds in forging a credential if he can prove that some Nym_A has been issued a credential if such a credential was never issued for pseudonym Nym_A. But just because such a credential was never issued to Nym_A, that does not mean that it wasn't issued to another pseudonym for the same user. Thus, instead, we can say that there exists some (potentially exponential) extraction which takes as input a pseudonym and outputs the corresponding secret key. Now we can specify a forgery as an instance where A can prove that Nym_A has a credential when such a credential was never issued to any pseudonym for $sk_A = \mathsf{Extract}(Nym_A)$. In fact, it is sufficient for our purposes if Extract produces $F(sk_A)$ for some bijection F, and so we get F-extraction.

Now, how do we formalize the notion that such a credential was never issued? We say a forgery is when the adversary produces a proof of a level L credential with authority O from which we extract $sk_1, \dots, sk_{L-1}, sk_A$ such that a level L credential rooted at O was never delegated by sk_{L-1} to sk_A. Thus, we are not concerned with exactly which set $sk_2, \dots, sk_{L-2}$ are extracted. In practical terms, this means that once sk_{L-1} has delegated a level L credential from authority O to sk_A, we don't care if the adversary can forge credentials with different credential chains as long as they have the same level, are from the same authority, and are for the same sk_A.

Of course, this only makes sense if sk_{L-1} belongs to an honest user; otherwise we have no way of knowing what credentials he issued. But what if the owner of sk_{L-1} is adversarial and the owner sk_{L-2} is honest? Then the owner of sk_A should be able to prove possession of a credential if and only if sk_{L-2} delegated a level $L-1$ credential rooted at authority O to user sk_{L-1}. Generalizing this idea, our definition says a forgery is successful if we extract $sk_0, \dots, sk_L$ such that there is a prefix $sk_O, \dots, sk_i$ such where sk_{i-1} is honest, but sk_{i-1} never issued a level i credential from root O to sk_i.

DEFINING THE GAME We need to capture unforgeability in all possible situations. The most general way to do this is to give the adversary access to an oracle through which he can direct the rest of the system. He can ask that new honest users and new honest pseudonyms be created, or that credentials be delegated between honest users. Further he can ask that credentials be delegated between honest and adversarial users, in which case he participates in the delegation protocol. The resulting oracle becomes very complex. So instead, we put the adversary in charge of keeping track of each honest user's credentials and pseudonyms (but, of course, not their secret keys). For example, the adversary will give the oracle a pseudonym and a credential and ask him to delegate to another pseudonym. If the pseudonym belongs to an honest party and the credential is correct, the oracle will oblige and will give the resulting delegated credential to the adversary. Note that this definition is strictly stronger than that with the more general oracle described above.

We will first present a sketch of the security definitions for delegatable anonymous credentials. Then we will give more formal definitions in Section 7.2.

7.1.1 Definitions of Delegatable Anonymous Credentials: A Sketch

(We say that a function $\nu : \mathbb{Z} \to \mathbb{R}$ is negligible if for all integers c there exists an integer K such that $\forall k > K$, $|\nu(k)| < 1/k^c$. We use the standard GMR [54] notation to describe probability spaces.) A credential system with efficient algorithms (Setup, Keygen, Nymgen, $\mathsf{NymProve}$, $\mathsf{NymVerify}$, $\mathsf{VerifyAux}$, Issue, Obtain, $\mathsf{CredProve}$, $\mathsf{CredVerify}$) constitute a secure anonymous delegatable credential scheme if the following properties hold (For more formal definition, see Appendix 7.2):

Correctness. We say that a credential *cred* is *a proper level L credential* from authority O for sk with respect to $params_{DC}$ if, for all pseudonyms Nym for sk, when $\mathsf{CredProve}$ uses this credential to compute a proof for Nym, $\mathsf{CredVerify}$ always accepts. We require the following properties:

(a). $\mathsf{Obtain}(params_{DC}, Nym_O, sk_U, Nym_U, aux(Nym_U), Nym_I, L)$ always outputs a proper level $L+1$ credential from authority O for sk_U or aborts.

(b). $\mathsf{Issue}(params_{DC}, Nym_O, sk_I, Nym_I, aux(Nym_I), cred, Nym_U, L)$ aborts without starting any communication if $cred$ is not a proper L credential from authority O for sk_I, or if Nym_U is not a valid pseudonym, or if Nym_I, $aux(Nym_I)$ are not consistent with sk_I. If all these values are correct, and if Issue is interacting with an honest Obtain with the appropriate inputs, then Obtain will output a valid credential.

(c). $\mathsf{CredProve}(params_{DC}, Nym_O, cred, sk, Nym, aux(Nym), L)$ aborts with no output if $cred$ is not a proper level L credential from O for sk, or if Nym, $aux(Nym)$ are not consistent with sk. If all these are correct, then CredProve interacting with CredVerify with the appropriate inputs, causes CredVerify to accept.

(d). VerifyAux always accepts pseudonyms generated by Nymgen.

Anonymity To capture that pseudonyms and proofs reveal no information about the user's secret keys or his credentials, we require that there exists a simulator (SimSetup, SimProve, SimObtain, SimIssue) such that:

(a). The public parameters generated by SimSetup are indistinguishable from those output by Setup.

(b). When generated using the parameters output by SimSetup, Nym is distributed independently of sk.

(c). The simulator SimProve can output a fake credential proof $credproof$ that cannot be distinguished from a real credential proof, even when SimProve is only told the authority, length of the credential chain, and the pseudonym of the user (it is not given the user's secret key, or his private credentials).

(d). The adversary cannot tell if it is interacting with Obtain run by an honest party with secret sk, or with SimObtain that is only given the authority, length of the credential chain, and the pseudonyms of the issuer and user (but not sk).

(e). The adversary cannot tell if it is interacting with Issue run by an honest party with a valid credential, or with SimIssue which is not given the credential and the issuer's secret key, but only told the authority, length of the credential chain, and the pseudonyms of the issuer and user.

Security of NymProve**.** The algorithm NymProve must be a zero knowledge proof of knowledge of sk, $aux(Nym)$ such that $\mathsf{VerifyAux}(params_{DC}, Nym, sk, aux(Nym)) = \mathsf{accept}$.

F**-Unforgeability.** Let F be an efficiently computable bijection and a one-way function. There exists a PPT ExtSetup, and a deterministic but potentially unbounded Extract with five properties:

(a). The parameters generated by ExtSetup are distributed identically to those generated by Setup.

(b). Under these parameters pseudonyms are perfectly binding, i.e. for any Nym, there exists at most one sk for which there exists $aux(Nym)$ such that $\mathsf{VerifyAux}(params_{DC}, Nym, sk, aux(Nym)) =$ accept

(c). Given an honestly generated level L credential proof, Extract can always extract the correct chain of L identities. I.e. if the credential is formed by using sk_0 to delegate to sk_1 which delegates to sk_2

and so on until sk_L, $\mathsf{Extract}$ will produce $(f_0, \ldots, f_L) = (F(sk_0), \ldots, F(sk_L))$. Note that this must hold for level 0 as well: for any valid pseudonym Nym_O, $\mathsf{Extract}$ will produce $f_0 = F(sk_O)$ where sk_O corresponds to Nym_O.

(d). Given an adversarially generated level L credential proof *credproof* from authority Nym_O for pseudonym Nym, $\mathsf{Extract}$ will always produce either the special symbol $\perp$ or $f_0, \ldots f_L$ such that Nym_O is a pseudonym for $F^{-1}(f_0)$ and Nym is a pseudonym for $F^{-1}(f_L)$.

(e). No adversary can output a valid credential proof from which an unauthorized chain of identities is extracted:

$$\begin{aligned}
&\Pr[(params_{DC}, td) \leftarrow \mathsf{ExtSetup}(1^k);\\
&\quad (credproof, Nym, Nym_O, L), \leftarrow \mathcal{A}^{\mathcal{O}(params_{DC},\cdot,\cdot)}(params_{DC}, td);\\
&\quad (f_0, \ldots, f_L) \leftarrow \mathsf{Extract}(params_{DC}, td, credproof, Nym, Nym_O, L) :\\
&\quad \mathsf{CredVerify}(params_{DC}, Nym_O, credproof, Nym, L) = \mathsf{accept} \wedge\\
&\quad (\exists i \text{ such that } (f_0, i, f_{i-1}, f_i) \notin \mathsf{ValidCredentialChains} \wedge f_{i-1} \in \mathsf{HonestUsers})] \leq \nu(k)
\end{aligned}$$

where $\mathcal{O}(params_{DC}, command, input)$ responds to the following types of queries as follows: The oracle simulation interaction with all honest parties. The adversary can ask to add an honest user, in which case the oracle generates and stores a secret key sk and returns a handle $F(sk)$ to the adversary. The adversary can the oracle to form a new pseudonym for an honest user, in which case the oracle stores the auxiliary information, and returns the resulting pseudonym. The adversary can ask one honest user to issue delegate a given credential to another honest user. In this case, we use $\mathsf{Extract}$ to extract the corresponding identity chain $f_0, \ldots, f_L$, record that the credential has been issued, and then return the credential. The adversary can also ask an honest user to delegate a credential to an adversarial user. This proceeds as in the previous case, except that the adversary participates in the delegation protocol. Similarly, the adversary can have an adversarial user delegate to an honest user. Finally, the adversary can ask an honest user to issue a proof for a given credential.

7.2 Formal Definition of Delegatable Credentials

Correctness. We say that a credential *cred* is *a proper level L credential* for organization Nym_O with respect to $(params_{DC}, sk)$ if

$$\begin{aligned}
&\Pr[Nym, aux(Nym) \leftarrow \mathsf{Nymgen}(params_{DC}, sk);\\
&\qquad credproof \leftarrow \mathsf{CredProve}(params_{DC}, Nym_O, cred, sk, Nym, aux(Nym), L) :\\
&\qquad\qquad \mathsf{CredVerify}(params_{DC}, Nym_O, credproof, Nym, L) = \mathsf{accept}] = 1.
\end{aligned}$$

We require the following property: (a). Obtain always outputs a proper credential (in the sense above) or aborts. More formally: For all adversaries $\mathcal{A}_1, \mathcal{A}_2$,

$$\begin{aligned}
&\Pr[\mathit{params}_{DC} \leftarrow \mathsf{Setup}(1^k);\\
&\qquad (\mathit{Nym}_O, \mathit{sk}_U, \mathit{Nym}_U, \mathit{aux}(\mathit{Nym}_U), \mathit{Nym}_D, L, \mathit{state}) \leftarrow \mathcal{A}_1(\mathit{params}_{DC});\\
&\quad \mathcal{A}_2 \leftrightarrow \mathsf{Obtain}(\mathit{params}_{DC}, \mathit{Nym}_O, \mathit{sk}_U, \mathit{Nym}_U, \mathit{aux}(\mathit{Nym}_U), \mathit{Nym}_D, L)) \rightarrow \mathit{cred}_U\\
&\qquad\qquad : (\mathsf{proper}(\mathit{params}_{DC}, \mathit{cred}_U, \mathit{sk}_U, \mathit{Nym}_O, L+1) \vee \mathit{cred}_U = \mathsf{abort})] = 1
\end{aligned}$$

(b). Users with proper level L credentials can delegate proper level $L+1$ credentials. Users with credentials that are not proper will output $\perp$.

For all $\mathit{params}_{DC} \leftarrow \mathsf{Setup}(1^k)$ and for all $\mathit{sk}, \mathit{Nym}_O$,

$$\begin{aligned}
&\Pr[\mathit{sk}_U \leftarrow \mathsf{Keygen}(\mathit{params}_{DC}), (\mathit{Nym}_D, \mathit{aux}(\mathit{Nym}_D)) \leftarrow \mathsf{Nymgen}(\mathit{params}_{DC}, \mathit{sk}_D),\\
&\quad (\mathit{Nym}_U, \mathit{aux}(\mathit{Nym}_U)) \leftarrow \mathsf{Nymgen}(\mathit{params}_{DC}, \mathit{sk}_U),\\
&\quad (\mathsf{Issue}(\mathit{params}_{DC}, \mathit{Nym}_O, \mathit{sk}, \mathit{Nym}, \mathit{aux}(\mathit{Nym}), \mathit{cred}, \mathit{Nym}_U)\\
&\qquad \leftrightarrow \mathsf{Obtain}(\mathit{params}_{DC}, \mathit{Nym}_O, \mathit{sk}_U, \mathit{Nym}_U, \mathit{aux}(\mathit{Nym}_U), \mathit{Nym}_D, L)) \rightarrow \mathit{cred}_U\\
&\quad : (\mathsf{proper}(\mathit{params}_{DC}, \mathit{cred}, \mathit{sk}, \mathit{Nym}_O, L) \wedge \neg\mathsf{proper}(\mathit{params}_{DC}, \mathit{cred}_U, \mathit{sk}_U, \mathit{Nym}_O, L+1))]\\
&\quad = 0
\end{aligned}$$

(c). $\mathsf{Issue}(\mathit{params}_{DC}, \mathit{Nym}_O, \mathit{sk}_D, \mathit{Nym}_D, \mathit{aux}(\mathit{Nym}_D), \mathit{cred}, \mathit{Nym}_U, L)$ aborts without starting any communication if $\mathsf{proper}(\mathit{params}_{DC}, \mathit{cred}, \mathit{sk}_D, \mathit{Nym}_O, L) = 0$, or if there does not exist sk_U, $\mathit{aux}(\mathit{Nym}_U)$ such that $\mathsf{VerifyAux}(\mathit{params}_{DC}, \mathit{Nym}_U, \mathit{sk}_U, \mathit{aux}(\mathit{Nym}_U)) = 1$, or if VerifyAux $(\mathit{params}_{DC}, \mathit{Nym}_D, \mathit{sk}_D, \mathit{aux}(\mathit{Nym}_D)) = 0$.

(d).$\mathsf{CredProve}(\mathit{params}_{DC}, \mathit{Nym}_O, \mathit{cred}, \mathit{sk}, \mathit{Nym}, \mathit{aux}(\mathit{Nym}), L)$ aborts without producing output if $\mathsf{proper}(\mathit{params}_{DC}, \mathit{cred}, \mathit{sk}, \mathit{Nym}_O, L) = 0$, or if $\mathsf{VerifyAux}(\mathit{params}_{DC}, \mathit{Nym}, \mathit{sk}, \mathit{aux}(\mathit{Nym})) = 0$.

(e.) VerifyAux always accepts pseudonyms generated by Nymgen. More formally:

$$\begin{aligned}
&\Pr[\mathit{params}_{DC} \leftarrow \mathsf{Setup}(1^k); \mathit{sk} \leftarrow \mathsf{Keygen}(\mathit{params}_{DC});\\
&\quad (\mathit{Nym}, \mathit{aux}(\mathit{Nym})) \leftarrow \mathsf{Nymgen}(\mathit{params}_{DC}, \mathit{sk}) :\\
&\quad \mathsf{VerifyAux}(\mathit{params}_{DC}, \mathit{Nym}, \mathit{sk}, \mathit{aux}(\mathit{Nym})) = 1] = 1
\end{aligned}$$

Anonymity During any protocol when a user reveals his pseudonym Nym but does not intentionally reveal $(\mathit{sk}, \mathit{aux}(\mathit{Nym}))$, the other user should learn no information about $(\mathit{sk}, \mathit{aux}(\mathit{Nym}))$. By no information, we mean that the user can be replaced by a simulator that does not know $(\mathit{sk}, \mathit{aux}(\mathit{Nym}))$, but still can execute the protocol. The simulator SimSetup, SimProve, SimObtain, SimIssue has the following properties:

(a). The public parameters generated by SimSetup is indistinguishable from those output by Setup.

$$\begin{aligned}
&|\Pr[\mathit{params}_{DC} \leftarrow \mathsf{Setup}(1^k); b \leftarrow \mathcal{A}(\mathit{params}_{DC}) : b = 1]\\
&\quad - \Pr[(\mathit{params}_{DC}, \mathit{sim}) \leftarrow \mathsf{SimSetup}(1^k); b \leftarrow \mathcal{A}(\mathit{params}_{DC}) : b = 1]| < \nu(k)
\end{aligned}$$

(b). A pseudonym Nym reveals no information about the corresponding identity sk. Let $params_{DC}$, $sim \leftarrow \mathsf{SimSetup}(1^k)$, $(sk, pk) \leftarrow \mathsf{Keygen}(params_{DC})$, and $(Nym, aux(Nym)) \leftarrow \mathsf{Nymgen}$ $(params_{DC}, sk)$. Then $(params_{DC}, sim, Nym)$ is information theoretically independent of sk.

(c). The simulator can output a fake credential *credproof* that cannot be distinguished from a real credential, even though the simulator does not have access to sk_U and $cred$ (and sk_U and $cred$ are chosen adversarially). Formally, for all PPTM adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there exists a negligible function ν so that:

$$\begin{aligned}
&|\Pr[(params_{DC}, sim) \leftarrow \mathsf{SimSetup}(1^k);\\
&\quad (Nym_O, cred, sk, Nym, aux(Nym), L, state) \leftarrow \mathcal{A}_1(params_{DC}, sim);\\
&\quad \pi \leftarrow \mathsf{CredProve}(params_{DC}, Nym_O, cred, sk, Nym, aux(Nym), L); b \leftarrow \mathcal{A}_2(state, \pi) : b = 1]\\
&- \Pr[(params_{DC}, sim) \leftarrow \mathsf{SimSetup}(1^k);\\
&\quad (Nym_O, cred, sk, Nym, aux(Nym), L, state) \leftarrow \mathcal{A}_1(params_{DC}, sim);\\
&\quad \mathsf{flag} \leftarrow \mathsf{Check}(params_{DC}, Nym_O, cred, sk, Nym, aux(Nym), L);\\
&\quad \mathsf{SimProve}(params_{DC}, sim, Nym_O, Nym, L, \mathsf{flag});\\
&\quad b \leftarrow \mathcal{A}_2(state, \pi) : b = 1]| < \nu(k).
\end{aligned}$$

$\mathsf{Check}(params_{DC}, Nym_O, cred, sk, Nym, aux(Nym), L)$ outputs accept if $\mathsf{VerifyAux}(params_{DC}$, $Nym, sk, aux(Nym)) = 1$ and $\mathsf{proper}(params_{DC}, cred, sk, Nym_O, L) = 1$

(d). The adversary cannot tell if it is interacting with Obtain or SimObtain. Formally, for all PPTM adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there exists a negligible function ν so that:

$$\begin{aligned}
&\Big|\Pr[params_{DC} \leftarrow \mathsf{Setup}(1^k); (Nym_O, sk, Nym, aux(Nym), L, Nym_A, state) \leftarrow \mathcal{A}_1(params_{DC});\\
&\quad b \leftarrow \mathcal{A}_2(state) \leftrightarrow \mathsf{Obtain}(params_{DC}, Nym_O, sk, Nym, aux(Nym), Nym_A, L) : b = 1]\\
&- \Pr[params_{DC} \leftarrow \mathsf{SimSetup}(1^k);\\
&\quad (Nym_O, sk, Nym, aux(Nym), L, Nym_A, state) \leftarrow \mathcal{A}_1(params_{DC});\\
&\quad \mathsf{flag} \leftarrow \mathsf{Check}(params_{DC}, sk, Nym, aux(Nym));\\
&\quad b \leftarrow \mathcal{A}_2(state) \leftrightarrow \mathsf{SimObtain}(params_{DC}, Nym_O, Nym, Nym_A, L, \mathsf{flag}) : b = 1]\Big| < \nu(k).
\end{aligned}$$

$\mathsf{Check}(params_{DC}, sk, Nym, aux(Nym))$ outputs accept if $\mathsf{VerifyAux}(params_{DC}, Nym, sk,$ $aux(Nym)) = 1$ and reject otherwise.

(e). The adversary cannot tell if it is interacting with Issue or SimIssue. Formally, for all PPTM adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there exists a negligible function ν so that:

$$\begin{aligned}
&|\Pr[(params_{DC}, sim) \leftarrow \mathsf{SimSetup}(1^k);\\
&\quad (Nym_O, sk, Nym, aux(Nym), cred, Nym_A, L, state) \leftarrow \mathcal{A}_1(params_{DC}, sim);\\
&\quad \mathsf{Issue}(params_{DC}, Nym_O, sk, Nym, aux(Nym), cred, Nym_A, L) \leftrightarrow \mathcal{A}_2(state) \rightarrow b :\\
&\quad b = 1]\\
&\ - \Pr[(params_{DC}, sim) \leftarrow \mathsf{SimSetup}(1^k);\\
&\quad (Nym_O, sk, Nym, aux(Nym), cred, Nym_A, L, state) \leftarrow \mathcal{A}_1(params_{DC}, sim);\\
&\quad \mathsf{flag} \leftarrow \mathsf{Check}(params_{DC}, Nym_O, cred, sk, Nym, aux(Nym), L);\\
&\quad \mathsf{SimIssue}(params_{DC}, sim, Nym_O, Nym, Nym_A, L, \mathsf{flag}) \leftrightarrow \mathcal{A}_2(state) \rightarrow b :\\
&\quad b = 1]| < \nu(k)
\end{aligned}$$

$\mathsf{Check}(params_{DC}, Nym_O, cred, sk, Nym, aux(Nym), L)$ outputs accept iff $\mathsf{VerifyAux}(params_{DC}, Nym, sk, aux(Nym)) = 1$ and $\mathsf{proper}(params_{DC}, cred, sk, Nym_O, L) = 1$.

Security of $\mathsf{NymProve}$**.** $\mathsf{NymProve}$ must be a zero knowledge proof of knowledge of sk, $aux(sk)$ such that $\mathsf{VerifyAux}(params_{DC}, Nym, sk, aux(sk)) = 1$. More formally, this means:

(a). Correctness

Let $params_{DC} \leftarrow \mathsf{Setup}(1^k)$, $(sk, pk) \leftarrow \mathsf{Keygen}(params_{DC})$, and $(Nym, aux(Nym)) \leftarrow \mathsf{Nymgen}(params_{DC}, sk)$. If an honest prover runs $\mathsf{NymProve}(params_{DC}, sk, Nym, aux(Nym))$ and an honest verifier runs $\mathsf{NymVerify}(params_{DC}, Nym)$, the verifier always outputs accept.

(b). Zero Knowledge.

The adversary cannot tell if it is interacting with $\mathsf{NymProve}$ or $\mathsf{SimNymProve}$. Formally, for all PPTM adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there exists a negligible function ν so that:

$$\begin{aligned}
&|\Pr[(params_{DC}, sim) \leftarrow \mathsf{SimSetup}(1^k);\\
&\quad (sk, Nym, aux(Nym), state) \leftarrow \mathcal{A}_1(params_{DC}, sim);\\
&\quad \mathsf{NymProve}(params_{DC}, sk, Nym, aux(Nym)) \leftrightarrow \mathcal{A}_2(state) \rightarrow b : b = 1]\\
&\ - \Pr[(params_{DC}, sim) \leftarrow \mathsf{SimSetup}(1^k);\\
&\quad (sk, Nym, aux(Nym), state) \leftarrow \mathcal{A}_1(params_{DC}, sim);\\
&\quad \mathsf{flag} \leftarrow \mathsf{VerifyAux}(params_{DC}, sk, Nym, aux(Nym));\\
&\quad \mathsf{SimNymProve}(params_{DC}, sim, Nym, \mathsf{flag}) \leftrightarrow \mathcal{A}_2(state) \rightarrow b : b = 1]| < \nu(k)
\end{aligned}$$

Note that $\mathsf{SimNymProve}$ is allowed to rewind $\mathcal{A}_2$.

(c). Knowledge Extraction.

There exists an interactive PPTM $\mathsf{NymExtract}$ such that for all PPTM $\mathcal{A}_1, \mathcal{A}_2$ there exists a negligible function ν such that:

$$\begin{aligned}
&|\Pr[(params_{DC}) \leftarrow \mathsf{Setup}(1^k); (state, Nym) \leftarrow \mathcal{A}_1;\\
&\quad \mathcal{A}_2(state) \leftrightarrow \mathsf{VerifyProof}(params_{DC}, Nym) \rightarrow result : result = \mathsf{accept}]\\
&- \Pr[(params_{DC}) \leftarrow \mathsf{Setup}(1^k); (state, Nym) \leftarrow \mathcal{A}_1;\\
&\quad \mathcal{A}_2(state) \leftrightarrow \mathsf{NymExtract}(params_{DC}, Nym) \rightarrow sk :\\
&\quad \exists open \text{ such that } \mathsf{VerifyAux}(Nym, sk, open) = 1]| < \nu(k)
\end{aligned}$$

where NymExtract is allow rewind $\mathcal{A}_2$.

F-Unforgeability. Let F be an efficiently computable bijection. There exists an extractor (ExtSetup, Extract) with four properties:

(a). The parameters generated by ExtSetup are distributed identically as those generated by Setup.

(b). Under these parameters pseudonyms are perfectly binding. I.e. for all $(params_{DC}, td) \leftarrow$ ExtSetup, for all Nym, if there exists $aux(Nym)$, $aux(Nym)'$ such that VerifyAux($params_{DC}$, Nym, sk, $aux(Nym)) = 1$ and VerifyAux($params_{DC}$, Nym, sk', $aux(Nym)') = 1$, then $sk' = sk$

(b). Given an honestly generated level L credential, Extract can always extract the chain of L identities.

$$\begin{aligned}
&\Pr[(params_{DC}, td) \leftarrow \mathsf{ExtSetup}(1^k);\\
&\quad (sk_\ell, pk_\ell) \leftarrow \mathsf{Keygen}(params_{DC}) \text{ for } \ell = 0...L;\\
&\quad (Nym_\ell, aux(Nym_\ell)) \leftarrow \mathsf{Nymgen}(params_{DC}, sk_\ell) \text{ for } \ell = 0...L;\\
&\quad privcred_0 = \perp\\
&\quad ;\mathsf{Issue}(params_{DC}, Nym_0, sk_\ell, Nym_\ell, aux(Nym_\ell), cred_\ell, Nym_{\ell+1}, \ell)\\
&\qquad \leftrightarrow \mathsf{Obtain}(params_{DC}, Nym_0, sk_{\ell+1}, Nym_{\ell+1}, aux(Nym_{\ell+1}), Nym_\ell, \ell) \rightarrow cred_{\ell+1}\\
&\qquad \text{for } \ell = 0...L-1;\\
&\quad credproof \leftarrow \mathsf{CredProve}(params_{DC}, Nym_0, cred_L, sk_L, Nym_L, aux(Nym_L), L);\\
&\quad (f_0, f_1, \ldots, f_L) \leftarrow \mathsf{Extract}(params_{DC}, td, credproof, Nym_L, L) :\\
&\quad \forall \ell = 0...L : f_\ell = F(sk_\ell)] = 1.
\end{aligned}$$

Also, for any valid pseudonym, Extract will produce the appropriate $f(sk)$: For all Nym

$$\begin{aligned}
&\Pr[(params_{DC}, td) \leftarrow \mathsf{ExtSetup}(1^k);\\
&\quad f_0 \leftarrow \mathsf{Extract}(params_{DC}, td, \perp, Nym, Nym, 0) :\\
&\quad (\exists sk, aux(Nym) \text{such that} \mathsf{VerifyAux}(params_{DC}, Nym, sk, aux(Nym)) = 1)\\
&\quad \wedge f_0 \neq F(sk)] = 0.
\end{aligned}$$

(d). Given an adversarially generated credential, Extract will always produce the correct values for

f_0, f_L, or produce $\perp$.

$$\begin{aligned}
&\Pr[(\mathit{params}_{DC}, td) \leftarrow \mathsf{ExtSetup}(1^k);\\
&\quad (\mathit{credproof}, \mathit{Nym}, \mathit{Nym}_O, L), \leftarrow \mathcal{A}(\mathit{params}_{DC}, td);\\
&\quad (f_0, \ldots, f_L) \leftarrow \mathsf{Extract}(\mathit{params}_{DC}, td, \mathit{credproof}, \mathit{Nym}, \mathit{Nym}_O, L) :\\
&\quad (f_0, \ldots, f_L) \neq \perp \wedge\\
&\quad ((\exists sk_{\mathcal{U}}{}' \, \exists aux(\mathit{Nym})' : \mathsf{VerifyAux}(\mathit{params}_{DC}, \mathit{Nym}, sk_{\mathcal{U}}{}', aux(\mathit{Nym})') \wedge F(sk_{\mathcal{U}}{}') \neq f_L)\\
&\quad \vee (\exists sk_O' \, \exists aux(\mathit{Nym}_O)' : \mathsf{VerifyAux}(\mathit{params}_{DC}, \mathit{Nym}_O, sk_O', aux(\mathit{Nym}_O)') \wedge F(sk_O') \neq f_0))]\\
&\quad \leq \nu(k)
\end{aligned}$$

(e). No adversary can output a valid credential from which the extractor extracts an unauthorized chain of identities.

$$\begin{aligned}
&\Pr[(\mathit{params}_{DC}, td) \leftarrow \mathsf{ExtSetup}(1^k);\\
&\quad (\mathit{credproof}, \mathit{Nym}, \mathit{Nym}_O, L), \leftarrow \mathcal{A}^{\mathcal{O}(\mathit{params}_{DC}, \cdot, \cdot)}(\mathit{params}_{DC}, td);\\
&\quad (f_0, \ldots, f_L) \leftarrow \mathsf{Extract}(\mathit{params}_{DC}, td, \mathit{credproof}, \mathit{Nym}, \mathit{Nym}_O, L) :\\
&\quad \mathsf{CredVerify}(\mathit{params}_{DC}, \mathit{Nym}_O, \mathit{credproof}, \mathit{Nym}, L) = \mathsf{accept} \wedge\\
&\quad (\exists i \text{ such that } (f_0, i, f_{i-1}, f_i) \notin \mathsf{ValidCredentialChains} \wedge f_{i-1} \in \mathsf{HonestUsers})] \leq \nu(k)
\end{aligned}$$

where $\mathcal{O}(\mathit{params}_{DC}, \mathit{command}, \mathit{input})$ behaves as follows:

AddUser. The oracle runs $sk \leftarrow \mathsf{Keygen}(\mathit{params}_{DC})$. It stores $(sk, F(sk))$ in the user database and gives the adversary $F(sk)$. Store $F(sk)$ in the list HonestUsers.

FormNym(y). The oracle looks up (sk, y) in its user database and terminates if it does not exist. It calls $(\mathit{Nym}, aux(\mathit{Nym})) \leftarrow \mathsf{Nymgen}(\mathit{params}_{DC}, sk)$. The oracle stores $(sk, \mathit{Nym}, aux(\mathit{Nym}))$ in its pseudonym database and gives the adversary Nym.

Issue$(\mathit{Nym}_D, \mathit{Nym}_U, \mathit{cred}_D, L, \mathit{Nym}_O)$. The oracle looks up $(sk_U, \mathit{Nym}_U, aux(\mathit{Nym}_U))$ and $(sk_D, \mathit{Nym}_D, aux(\mathit{Nym}_D))$ in its pseudonym database and outputs an error if they do not exist. The oracle then generates a credential proof by running $\mathsf{CredProve}(\mathit{params}_{DC}, \mathit{Nym}_O, \mathit{cred}_D, sk_D, \mathit{Nym}_D, aux(\mathit{Nym}_D), L)$ to obtain $\mathit{credproof}_D$ (for $L = 0$, $\mathit{credproof}_D = \perp$). It runs Extract $(\mathit{params}_{DC}, td, \mathit{credproof}_D, \mathit{Nym}_O, \mathit{Nym}_D, L)$ to obtain $f_0, f_1, \ldots f_L$. The oracle then runs

$$\begin{aligned}
&\mathsf{Issue}(\mathit{params}_{DC}, \mathit{Nym}_O, sk_D, \mathit{Nym}_D, aux(\mathit{Nym}_D), \mathit{cred}_D, \mathit{Nym}_U, L)\\
&\qquad \leftrightarrow \mathsf{Obtain}(\mathit{params}_{DC}, \mathit{Nym}_O, sk_U, \mathit{Nym}_U, aux(\mathit{Nym}_U), \mathit{Nym}_D, L) \rightarrow \mathit{cred}_U.
\end{aligned}$$

Finally, the oracle stores $(f_0, L+1, f_L, F(sk_U))$ in ValidCredentialChains and outputs cred_U to the adversary.

IssueToAdv$(\mathit{Nym}_D, \mathit{cred}_D, \mathit{Nym}, L, \mathit{Nym}_O)$. The oracle looks up $(sk_D, pk_D, \mathit{Nym}_D, aux(\mathit{Nym}_D))$ in its pseudonym database, and outputs an error if they do not exist. The oracle generates a credential proof by running $\mathsf{CredProve}(params_{DC}, Nym_O, cred, sk_D, Nym_D, aux(Nym_D), L)$ to obtain $credproof_D$. It runs $\mathsf{Extract}(params_{DC}, td, credproof_D, Nym_D, Nym_O, L)$ to obtain $f_0, \dots f_L$. It then identifies the recipient by running $\mathsf{Extract}(params_{DC}, td, \perp, Nym, Nym, 0)$ to obtain f_{L+1}. Finally the oracle executes the algorithm $\mathsf{Issue}(params_{DC}, Nym_O, sk_D, Nym_D, aux(Nym_D), cred_D, Nym, L)$ interacting with the adversary. If the protocol does not abort, the oracle stores $(f_0, L+1, f_L, f_{L+1})$ in ValidCredentialChains.

$\mathsf{ObtainFromAdv}(Nym_A, Nym_U, Nym_O, L)$ The oracle looks up $(sk_U, Nym_U, aux(Nym_U))$ in its pseudonym database, and outputs an error if they do not exist. Then it runs $\mathsf{Obtain}(params_{DC}, Nym_O, sk_U, Nym_U, aux(Nym_U), Nym_A)$ with the adversary to get $cred$. It outputs $cred$.

$\mathsf{Prove}(Nym, cred, Nym_O, L)$ The oracle looks up $(sk, pk, Nym, aux(Nym))$ in its pseudonym database, and outputs an error if they do not exist. The oracle then runs $\mathsf{CredProve}(params_{DC}, Nym_O, cred, sk, Nym, aux(Nym), L)$ to obtain $credproof_D$, and outputs this result.

7.3 Construction of Delegatable Credentials

We now construct delegatable credentials using the tools defined in Section 4.3.1. The parameters of the system combine the parameters $params_A$ needed for the authentication scheme and $params_{PK}$ needed for a composable and randomizable NIZKPK proof system and its associated commitment scheme Commit. It is important that the message authentication domain $\mathsf{AuthKg}(params_A)$ is a subset of the domain of inputs to the commitment scheme. Each user U has a secret key $sk_U \leftarrow \mathsf{AuthKg}(params_A)$, and forms his pseudonyms using Commit: $Nym_U = \mathsf{Commit}(sk_U, open_U)$. U can create arbitrarily many different pseudonyms by choosing new random values $open_U$. As described in 7.2, a user can act as an authority (originator) for some type of a credential by making his pseudonym Nym_O publicly available.

We denote a user's private credential as $cred$. To show or delegate the credential, the user randomizes $cred$ to form $credproof$. In our construction, $cred$ is in fact an NIZKPK of a statement about U's specific *secret* pseudonym $S_U = \mathsf{Commit}(sk_U, 0)$ (this specific pseudonym does not in fact hide sk_U since it is formed as a deterministic function of sk_U) while $credproof$ is a statement about a proper pseudonym, $Nym_U = \mathsf{Commit}(sk_U, open)$ for a randomly chosen $open$. So U randomizes $cred$ to obtain $credproof$ using the RandProof algorithm described in Section 4.3.1.

Suppose a user with secret key sk_U has a level L credential from some authority A, and let $(sk_O, sk_1, \dots, sk_{L-1}, sk_U)$ be the keys such that the owner of sk_i delegated the credential to sk_{i+1} (we let $sk_0 = sk_O$ and $sk_L = sk_U$). A *certification chain* is a list of authenticators $auth_1, \dots, auth_L$, such that sk_i generated authenticator $auth_{i+1}$ on sk_{i+1}.

To make sure that pieces of different certification chains cannot be mixed and matched, we add labels r_i to each authenticator. The labels have to be unique for each authority and delegation level. Let H be a collision resistant hash function with an appropriate range. For a credential chain rooted at Nym_O, we set $r_i = H(Nym_O, i)$. Each $auth_i$ is an output of $\mathsf{Auth}(params_A, sk_{i-1}, (sk_i, r_{i-1}, r_i))$. The user U's level L private credential $cred$ is, therefore a proof as follows:

$$cred \in \mathsf{NIZKPK}[sk_O \text{ in } Nym_O; sk_U \text{ in } S_U]\{(F(sk_O), F(sk_1), \ldots, F(sk_{L-1}), F(sk_U), auth_1, \ldots, auth_L) : \\ \mathsf{VerifyAuth}(params, sk_O, (sk_1, r_0, r_1), auth_1) \wedge \mathsf{VerifyAuth}(params, sk_1, (sk_2, r_1, r_2), auth_2) \\ \wedge \ldots \wedge \mathsf{VerifyAuth}(params, sk_{L-1}, (sk_U, r_{L-1}, r_L), auth_L)\} \ .$$

Here, F is a bijection such that the authentication scheme is F-unforgeable. As a result, if on input *cred* the extractor outputs a certification chain involving honest users, this certification chain will correspond to these users' delegation activities, or if not, then the security of the authentication scheme does not hold. We use concatenation, projection and randomization properties of our NIZKPK to allow a delegating issuer I with a level L credential to delegate it to a user U. We have already explained this in Section 4.3.1.

We now give the full construction. Let PKSetup, PKProve, PKVerify be a proof system and let AuthSetup, AuthKg, Auth, VerifyAuth be an authentication scheme, and let $H : \{0,1\}^* \to Z_p$ be a hash function.

$\mathsf{Setup}(1^k)$. Use $\mathsf{AuthSetup}(1^k)$ to generate $params_A$ and $\mathsf{PKSetup}(1^k)$ to generate $params_{PK}$; choose the hash function H (as explained above) and outputs public parameters $params_{DC} = (params_A, params_{PK}, H)$.

$\mathsf{Keygen}(params_{DC})$. Run $\mathsf{AuthKg}(params_A)$ and outputs the secret key sk.

$\mathsf{Nymgen}(params_{DC}, sk)$. Choose a random $open \in Y$ (where Y is the domain of openings of the commitment scheme Commit associated with $params_{PK}$, as explained in Section 4.3.1). Compute $Nym = \mathsf{Commit}(params_{PK}, sk, open)$ and outputs pseudonym Nym and auxiliary information $open$.

$\mathsf{NymProve}(params_{DC}, sk, Nym, open) \leftrightarrow \mathsf{NymVerify}(params_{DC}, Nym)$. The prover and verifier carry out an *interactive* zero-knowledge proof of knowledge of a witness $(sk, open)$, such that $Nym = \mathsf{Commit}(params_{PK}, sk, open)$[1].

$\mathsf{VerifyAux}(params_{DC}, Nym, sk, open)$ accepts iff $Nym = \mathsf{Commit}(sk, open)$.

$\mathsf{CredProve}(params_{DC}, Nym_O, cred, sk_U, Nym_U, open_U, L)$. Recall that *cred* should be an NIZKPK of a certification chain (above, we already gave the NIZKPK formula for it with the correct Condition and extraction function f). If $\mathsf{PKVerify}(params_{PK}, (Nym_O, \mathsf{Commit}(sk_U, 0)), cred)$ rejects, or if $Nym_U \neq \mathsf{Commit}(sk_U, open_U)$, abort. Otherwise, set $credproof \leftarrow \mathsf{RandProof}((Nym_O, Nym_U), (0, open_U), cred)$. Note that, by the randomization properties of the proof system,

$$credproof \in \mathsf{NIZKPK}[sk_O \text{ in } Nym_O; sk_U \text{ in } Nym_U]\{(F(sk_O), F(sk_1), \ldots, F(sk_{L-1}), F(sk_U), auth_1, \ldots, auth_L) : \\ \mathsf{VerifyAuth}(params, sk_O, (sk_1, r_0, r_1), auth_1) \wedge \mathsf{VerifyAuth}(params, sk_1, (sk_2, r_1, r_2), auth_2) \\ \wedge \ldots \wedge \mathsf{VerifyAuth}(params, sk_{L-1}, (sk_U, r_{L-1}, r_L), auth_L)\} \ .$$

$\mathsf{CredVerify}(params_{DC}, Nym_O, credproof, Nym_U, L)$ runs PKVerify to verify the above.

$\mathsf{Issue}(params_{DC}, Nym_O, sk_I, Nym_I, open_I, cred, Nym_U, L)$
$\leftrightarrow \mathsf{Obtain}(params_{DC}, Nym_O, sk_U, Nym_U, open_U, Nym_I, L)$. If $L = 0$ and $Nym_O \neq Nym_I$, then this

[1] Such protocols exist for all NP; in our instantiation, Commit is Pedersen-like and therefore this proof can be done efficiently.

protocol is aborted. The issuer verifies his $cred$ using $\mathsf{CredVerify}$ and if it does not verify or if $Nym_I \neq \mathsf{Commit}(sk_I, open_I)$ or Nym_U is not a valid pseudonym, the issuer aborts.

Else, the issuer and the user both compute $r_{L-1} = H(Nym_O, L-1)$ and $r_L = H(Nym_O, L)$. The issuer and the user run a parameterized (by $params_{DC}$) two-party protocol with the following specifications: the public input is $(Nym_I, Nym_U, r_{l-1}, r_L)$; the issuer's private input is $(sk_I, open_I)$ and the user's private input is $(sk_U, open_U)$. The output of the protocol is as follows: if the issuer did not supply $(sk_I, open_I)$ such that $Nym_I = \mathsf{Commit}(sk_I, open_I)$, or if the user did not supply $(sk_U, open_U)$ such that $Nym_U = \mathsf{Commit}(sk_U, open_U)$, the protocol aborts; otherwise, the issuer receives no output while the user receives as output the value π computed as:

$$\pi \leftarrow \mathsf{NIZKPK}[sk_I \text{ in } Nym_I; sk_U \text{ in } \mathsf{Commit}(sk_U, 0)]\{(F(sk_I), F(sk_U), auth) : \\ \mathsf{VerifyAuth}(params, sk_I, (sk_U, r_L, r_{L+1}), auth)\} \ .$$

In Section 4.3.3 we give an efficient instantiation of this 2PC protocol for the specific authentication and NIZKPK schemes we use.

If $L = 0$, then $Nym_I = Nym_O$, and so we are done, the user outputs $cred_U = \pi$. If $L > 0$, then the issuer is not the original authority for this credential, so he is a delegator. The issuer obtains $credproof_I \leftarrow \mathsf{CredProve}(params_{DC}, Nym_O, cred, sk_I, Nym_I, open_I, L)$ and sends it to the user. Let $S_U = \mathsf{Commit}(sk_U, 0)$. Intuitively, $credproof_I$ is proof that the owner of Nym_I has a level L credential under public key Nym_O, while π is proof that the owner of Nym_I delegated to the owner of S_U. The user concatenates $credproof_I$ and π to obtain:

$$credproof_I \circ \pi \in \mathsf{NIZKPK}[sk_O \text{ in } Nym_O; sk_I \text{ in } Nym_I; sk_U \text{ in } S_U]\{(F(sk_O), F(sk_1), \ldots, F(sk_{L-1}), \\ F(sk_I), F(sk_U), auth_1, \ldots, auth_L, auth_{L+1}) : \mathsf{VerifyAuth}(params, sk_O, (sk_1, r_0, r_1), auth_1) \wedge \\ \mathsf{VerifyAuth}(params, sk_1, (sk_2, r_1, r_2), auth_2) \wedge \ldots \wedge \mathsf{VerifyAuth}(params, sk_{L-1}, (sk_I, r_{L-1}, r_L), \\ auth_L \wedge \mathsf{VerifyAuth}(params, sk_I, (sk_U, r_L, r_{L+1}), auth_{L+1})\} \ .$$

To get $cred_U$, U now needs to project $credproof_I \circ \pi$ so it becomes a proof about (Nym_O, S_U) and not about Nym_I.

Theorem 25. *If* $\mathsf{AuthSetup}, \mathsf{AuthKg}, \mathsf{Auth}, \mathsf{VerifyAuth}$ *is an F-unforgeable certification-secure authentication scheme, and if H is a collision resistant hash function, and if* $\mathsf{PKSetup}, \mathsf{PKProve}, \mathsf{PKVerify}$ *is a randomizable, perfectly extractable, composable zero knowledge non-interactive proof of knowledge system with simulation setup* $\mathsf{SimSetup}$ *and extraction setup* $\mathsf{ComExtractSetup}$*, and if the two party protocol is trapdoor secure for the simulation trapdoors generated by* $\mathsf{SimSetup}$ *and trapdoor secure for the extraction trapdoors generated by* $\mathsf{ComExtractSetup}$*, then the above construction constitutes a secure anonymous delegatable credential scheme. (See Section 7.5 for proof.)*

7.3.1 Efficiency

Here we will consider the efficiency of the $\mathsf{CredProve}$ and $\mathsf{CredVerify}$ algorithms. A credential proof for a level L credential consists of $L + 1$ commitments to values $u^{sk_i} \in G_1$, $L + 1$ commitments to values

$h^{sk_i} \in G_2$, $L+1$ Groth-Sahai proofs for pairing products with $Q = 2$ which prove that these values are computed correctly with respect to each other, and L zero knowledge proofs each of which proves knowledge of an authenticator under a committed secret key on 3 committed messages, computed as described in Section 3.5. (Note that commitments to values u^{r_i}, h^{r_i} will use randomness 0, and can be computed by anyone, so they do not need to be included in the proof.) Running CredProve, just involves randomizing all of these commitments and proofs, which as was mentioned in Section 3.7.3 takes the same amount of time as forming new commitments and new proofs for these statements. Similarly, CredVerify simply verifies all the proofs involved. Applying the efficiency formulas given in Sections 3.4 and 4.3.3, we get the following efficiency results:

Theorem 26. *When the underlying Groth-Sahai proofs are instantiated using the SXDH instantiation given in Section 3.3 the above credential system will have the following efficiency: Generating a proof for a level L credential will require* $404L + 20$ *exponentiations in* G_1 *and* $360L + 20$ *exponentiations in* G_2*. The resulting proof will consist of* $136L + 6$ *elements of* G_1 *and* $114L + 6$ *elements of* G_2*. Verifying the proof will involve computing* $516L + 24$ *bilinear group pairings.*

When the underlying Groth-Sahai proofs are instantiated using the symmetric setting version of the DLIN instantiation given in Section 3.3, the above credential system will have the following efficiency: Generating a proof for a level L credential will require $1503L + 81$ *exponentiations in G. The resulting proof will consist of* $303L + 15$ *elements in G. Verifying the proof will involve computing* $945L + 45$ *bilinear group pairings.*

7.4 Adding Attributes

In certain contexts, we want the ability to express *why* a credential has been delegated. For example, a Prof. Alice with a level 1 credential from the university can delegate a level 2 credential to Bob either because Bob is her teaching assistant and needs access to student files or because Bob is a pizza delivery man and needs access to the building after 5pm.

A simple solution is for the university to use multiple public keys and give Prof. Alice credentials under each of them. The credential from pk_O is for delegating to teaching assistants; the credential from pk'_O is for delegating to pizza delivery. However, the situation is more complicated. The pizza delivery is authorized for a particular evening. The university does not want to give each professor a new credential for every day of the year! This become even more complicated when Alice *and* the university must predict how Bob will delegate his credential.

The solution is to include an attribute at each level of the delegatable credential. Thus, we would say that the authority delegated to the owner of $comm_1$ with attr_1, the owner of $comm_1$ delegated to the owner of $comm_2$ with attr_2, etc. This lets Bob present a credential that states that the university authorized the owner of $comm_1$ with $\mathsf{attr}_1 =$"professor", the owner of $comm_1$, in turn, authorized the owner of my pseudonym $comm_2$ with $\mathsf{attr}_2 =$"deliver a pizza on May 4th, at 5pm."

Now we briefly explain how to extend our credential scheme to allow attributes. Our authentication scheme in Section 4.2.4 can handle multiple messages. We use the first three messages for the delegatable credentials as before. The rest are for attributes. Section 4.3.3 already explains how to create proofs of knowledge of an authenticator for multiple messages. Delegation is straightforward because the user is allowed to know the value of the attribute and the opening.

7.5 Security Proof for Delegatable Credential Construction

7.5.1 Correctness

Claim. *The delegatable credential construction given in Section 7.3 satisfies the Correctness property*

(a). Note that Obtain aborts if the $credproof$ that it receives from the issuer does not pass CredVerify. Then Obtain only completes without aborting if the two party computation completes successfully. In this case, by the security of the two party computation, we are certain that π_L is indistinguishable from an honestly generated proof of knowledge of an honestly generated authenticator. Then by the correctness of the NIZKPK system and correctness of the authentication scheme, we know that π_L will pass the NIZKPK verification. Thus, the new $cred$ that results from combining $credproof$ with π_L will pass the NIZKPK verification. Furthermore, by correctness of randomization, this means that any randomization of this proof will pass the NIZKPK verification and thus the CredVerify, so $cred$ is proper.

(b). If the $cred$ is not a proper credential, then by the properties of the randomization, CredVerify will not accept the resulting $credproof$, so Issue will abort. Issue also aborts if Nym_U is not valid, or if sk_I, $open_I$ do not satisfy VerifyAux. Now, if Issue is interacting with Obtain with the appropriate inputs, then it will produce $credproof$ that will be accepted by the honest user. Then the two party computation will be successful by the 2PC correctness property. That means Obtain will not abort, so by property (a). it will produce a proper credential.

(c). Note that if $cred$ passes the CredVerify, then by correctness of randomization, it must be a proper credential. Thus, if $cred$ is not a proper credential, or if $Nym, sk, aux(Nym)$ does not pass VerifyAux, then CredProve aborts. If both of these are correct, then $credproof$ is a randomization of a proper credential. That means that the proof must verify, so CredVerify will accept.

(d). Follows from the definition of VerifyAux.

7.5.2 Anonymity

Claim. *The delegatable credential construction given in Section 7.3 satisfies the* Anonymity *property under the assumption that our building blocks are secure.*

Proof. Define the simulator algorithms as follows:

SimSetup(1^k) Uses AuthSetup(1^k) to generate $params_A$ for an F-unforgeable certification secure authentication scheme and then uses GSSimSetup to choose corresponding $params_P$ for a randomizable commitment scheme with a partially extractable randomizable composable NIZKPK proof system, and to choose the appropriate trapdoor sim_{NIZK}. Finally, the setup chooses a collision resistant hash function H whose range is the message space of the authentication scheme and outputs public parameters $params_{DC} = (params_A, params_P, H)$, $sim = sim_{\text{NIZK}}$.

$\mathsf{SimProve}(params_{DC}, sim, Nym_O, Nym, L, flag)$**.** If $flag = \mathsf{reject}$, abort and return $\perp$.

Otherwise let $Nym_0 = Nym_O$ and $Nym_L = Nym$, generate commitments $Nym_1, \ldots Nym_{L-1}$ to random values, and computes $r_0, \ldots r_L$ where $r_i = H(Nym_O, i)$.

Then we use the NIZKPK simulator and the simulation trapdoor sim to simulate proofs $\pi_1 \ldots \pi_L$, where π_i is a simulated proof of the form

$$\begin{aligned}\pi_i \leftarrow \mathsf{SimNIZKPK}[sk_{i-1} \text{ in } Nym_{i-1}; sk_i \text{ in } Nym_i]\{ \\ (F(sk_{i-1}), F(sk_i), auth) : \\ \mathsf{VerifyAuth}(params, sk_{i-1}, (sk_i, r_{i-1}, r_i), auth) = \mathsf{accept}\}\end{aligned}$$

Finally, we output $Nym_0, \ldots Nym_L$, and $\pi_1 \circ \cdots \circ \pi_L$.

$\mathsf{SimObtain}(params_{DC}, Nym_O, Nym_U, Nym_A, L, flag)$ If $flag = \mathsf{reject}$, abort and return $\perp$.

It then proceeds as follows:

1. Receives $credproof$ from the adversary.
2. Runs $\mathsf{CredVerify}(params_{DC}, Nym_O, credproof, Nym_I, L)$ to checks, that $credproof$ is correct. If the checks fail, it aborts.

   Otherwise, it compute $r_{L-1} = H(Nym_O, L-1)$, $r_L = H(Nym_O, L))$, the commitment vector $comm_{\vec{m_h}} = (Nym_U)$, and the vector of public inputs $\vec{m_o} = (r_{L-1}, r_L)$.
3. Now we must simulate the two-party computation protocol. We will do this by using the 2PC simulator which interacts with a corrupt signer. (Note that this simulator expects no input from the trusted functionality.)

$\mathsf{SimIssue}(params_{DC}, sim, Nym_O, Nym_D, Nym_A, L, flag)$ If $flag = \mathsf{reject}$, abort and return $\perp$.

Otherwise let $Nym_0 = Nym_O$, $Nym_L = Nym_D$ and $Nym_{L+1} = Nym_A$, generate commitments $Nym_1, \ldots Nym_{L-1}$ to random values, and computes $r_0, \ldots r_{L+1}$ where $r_i = H(Nym_O, i)$.

Then we use the NIZKPK simulator and the simulation trapdoor sim to simulate proofs $\pi_1 \ldots \pi_L$, where π_i is a simulated proof of the form

$$\begin{aligned}\pi_i \leftarrow \mathsf{SimNIZKPK}[sk_{i-1} \text{ in } Nym_{i-1}; sk_i \text{ in } Nym_i]\{ \\ (F(sk_{i-1}), F(sk_i), auth) : \\ \mathsf{VerifyAuth}(params, sk_{i-1}, (sk_i, r_{i-1}, r_i), auth) = \mathsf{accept}\}\end{aligned}$$

1. Send $Nym_0, \ldots Nym_L, \pi_1, \ldots \pi_{L-1}$ to the adversary
2. Receive $comm_{\vec{m}}$ and check that the first commitment is Nym_A.
3. Now we must simulate the two-party computation protocol. We will do this by using the 2PC simulator which interacts with a corrupt recipient. Note that this simulator expects to be provided with a proof of knowledge of the appropriate authenticator. Thus, we will give it the proof π_L computed above.

Now we will prove that these algorithms satisfy the required properties:

(a). Holds by the composable zero knowledge properties of the underlying NIZK proof system.

(b). Holds by the strong computational hiding property of the underlying commitment scheme.

(c). Note that the difference between SimProve and CredProve is that SimProve generates $Nym_1, \ldots,$ Nym_{L-1} as random commitments and uses SimNIZKPK to generated simulated proofs $\pi_1, \ldots, \pi_L$. These commitments are identical to the honest commitments by the strong computational hiding property, and the simulated proofs are indistinguishable from the honestly randomized proofs by the randomizability property of the randomizable proof system.

(d). Note that the difference between SimObtain and Obtain is that SimObtain uses the simulator to run the two party computation. This should be indistinguishable from the honest Obtain protocol by the security of the 2PC.

(e). SimIssue differs in three ways from Issue. First, the initial $credproof$ that is sent to the user is formed using SimNIZKPK instead of by randomizing a real $cred$. Second, SimIssue uses the simulator to run the two party computation. Third, the resulting π_L is the output of SimNIZKPK and not the result of a valid NIZKPK of an Auth computed by the 2PC.

We can prove that SimIssue and Issue are indistinguishable by considering several hybrid algorithms.

Hybrid 1 will be given the same input as Issue. It will verify that $cred$ is proper, that it has been given a correct $sk, Nym, aux(Nym)$ and that Nym_A is a valid pseudonym. It will compute $credproof$ honestly. Then it will use the 2PC simulator to run the two party protocol. This simulator will extract sk_A, $aux()A$ from the user, and expect to receive a corresponding authenticator proof. We will use sk to form and authenticator on sk_A, r_L, r_{L+1}, and then we will use the honest NIZKPK to generate π_L from it. Finally, we will pass π_L to the 2PC simulator which will complete the protocol.

Note that Hybrid 1 is indistinguishable from the game involving the real Issue by the security of the 2PC.

Hybrid 2 will be given the same input as Issue. It will verify that $cred$ is proper, that it has been given a correct $sk, Nym, aux(Nym)$ and that Nym_A is a valid pseudonym. It will compute $credproof$ honestly. Then it will use the 2PC simulator to run the two party protocol. This simulator will extract sk_A, $aux()A$ from the user, and expect to receive a corresponding authenticator proof. This time, we will use SimNIZKPK to simulate the proof of knowledge of an authenticator on for Nym_A. We will pass π_L to the 2PC simulator which will complete the protocol.

Note that Hybrid 2 is indistinguishable from Hybrid 1 by the zero knowledge properties of SimNIZKPK.

Finally, note that the difference between Hybrid 2 and game with SimIssue is that Hybrid 2 generates $credproof$ by randomizing $cred$, while SimIssue the SimNIZKPK. This means the two games are indistinguishable by the randomization properties of the NIZKPK system.

□

7.5.3 F-Unforgeability

Claim. *The delegatable credential construction given in Section 7.3 satisfies the F-*Unforgeability *property under the assumption that our building blocks are secure.*

Proof. Let ExtSetup be the same as Setup except that when generating $params_{PK}$ it uses the extraction setup of the partially extractable randomizable composable NIZKPK proof system. (a) As these parameters are required to be indistinguishable, so are the $params_{DC}$. (b) The commitment schemes used with the NIZKPK proof system is perfectly binding, so are our pseudonyms.

Let F correspond to the F of the F-unforgeable authentication scheme and let Extract be an algorithm that verifies the credential and aborts if CredVerify aborts, otherwise it uses the extractability features of the NIZKPK proof system to extract all $F(sk_i)$ values. (c) An honestly generated level L credential is a proof of knowledge of a certification chain from sk_0 to delegate to sk_L. It allows to extract $(f_0, \dots, f_L) = (F(sk_0), \dots, F(sk_L))$. If the length of certification chain is 0 Extract extract $f_0 = F(sk_O)$ from any valid commitment Nym_O using the extraction property of the commitment. (d) An adversarially generated level L credential proof $credproof$ from authority Nym_O for pseudonym Nym either verifies, in which case we can extract $f_0, \dots f_L$ from the proof such that Nym_O is a pseudonym for $F^{-1}(f_0)$ and Nym is a pseudonym for $F^{-1}(f_L)$, or it doesn't. In the latter case Extract outputs $\perp$.

(e) Let Q be the maximum number of users in a credential system. We consider two games. In Game 1 the adversary plays the real unforgeability game. In Game 2 we pick a random $q \in \{1, \dots, Q\}$. Game 2 is the same as Game 1, except that in the oracle queries with command IssueToAdv and ObtainFromAdv are answered differently for the qth user.

Let sk^* be the secret key generated for the qth AddUser query.

IssueToAdv$(Nym_I, cred_I, Nym, L, Nym_O)$**.** If Nym is not a valid pseudonym for sk, $aux(Nym)$, the oracle terminates. The oracle looks up $(sk_I, pk_I, Nym_I, aux(Nym_I))$ in its pseudonym database, and outputs an error if they do not exist. If $sk_I \neq sk^*$ it proceeds as in Game 1. Otherwise the oracle follows the Issue protocol until Step 4, but uses the simulator for the two-party protocol to simulate interaction with the adversarial user.

ObtainFromAdv(Nym_A, Nym_U, Nym_O, L) Similarly, the oracle looks up $(sk_U, Nym_U, aux(Nym_U))$ in its pseudonym database, and outputs an error if they do not exist. If $sk_U \neq sk^*$ the oracle runs Obtain $(params_{DC}, Nym_O, sk_U, Nym_U, aux(Nym_U), Nym_A)$ with the adversary to get $cred$ (the same as Game 1). Otherwise it follows the Obtain protocol until Step 5, and now uses the simulator for the two-party protocol to simulate the interaction with the adversarial issuer. It outputs $cred$.

By a simple hybrid argument either Game 1 and Game 2 are computationally indistinguishable or we break the security of the two-party computation.

Next we give a reduction to show that an adversary $\mathcal{A}$ that can win in Game 2 can be used to break the security of our authentication scheme. The reduction gets $params_A$, $f^* = F(sk^*)$ and access to $\mathcal{O}_{\mathsf{Auth}}(params_A, sk^*, .)$, $\mathcal{O}_{\mathsf{Certify}}(params_A, ., (sk^*, ., \dots))$ for a challenge secret key sk^* from the authentication scheme's unforgeability game. It creates matching proof system parameters $params_{PK}$ and a trapdoor td, and combines them into $params_{DC}$. It hands $params_{DC}$ and td to $\mathcal{A}$.

The reduction answers $\mathcal{A}$s oracle queries as follows:

AddUser**.** The oracle keeps a counter indicating the number of times it was queried, otherwise it behaves as the original oracle, except for the qth query. To answers the qth query the oracle stores $(\square, F(sk^*))$ in the user database and adds $F(sk^*)$ in the list HonestUsers. It returns $F(sk^*)$ to $\mathcal{A}$. Note that we use the special token '$\Box$' to indicate the unknown challenge key.

FormNym(f). The oracle looks up (sk, f) in its user database and terminates if it does not exist (we explicitly allow $sk = \Box$). If $f \neq f^*$ the oracle behaves as the original oracle. Otherwise it picks a random $aux(Nym)$ and computes $Nym = \mathsf{Commit}'(f, aux(Nym))$. The oracle stores $(\Box, Nym, aux(Nym))$ in its pseudonym database and gives the adversary Nym.

Issue$(Nym_I, Nym_U, cred_I, L, Nym_O)$. The oracle looks up $(sk_U, Nym_U, aux(Nym_U))$ and $(sk_I, Nym_I, aux(Nym_I))$ in its pseudonym database and outputs an error if they do not exist or if $sk_U = sk_I$. If $sk_U \neq \Box$ and $sk_I \neq \Box$ the oracle behaves as the original oracle. Otherwise we distinguish two cases (note that $sk_U = \Box$ and $sk_I = \Box$ is not possible as honest users do not issue to themselves):

(a) $sk_U = \Box$: If $L > 0$, it runs Extract$(params_{DC}, td, credproof_I, Nym_I)$ to obtain $f_0, f_1, \ldots f_L$. If $f_L \neq F(sk_I)$, it aborts. Otherwise the oracle computes $cred_U$ in the same way as Issue $\leftrightarrow$ Obtain except that it uses a query to $\mathcal{O}_{\mathsf{Certify}}(params_A, sk_I, (sk^*, H(Nym_O, L-1), H(Nym_O, L)))$ to obtain the witness for proof π which it then can compute itself. The oracle stores $(f_0, L, f(sk_I), f^*)$ in ValidCredentialChains and outputs $cred_U$ to the adversary.

(b) $sk_I = \Box$: If $L > 0$,it runs Extract$(params_{DC}, td, credproof_I, Nym_I)$ to obtain $f_0, f_1, \ldots f_L$. If $f_L \neq f^*$, it aborts. Otherwise the oracle computes $cred_U$ in the same way as Issue $\leftrightarrow$ Obtain except that it uses a query to $\mathcal{O}_{\mathsf{Auth}}(params_A, sk^*, (sk_U, H(Nym_O, L-1), H(Nym_O, L)))$ to obtain the witness for proof π which it then can compute itself. The oracle stores $(f_0, L, f^*, F(sk_U))$ in ValidCredentialChains and outputs $cred_U$ to the adversary.

IssueToAdv$(Nym_I, cred_I, Nym, L, Nym_O)$. If Nym is not a valid pseudonym for sk, $aux(Nym)$, the oracle terminates. The oracle looks up $(sk_I, pk_I, Nym_I, aux(Nym_I))$ in its pseudonym database, and outputs $\perp$ if they do not exist. If $sk_I \neq \Box$ the oracle behaves as the original oracle. Otherwise the oracle follows the Issue protocol until Step 4, but uses the simulator for the two-party protocol to simulate interaction with the adversarial user. We can simulate the ideal functionality of the two-party protocol with a $\mathcal{O}_{\mathsf{Auth}}(params_A, sk^*, (sk_U, H(Nym_O, L-1), H(Nym_O, L)))$ query. Note that the simulator of the two-party protocol provides us with sk_U. If the oracle's output is not $\perp$, the oracle stores $(f_0, L, f^*, f(sk_U))$ in ValidCredentialChains.

ObtainFromAdv(Nym_A, Nym_U, Nym_O) Similarly, the oracle looks up $(sk_U, Nym_U, aux(Nym_U))$ in its pseudonym database, and outputs an error if they do not exist. If $sk_U \neq \Box$ the oracle behaves normally. Otherwise it follows the Obtain protocol until Step 4, and now uses the simulator for the two-party protocol to simulate the interaction with the adversarial issuer. We can simulate the ideal functionality of the two-party protocol with a $\mathcal{O}_{\mathsf{Certify}}(params_A, sk_I, (sk^*, H(Nym_O, L-1), H(Nym_O, L)))$ query. After a successful protocol execution the oracle outputs $cred$.

Prove$(Nym, cred, Nym_O)$ The prove protocol does not require a user's secret key, and is answered as the original oracle.

This simulation of Game 2 is perfect. We now need to show a forgery in the credential system can be turned into a forgery for the authentication scheme.

A successful adversary outputs $(credproof, Nym, Nym_O, L)$ such that $\mathsf{CredVerify}(params_{DC}, pk_O, credproof, Nym, L) = \mathsf{accept}$ and $(f_0, \ldots, f_L) \leftarrow \mathsf{Extract}(params_{DC}, td, credproof, Nym, Nym_O)$.

If we guessed correctly for which honest user $\mathcal{A}$ would create the forgery, $f_{i-1} = f^*$ and we can extract an $auth_i$ such that $\mathsf{VerifyAuth}(params_A, sk^*, (sk_i, H(Nym_O, i-1), H(Nym_O, i)), auth_i) = 1$. As $(f_0, i, f_i) \notin \mathsf{ValidCredentialChains}$, this message was never signed before and constitutes a valid authentication forgery. $\square$

Chapter 8

Application to Efficient NIZK for Other Languages

It has been more than twenty years since the discovery of zero-knowledge proofs. In that time, they have attracted interest from the theoretical computer science community (leading to the study of interactive proof systems and PCPs), theoretical cryptography community, and, more recently, cryptographic practice.

The proof protocols that have been implemented so far [15, 19, 17], even though zero-knowledge in spirit, are not, strictly speaking, zero-knowledge proofs as we usually define them. Typically, they are honest-verifier interactive zero-knowledge proofs (sometimes, actually, arguments of knowledge) with the interactive step removed using the Fiat-Shamir paradigm [47, 52]. Interaction is an expensive resource, and so using a heuristic such as the Fiat-Shamir transform in order to remove interaction is more attractive than using an interactive proof.

SINGLE-THEOREM NIZK In contrast to the Fiat-Shamir-based protocols adopted in practice, that do not in fact provide more than just a heuristic security guarantee [52], there are also well-known provable techniques for achieving zero-knowledge in non-interactive proofs. Blum et al. [8, 42, 7] introduced the notion of a *non-interactive zero-knowledge* (NIZK) proof system. In such a proof system, some parameters of the system are set up securely ahead of time. Specifically, a common random string σ is available to all participants. The prover in such a proof system is given an $x \in L$ for some language L and a witness w attesting that $x \in L$. (For example, L can be the language of all pairs (n, e) e is relatively prime to $\phi(n)$. The witness w can be the factorization of n.) The prover computes a proof π, and the proof system is zero-knowledge in the following sense: the simulator can pick its own σ' for which it can find a proof π' for the statement $x \in L$. The values (σ', π') output by the simulator are indistinguishable from (σ, π) that are generated by first picking a random σ and then having the honest prover produce π for $x \in L$ using witness w. Blum et al. also gave several languages L with reasonably efficient NIZK proof systems.

Let us explain the Blum et al. NIZK proof system for the language L in the example above: $L = \{(n, e) \mid \gcd(\phi(n), e) = 1\}$. First, recall that if e is not relatively prime to $\phi(n)$, then the probability that for a random $x \in \mathbb{Z}_n^*$ there exists y such that $y^e = x \bmod n$ is upper-bounded by $1/2$. On the other hand, if e *is* relatively prime to $\phi(n)$, then for all $x \in \mathbb{Z}_n^*$ there exists such a y. So the proof system would go as follows: parse the common random string σ as a sequence $z_1, \ldots, z_\ell$ of elements of $\mathbb{Z}_n^*$, and for each z_i, compute y_i such that $y_i^e = z_i \bmod n$. The proof π consists of the values $y_1, \ldots, y_\ell$. The verifier simply needs to check that each y_i is the e^{th} root of z_i. For any specific instance (n, e), the probability (over the choice of the common random string σ) that a cheating prover can come up with a proof that passes the verification is $2^{-\ell}$. By the union bound, letting $\ell = k(|n| + |e|)$ guarantees that the probability, over the choice of σ, that a cheating prover can find an instance (n, e) and a proof π passing the verification, is negligible in k.

Note that the proof system described above, although expensive, is not prohibitively so. Proof systems of this type have been shown to yield themselves to further optimizations [70]. So why aren't such proofs attractive in practice?

MULTI-THEOREM NIZK The problem with NIZK as initially defined and explained above was that one proof π completely used up the common random string σ, and so to produce more proofs, fresh common randomness was required. Blum et al. [7] showed a single-prover multi-theorem NIZK proof system for 3SAT, and since 3SAT is NP-complete, the result followed for any language in NP, assuming quadratic residuocity. Feige, Lapidot and Shamir [46] constructed a multi-prover, multi-theorem NIZK proof system for all NP based on trapdoor permutations. Recently, Groth, Ostrovsky and Sahai [56] gave a multi-theorem NIZK proof system for circuit satisfiability with very compact common parameters and achieving perfect zero-knowledge (with computational soundness), based on the assumption that the Boneh, Goh, Nissim [13] cryptosystem is secure.

In each of the multi-theorem NIZK results mentioned above, to prove that $x \in L$ for a language L in NP, the prover would proceed as follows: first, reduce x to an instance of the right NP-complete problem, also keeping track of the witness w. Then invoke the multi-theorem NIZK proof system constructed for this NP-complete problem. In other words, even if the language L itself had an efficient *single-theorem* NIZK, existing multi-theorem NIZK constructions have no way of exploiting it. The Feige et al. result, which is the most attractive because it is based on general assumptions, is especially bad in this regard: their construction explicitly includes a step that transforms every instance x into a new instance x' via a Cook-Levin reduction. These reductions are what makes NIZK prohibitively expensive to be considered for use in practice.

In this paper, we give a construction for achieving multi-theorem NIZK for any language L based on single-theorem NIZK for L, without having to reduce instances of L to instances of any NP-complete languages. This construction is based on a new building block: a *simulatable verifiable random function* (sVRF) as described in chapter 5.

USING AN SVRF TO TRANSFORM SINGLE-THEOREM NIZK TO MULTI-THEOREM NIZK. A simulatable VRF with domain of size $\ell(k)$ and binary range allows a prover to come up with a fresh verifiably random string R of appropriate length $\ell(k)$ every time he wants to prove a new theorem. He simply comes up with a new pk for a VRF, and evaluates F_{pk} on input i to obtain the ith bit of R, R_i. The VRF allows him to prove that R was chosen correctly. He can then XOR R with a truly random public string σ_1 to obtain a string σ to be used in a single-theorem NIZK. The resulting construction is zero-knowledge because of the simulatability properties of both the sVRF and the single-theorem NIZK. It is sound because σ_1 is a truly random string, and so it inherits the soundness from the single-theorem NIZK (note that it incurs a penalty in the soundness error). Note that because our sVRF construction is in the public parameters model, the resulting multi-theorem proof system is also in the public parameters model (rather than the common random string model).

Since we give an *efficient* instantiation of sVRFs, our results essentially mean that studying efficient single-theorem NIZK proof systems for languages of interest is a good idea, because our construction gives an *efficient* transformation from such proof systems to multi-theorem ones.

USING AN SVRF INSTEAD OF THE RANDOM ORACLE. An sVRF shares some characteristics with a programmable random oracle: assuming that the parameters of the system were picked by the simulator, the simulator can *program* it to take certain values on certain inputs. One cannot necessarily use it instead of the hash function in constructions where the adversary gets the code for the hash function. But it turns out that it can sometimes replace the random oracle in constructions where the adversary is allowed oracle access to the hash function and requires some means to be sure that the output is correct. For example, using an sVRF instead of H in the RSA-FDH construction [6, 37] would make the same proof of security hold without the random oracle. Of course, it is not a useful insight: an sVRF is already a signature, so it is silly to use it as a building block in constructing another signature. The reason we think the above observation is worth-while is that it is an example of when using an sVRF instead of an RO gives provable instead of heuristic guarantees.

8.1 Efficient Transformation based on sVRFs

Here, we omit the definition of single-theorem and multi-theorem NIZK, but refer the reader to Blum et al. [7] and Feige, Lapidot, Shamir [46]. Instead, we informally sketch this definition:

Algorithms `NIZKProve` **and** `NIZKVerify` The algorithm `NIZKProve` takes as input the common random string σ of length $\ell(k)$, and values (x, w), $|x| \leq q(k)$, such that $x \in L$, and w is a witness to this. `NIZKProve` outputs a *proof* Π. `NIZKVerify` is the algorithm that takes (σ, x, Π) as input, and outputs *ACCEPT* or *REJECT*.

Perfect completeness For all $x \in L$, for all witnesses w for x, for all values of the public random string σ, and for all outputs π of $\texttt{NIZKProve}(\sigma, x, w)$, $\texttt{NIZKVerify}(\sigma, x, \pi) =$ *ACCEPT*.

Soundness $s(k)$ For all adversarial prover algorithms $\mathcal{A}$, for a randomly chosen σ, the probability that $\mathcal{A}$ can produce (x, π) such that $x \notin L$ but $\texttt{NIZKVerify}(\sigma, x, \pi) =$ *ACCEPT*, is $s(k)$.

Single-theorem ZK There exists an algorithm `SimProveOne` that, on input 1^k and $x \in L$, $|x| \leq q(k)$, outputs simulated CRS σ^S together with a simulated proof Π^S, such that (σ^S, Π^S) are distributed indistinguishably from (σ, Π) produced by generating a random CRS σ, and obtaining Π by running `NIZKProve`.

Multi-theorem ZK There exist algorithms `SimCRS` and `NIZKSimProve`, as follows: $\texttt{SimCRS}(1^k)$ outputs (σ, s). For all x, $\texttt{NIZKSimProve}(\sigma, s, x)$ outputs a *simulated proof* Π^S. Even for a sequence of adversarially and adaptively picked $(x_1, \dots, x_m)$ (m is polynomial in k), if for all $1 \leq i \leq m$, $x_i \in L$, then the simulated proofs $\Pi_1^S, \ldots, \Pi_m^S$ are distributed indistinguishably from proofs $\Pi_1, \ldots, \Pi_m$ that are computed by running $\texttt{NIZKProve}(\sigma, x_i, w_i)$, where w_i is some witness that $x_i \in L$.

Suppose that, for a language L, we are given a single-theorem NIZK proof system $(\texttt{ProveOne}, \texttt{VerOne})$ in the CRS model, with perfect completeness and unconditional soundness error $s(k)$. Let $\ell(k)$ denote the function such that an $\ell(k)$-bit random string serves as the CRS for this proof system. Let $q(k)$ denote the polynomial upper bound on the size of the input x. Suppose also that we are given a simulatable VRF $(G, \texttt{Eval}, \texttt{Prove}, \texttt{Verify})$ in the parameter model $params$, whose domain is $[1, \ell(k)]$, with range $\{0, 1\}$. Consider the following construction for multi-theorem NIZK in the common reference string model for instances of size k:

Generate common parameters The algorithm $\texttt{NIZKParams}$: Obtain $\sigma_1 \leftarrow \{0,1\}^{\ell(k)}$. Let $\texttt{p} \leftarrow params$ (1^k). The values $(\sigma_1, \texttt{p})$ are the parameters of the system.

Prove The algorithm $\texttt{NIZKProve}$: On input instance $x \in L$ with witness w, and common parameters $(\sigma_1, \texttt{p})$ do: Obtain $(pk, sk) \leftarrow G(1^k, \texttt{p})$. Let R be the $\ell(k)$-bit string computed as follows: for $1 \leq i \leq \ell(k)$, $R_i = \texttt{Eval}(\texttt{p}, sk, i)$, where R_i denotes the ith bit of R. For $1 \leq i \leq \ell(k)$, let $\pi_i \leftarrow \texttt{Prove}(\texttt{p}, sk, i)$. Let $\sigma = \sigma_1 \oplus R$. Obtain $\Pi' \leftarrow \texttt{ProveOne}(\sigma, x, w)$. Output the proof $\Pi = (pk, R, \pi_1, \ldots, p_{\ell(k)}, \Pi')$.

Verify The algorithm $\texttt{NIZKVerify}$: On input x and Π, and common parameters $(\sigma_1, \texttt{p})$, do: (1) for $1 \leq i \leq \ell(i)$, check that $\texttt{Verify}(\texttt{p}, pk, i, R_i, \pi_i)$ accepts; (2) let $\sigma = \sigma_1 \oplus R$; check that $\texttt{VerOne}(\sigma, x, \Pi')$ accepts; if all these checks passed, accept, otherwise, reject.

8.2 Efficiency

Note that we need an sVRF whose output is a random bit-string (so that we can use it to form the CRS for our single theorem NIZK proof system). The construction given in Section 5.2.2 has as its output range the bilinear group G_1, and the only known way to convert it to a bit string output is to apply the techniques in Section 5.1.2. This means that there is no efficiency advantage to using this construction over the one given in 5.2.1 (where we also need to apply these techniques, but where the underlying weak sVRF is more efficient). Thus, the theorem below presents the efficiency of the transformation using this second sVRF construction:

Theorem 27. *Suppose* $(\texttt{ProveOne}, \texttt{VerOne})$ *is a single theorem proof system which requires a common random string of length* $\ell(k)$ *for each proof. Then the above transformation instantiated with the sVRF construction from Sections 5.1.2 and 5.2.1 results in a many theorem proof system, where the additional overhead for each proof is: an increase in prover computation by* $4\ell(k)$ *exponentiations in the composite order bilinear group, an increase in proof length by* $3\ell(k)$ *elements of the composite order bilinear group, and an increase in verification time by* $5\ell(k)$ *composite order bilinear pairings.*

8.3 Proof of Security

Theorem 28. *If for a language L, $(\texttt{ProveOne}, \texttt{VerOne})$ is a single-theorem NIZK proof system in the $\ell(k)$-bit CRS model for instances of length up to $q(k)$ with perfect completeness and unconditional soundness error $s(k)$, and $(G, \texttt{Eval}, \texttt{Prove}, \texttt{Verify})$ in the parameter model $params(1^k)$, is a strong simulatable VRF with domain $[1, \ell(k)]$ and range $\{0, 1\}$, then the above construction is a multi-theorem NIZK proof system in the public parameters model that comprises the $\ell(k)$-bit CRS and $params(1^k)$, with perfect completeness and unconditional soundness error $s(k)2^{u(k)}$, where u denotes the bit length of a pk output by $G(\texttt{p})$ on input $\texttt{p} \leftarrow params(1^k)$.*

Proof. (Sketch) The perfect completeness property follows from the perfect completeness property of the single-theorem NIZK.

Let us show the multi-theorem zero-knowledge property. Recall that, by the definition of (strong) sVRF, we have a simulator consisting of *SimParams*, *SimG* and *SimProve* such that, if (pk, sk) were generated by *SimG*, then for a randomly sampled y from the range of the sVRF, and for any x in the domain, *SimProve* can generate a fake proof that $y = \texttt{Eval}(sk, x)$. (See Section 5.1.)

Also recall that by the definition of NIZK, there exists a simulator
SimProveOne such that no adversary $\mathcal{A}$ can distinguish between the following two distributions for any $x \in L$ and any witness w for x: (1) choose $\sigma \leftarrow \{0, 1\}^{\ell(k)}$, and let $\Pi \leftarrow \texttt{ProveOne}(\sigma, x, w)$; give (σ, Π) to $\mathcal{A}$; (2) $(\sigma, \Pi) \leftarrow \texttt{SimProveOne}(1^k, x)$; give (σ, Π) to $\mathcal{A}$.

Consider the following simulator S for our multi-theorem NIZK construction. The simulator will consist of SimCRS that generates the simulated parameters, and of NIZKSimProve that generates the simulated proof. SimCRS works as follows: generate $(\texttt{p}, t) \leftarrow$ *SimParams*, and $\sigma_1 \leftarrow \{0, 1\}^{\ell(k)}$; publish $(\sigma_1, \texttt{p})$ as the parameters of the system. NIZKSimProve works like this: generate $(\sigma, \Pi') \leftarrow \texttt{SimProveOne}(1^k, x)$. Then let $R = \sigma \oplus \sigma_1$. Let $(pk, sk) \leftarrow$ *SimG*$(\texttt{p}, t)$. For $1 \leq i \leq \ell(k)$, let $\pi_i =$ *SimProve*$(\texttt{p}, sk, x, R_i, t)$. Output the proof $\Pi = (pk, R, \pi_1, \ldots, p_{\ell(k)}, \Pi')$. In the full version, we show that the view that the adversary obtains in the simulation is indistinguishable from the view obtained when interacting with the prover.

We now show soundness. We are given that, for $\sigma \leftarrow \{0, 1\}^{\ell(k)}$, the probability that there exists $x \notin L$ and a proof Π' such that $\texttt{Verify}(\sigma, x, \Pi') = 1$, is $s(k)$.

Consider $\texttt{p} \leftarrow params$, and $(pk, sk) \leftarrow G(1^k)$. Let R be as defined in NIZKProve: $R_i = \texttt{Eval}(sk, i)$. Note that by the verifiability property of the sVRF, there is a *unique* R for which there exists a proof of correctness $(\pi_1, \ldots, \pi_{\ell(k)})$. The probability, over the choice of σ_1, that there exists $x \notin L$ and a proof Π' such that $\texttt{Verify}(R \oplus \sigma_1, x, \Pi') = 1$ (if such an x exists, we say that pk is *bad* for σ_1), is still $s(k)$, since we first fixed p and pk, and then randomly chose σ_1. By the union bound, since there are $2^{u(k)}$ possible pk's, for every p, the probability that there exists a bad pk for a particular σ_1, is $s(k)2^{u(k)}$. □

Remark. Note that if an NIZK proof system is in the hidden-random-string (HRS) model (such as those due to Feige, Lapidot and Shamir [46] and Kilian and Petrank [63]), then we can take advantage of it as follows: the hidden random string can be obtained the way that σ is currently obtained by the prover in the construction above; only in the construction above, the prover reveals the entire string σ and the proof that each bit of σ is computed correctly; while in the HRS model, the prover only reveals the subset of bits of the hidden random string that he needs to reveal. This observation was inspired by Dwork and Naor's construction of zaps from VRFs and verifiable PRGs [44] based on NIZK using HRS model. We give more details on consequences in the HRS model in the full version.

Chapter 9

Application to E-cash

Electronic cash (e-cash) was introduced by David Chaum [33] as an electronic analogue of physical cash. Cash transactions have the advantage over checks or credit cards in that they are private. At the same time, it is very difficult to counterfeit actual coins and banknotes. E-cash means to provide the same anonymity and unforgeability guarantees. E-cash and variants are an attractive cryptographic application, because they allow the accountability without the privacy infringement that other accountability mechanisms may provide. Of recent interest is using e-cash in peer-to-peer systems to give peers an incentive to participate [4]. For example, in a file sharing system, currently peers make their content available to others out of pure altruism, with no incentive. As a result, the potential of peer-to-peer systems is underutilized, since many potential participants are turned away because they cannot find what they are looking for or because they can download only at a very slow rate. So an accountability mechanism that would ensure that peers upload and download at roughly the same rate is very desirable, but not at the expense of privacy. This is where e-cash comes in.

The participants in an e-cash system are users who withdraw and spend e-cash; a bank that creates e-cash and accepts it for deposit, and merchants who offer goods and services in exchange for e-cash, and then deposit the e-cash to the bank. The algorithms and protocols involved are a key generation algorithm for each participant, a withdrawal protocol that is meant to be a protocol between a user and the bank, a spend protocol between a user and a merchant, and a deposit protocol for users and merchants to deposit their e-cash back into their accounts (and a few additional algorithms we discuss below). The main security requirements are (1) anonymity: even if the bank and the merchant and all the remaining users collude with each other, they still cannot distinguish Alice's purchases from Bob's; (2) unforgeability: even if all the users and all the merchants collude against the bank, they still cannot deposit more money than they withdrew. (There also additional requirements that we will come back to later.)

Unfortunately, it is easy to see that, as described above, e-cash is useless. The problem is that here money is represented by data, and it is possible to copy data. So nothing stops Alice from withdrawing a dollar and spending it twice. In both transactions, the merchants will accept. Unforgeability will guarantee that the bank will only honor at most one of them for deposit and will reject the other one. Anonymity will guarantee that there is no recourse against Alice. So one of the merchants will be cheated.

There are two known remedies against this double-spending behavior. In both, each coin has a serial number. The withdrawal protocol can be thought of as a “blind signature” [34] in which the user obtains the bank’s signature on this serial number (thus certifying that this is a serial number of a valid coin) without the bank learning anything about this serial number. The first remedy is on-line e-cash [33], where the bank is asked to vet a serial number before the spend protocol can terminate successfully. If this is a serial number of an already spent coin, the bank will warn the merchant that it would not accept it for deposit, and so the spend transaction will be cancelled. If everything goes well, as a result of the spend protocol, the merchant will obtain the serial number S and a proof π, where π is a non-interactive zero-knowledge proof of knowledge of the bank’s signature on S. It is, of course, undesirable to involve the bank in every transaction.

The second remedy is off-line e-cash, introduced by Chaum, Fiat and Naor [36]. The idea is that, first of all, the withdrawal protocol is not just a blind signature on the serial number S, but in fact a blind signature on the user’s identifier x and another random value T. When spending an e-coin, the user must reveal S and $B = x + RT \bmod q$ where R is a fresh random value agreed upon between the user and the merchant, and q is a prime greater than x, R and T. Now double-spending a coin with serial number S leads to identification through simply solving a system of linear equations in case of double-spending. So the additional requirement of an offline e-cash system is (informally) that no coin can be double-spent without revealing the identity of the perpetrator.

A further development in the literature on e-cash was compact e-cash [21]. In compact e-cash, the user withdraws N coins in a withdrawal protocol whose complexity is $O(\log N)$ rather than $O(N)$. The main idea is as follows: in the withdrawal protocol, a user obtains the Bank’s signature on (x, s, t), where s and t are random seeds of a pseudorandom function $F_{(\cdot)}(\cdot)$, x is the user’s identifier. In the spend protocol, the serial number of the ith coin was computed as $S = F_s(i)$, and the double spending equation is computed as $T = x + RF_t(i)$. The coin itself consists of (S, T, π), where π is a non-interactive zero-knowledge proof of knowledge of the following values: x, s, t, i, σ where σ is the Bank’s signature on (x, s, t), $1 \leq i \leq N$, $S = F_s(i)$ and $T = x + RF_t(i) \bmod q$. If g is a generator of a group G of order q, and G is the range of the PRF $F_{(\cdot)}(\cdot)$, then the double-spending equation can instead be computed as $T = g^x F_t(i)^R$. It is easy to see that two double-spending equations for the same t, i but different R’s allow us to compute g^x. It was shown that this approach yields a compact e-cash scheme [21]. Later, this was extended to so-called e-tokens [20] that allow up to k anonymous transactions per time period (for example, this would correspond to subscriptions to interactive game sites or anonymous sensor reports).

Thus, we see that compact e-cash and variants such as e-tokens can be obtained from a signature scheme, a pseudorandom function, and a non-interactive zero-knowledge proof of knowledge (ZKPOK) proof system for the appropriate language.

Our contribution to the study of electronic cash is to give a construction of a signature scheme and a PRF such that the non-interactive ZKPOK proof system is provably secure and can be realized efficiently enough to be usable in practice. Our construction is in the common parameters model and relies on several number-theoretic assumptions (discussed in Section 2.3). All prior work on e-cash [36, 15, 21] was exclusively in the random-oracle model, with non-interactive proofs obtained from interactive proofs via the Fiat-Shamir heuristic [47] which is known not to yield provably secure constructions [52]. The reason for this is that, until the recent proof system of Groth and Sahai [57] there were no efficient NIZKPOK proof systems for languages most heavily used in cryptographic constructions (such as languages of true statements about discrete logarithm representations).

One of the main building blocks is a pseudorandom function and an unconditionally binding commitment scheme $Commit$ with an efficient proof of knowledge system for the following language: $L_F = \{S, C_y, C_s \mid \exists s, y, r_s, r_y \text{ such that } S = F_s(y), C_y = Commit(y, r_y), C_s = Commit(s, r_s)\}$ (the prover will prove knowledge of the witness). Such a pseudorandom function with a proof system is a special case of a simulatable verifiable random function (sVRF), introduced by Chase and Lysyanskaya [32]. Chase and Lysyanskaya also gave an efficient construction of a multi-theorem non-interactive zero-knowledge proof systems for any language L from a single-theorem one for the same language (while other single-theorem to multi-theorem transformations required the Cook-Levin reduction to an NP-complete language first). Our construction of an sVRF is more efficient than the one due to Chase and Lysyanskaya by a factor of the security parameter; it is also designed in a way that is more modular and therefore easier to understand. Therefore, this result is of independent interest.

Note, however, that although a ZKPOK for L_F allows proofs that a given serial number S is computed as a PRF (this is a step towards proving that it is a valid serial number), it does not in itself give a proof that a double-spending equation T is correct. And so for the PRF we construct, we give a NIZKPOK for a more general language $L_T = \{T, C_x, C_y, C_t, R, \langle g \rangle \mid \exists x, y, t, r_x, r_y, r_t \text{ such that } T = g^x F_t(y)^R, C_x = Commit(x, r_x), C_y = Commit(y, r_y), C_t = Commit(t, r_t)\}$, where the group $\langle g \rangle$ is the range of the PRF.

Our other building block is a signature scheme and an unconditionally binding commitment scheme (the same one as for the sVRF construction) that allows for an efficient NIZKPOK of a signature on a set of committed values, as well as for an efficient protocol for getting a committed value signed. This is done via a modification of the P-signature construction of Belenkiy et al. [5] that allowed the same for one committed value.

Finally, we show how to obtain compact e-cash using these building blocks, and show that the resulting construction is secure.

9.1 Definitions

Compact e-cash as defined by [21] involves a bank $\mathcal{I}$ as well as many users $\mathcal{U}$ and merchants $\mathcal{M}$. Merchants are treated as a special type of user. The parties $\mathcal{I}$, $\mathcal{U}$, and $\mathcal{M}$ interact using the algorithms CashSetup, BankKG, UserKG, SpendCoin, VerifyCoin, Deposit, Identify, and the interactive protocol Withdraw. We write $\mathsf{Protocol}(\mathcal{A}(I_\mathcal{A}), \mathcal{I}(I_\mathcal{I}))$ to denote an interactive protocol Protocol between $\mathcal{A}$ and $\mathcal{I}$ with secret inputs $I_\mathcal{A}, I_\mathcal{I}$ and secret outputs $O_\mathcal{A}, O_\mathcal{I}$ respectively.

$\mathsf{CashSetup}(1^k)$ creates the public parameters $params$.

$\mathsf{BankKG}(params, n)$ outputs the key pair $(pk_\mathcal{I}, sk_\mathcal{I})$ that is used by the bank to issue wallets of n coins. For simplicity, we assume that the secret key contains the corresponding public key.

$\mathsf{UserKG}(params)$ generates a user (or merchant) key pair $(pk_\mathcal{U}, sk_\mathcal{U})$. The keys are used for authentication and non-repudiation.[1]

$\mathsf{Withdraw}(\mathcal{U}(params, pk_{\mathcal{I}}, sk_{\mathcal{U}}), \mathcal{I}(params, pk_{\mathcal{U}}, sk_{\mathcal{I}}))$ is an interactive protocol in which a user withdraws a *wallet* W of n coins from the bank where n is specified in $pk_{\mathcal{I}}$. The wallet includes the public key of the bank. The bank learns some trace information T_W that it can later use to identify double-spenders. We say that Withdraw succeeds if both the user and the bank output acceptat the end of execution.

$\mathsf{SpendCoin}(params,\ W, pk_{\mathcal{M}}, info)$ allows a user with a non-empty wallet W and some unique transaction information $info$ to create a coin. The output of the algorithm is $(\ W', coin)$, the updated wallet and an e-coin that can be given to a merchant. The e-coin consists of a serial number S, transaction information $info$, and a proof π.

$\mathsf{VerifyCoin}(params, pk_{\mathcal{M}}, pk_{\mathcal{I}}, coin)$ allows a merchant to verify $coin = (S, info, \pi)$ received from a user. The output of the algorithm is either accept or reject. The merchant accepts the coin on accept but only if he has never accepted a coin with the same $info$ before.

$\mathsf{Deposit}(params, pk_{\mathcal{I}}, pk_{\mathcal{M}}, coin, state_{\mathcal{I}})$ allows the bank to verify a coin received from merchant. The bank needs to maintain a database $state_{\mathcal{I}}$ of all previously accepted coins. The output of the algorithm is an updated database $state'_{\mathcal{I}}$ and the flag $result$, which can have three values:

(i) accept indicates that the coin is correct and fresh. The bank deposits the value of the e-coin into the merchant's account and adds $(pk_{\mathcal{M}}, coin)$ to $state_{\mathcal{I}}$.

(ii) merchant indicates that either $\mathsf{VerifyCoin}(params, sk_{\mathcal{M}}, pk_{\mathcal{I}}, coin) = 0$, or that $state_{\mathcal{I}}$ already contains an entry $(pk_{\mathcal{M}}, coin)$. The bank refuses to accept the e-coin because the merchant failed to properly verify it.

(iii) user indicates that there exists a second coin with the same serial number S registered in $state_{\mathcal{I}}$. (Using the two coins the bank will identify the double-spending user.) The bank pays the merchant (who accepted the e-coin in good faith) and punishes the double-spending user.

$\mathsf{Identify}(params, coin, coin')$ allows the bank to identify a double-spender. The algorithm outputs T_W, which the bank compares to the trace information it stores after each withdrawal transaction.

Notes. Camenisch et al. [21] only define spending as the interactive protocol $\mathsf{Spend}(\mathcal{U}(params,\ W, pk_{\mathcal{M}}), \mathcal{M}(params, sk_{\mathcal{M}}, pk_{\mathcal{I}})$. We can derive their protocol from our non-interactive algorithms. First the merchant sends the user $info$. Then the user runs $\mathsf{SpendCoin}(params,\ W, pk_{\mathcal{M}}, info)$ and sends the resulting coin back to the merchant. The merchant accepts the e-coin only if $\mathsf{VerifyCoin}(params, pk_{\mathcal{M}}, pk_{\mathcal{I}}, coin)$ outputs accept and the $info$ used to construct the e-coin is correct. Non-interactive spend protocols are important when two-way communication is not available or impractical, e.g. when sending an e-coin by email.

[1]Our scheme does not involve the user's secret in the creation of the wallet. This means that we are unable to implement the exculpability scheme of [21]. Instead we provide a different solution.

Definition 34 (Secure Compact E-Cash with Non-Interactive Spend)**.** *A compact e-cash scheme consists of the non-interactive algorithms* CashSetup, BankKG, UserKG, SpendCoin, VerifyCoin, Deposit, Identify, *and the interactive protocol* Withdraw. *We say that such a scheme is secure if it has the* Correctness, Anonymity, Balance, *and* Identification *properties.*

Correctness. When the bank and user are honest, and the user has sufficient funds, Withdraw will always succeed. An honest merchant will always accept an e-coin from an honest user. A honest bank will always accept an e-coin from an honest merchant.

Anonymity. A malicious coalition of banks and merchants should not be able to distinguish if the Spend protocol is executed by honest users or by a simulator that does not know any of the users' secret data. Formally, there must exist a simulator $\mathsf{Sim} = (\mathsf{SimCashSetup}, \mathsf{SimSpend})$. such that for all non-uniform polynomial time $\mathcal{A}$ there exists a negligible function ν such that:

$$\begin{aligned}\Big| Pr[&params \leftarrow \mathsf{CashSetup}(1^k); (pk_{\mathcal{I}}, state) \leftarrow \mathcal{A}_1(params) : \\ &\mathcal{A}_2^{\mathcal{O}_{\mathsf{Spend}}(params, pk_{\mathcal{I}},\cdot,\cdot)}{}^{\mathcal{O}_{\mathsf{GetKey}}(params,\cdot), \mathcal{O}_{\mathsf{Withdraw}}(params, pk_{\mathcal{I}},\cdot,\cdot)}(state) = 1] \\ -Pr[&(params, sim) \leftarrow \mathsf{SimCashSetup}(1^k); (pk_{\mathcal{I}}, state) \leftarrow \mathcal{A}_1(params) : \\ &\mathcal{A}_2^{\mathcal{O}_{\mathsf{SimSpend}}(params, pk_{\mathcal{I}},\cdot,\cdot,\cdot)}{}^{\mathcal{O}_{\mathsf{GetKey}}(params,\cdot), \mathcal{O}_{\mathsf{Withdraw}}(params, pk_{\mathcal{I}},\cdot,\cdot)}(state) = 1]\Big| < \nu(k)\end{aligned}$$

The oracles $\mathcal{O}_{\mathsf{GetKey}}$, $\mathcal{O}_{\mathsf{Withdraw}}$, $\mathcal{O}_{\mathsf{SpendCoin}}$, and $\mathcal{O}_{\mathsf{SimSpend}}$ are defined as follows:

$\mathcal{O}_{\mathsf{GetKey}}(params, i)$**.** The oracle returns $pk_{\mathcal{U}_i}$, the public-key of user $\mathcal{U}_i$. If $pk_{\mathcal{U}_i}$ doesn't exist, the oracle generates $(pk_{\mathcal{U}_i}, sk_{\mathcal{U}_i})$ using $\mathsf{UserKG}(params)$.

$\mathcal{O}_{\mathsf{Withdraw}}(params, pk_{\mathcal{I}}, i, j)$**.** The oracle runs the Withdraw protocol with the adversary: $\mathsf{Withdraw}(\mathcal{U}(params, pk_{\mathcal{I}}, sk_{\mathcal{U}_i}), \mathcal{A}_2(state))$. The adversary plays the role of the bank and the oracle takes the role of user $\mathcal{U}_i$. If $sk_{\mathcal{U}_i}$ doesn't exist, the oracle generates it using $\mathsf{UserKG}(params)$. The value j serves to identify the wallet to the oracle for later use; the adversary must use a fresh value j each time it calls $\mathcal{O}_{\mathsf{Withdraw}}$. The oracle will not reveal the wallet W_j it obtained to the adversary.

$\mathcal{O}_{\mathsf{Spend}}(params, pk_{\mathcal{I}}, pk_{\mathcal{M}}, i, j, info)$**.** The oracle runs $\mathsf{SpendCoin}(params, W_j, pk_{\mathcal{M}}.info)$ and returns the resulting $coin$ to the adversary. The oracle outputs error if the adversary has not previously called $\mathcal{O}_{\mathsf{Withdraw}}(params, pk_{\mathcal{I}}, i, j)$, or if the adversary has already called $\mathcal{O}_{\mathsf{Spend}}(params, pk_{\mathcal{I}}, pk_{\mathcal{M}}, i, j, \cdot)$ n times.

$\mathcal{O}_{\mathsf{SimSpend}}(params, pk_{\mathcal{I}}, pk_{\mathcal{M}}, i, j, info)$. The oracle runs $\mathsf{SimSpend}(params, sim, pk_{\mathcal{I}}, pk_{\mathcal{M}}, info)$ and return the resulting $coin$. The oracle outputs error if the adversary has not previously called $\mathcal{O}_{\mathsf{Withdraw}}(params, pk_{\mathcal{I}}, i, j)$, or if the adversary has already called $\mathcal{O}_{\mathsf{Spend}}(params, pk_{\mathcal{I}}, pk_{\mathcal{M}}, i, j, \cdot)$ n times.

Balance. No coalition of users should be able to deposit more e-coins than they collectively withdrew. Formally, for all non-uniform polynomial time $\mathcal{A}$ and every $n < poly(k)$ there exists a negligible function

ν such that

$$\begin{aligned}&Pr[params \leftarrow \mathsf{CashSetup}(1^k); (pk_{\mathcal{I}}, sk_{\mathcal{I}}) \leftarrow \mathsf{BankKG}(params, n);\\ &\quad (withdrawals, deposits) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Withdraw}}(params,\cdot,sk_{\mathcal{I}}),\mathcal{O}_{\mathsf{Deposit}}(params,pk_{\mathcal{I}},\cdot,\cdot,state_{\mathcal{I}})} :\\ &\quad withdrawals < deposits] < \nu(k)\end{aligned}$$

Where $withdrawals$ is the total number of successful calls to $\mathcal{O}_{\mathsf{Withdraw}}$ multiplied by n (i.e. the number of coins withdrawn) and $deposits$ is the total number of successful calls to $\mathcal{O}_{\mathsf{Deposit}}$. Success means that the oracles output accept. We define the oracles as follows:

$\mathcal{O}_{\mathsf{Withdraw}}(params, pk_{\mathcal{U}}, sk_{\mathcal{I}})$. Runs the Withdraw protocol, where the oracle acts as the bank and the adversary plays the role of the user: $\mathsf{Withdraw}(\mathcal{A}(state), \mathcal{I}(params, pk_{\mathcal{U}}, sk_{\mathcal{I}}))$. $\mathcal{O}_{\mathsf{Withdraw}}$ outputs accept if the protocol outputs accept.

$\mathcal{O}_{\mathsf{Deposit}}(params, pk_{\mathcal{I}}, pk_{\mathcal{M}}, coin, state_{\mathcal{I}})$ Runs the Deposit protocol, where the oracle acts as the bank and the adversary plays the role of a merchant. If this is the first call to $\mathcal{O}_{\mathsf{Deposit}}$, then the oracle sets $state_{\mathcal{I}}$ to $\perp$. Then the oracle updates $state_{\mathcal{I}}$ in the usual way. The oracle outputs accept only if the Deposit protocol outputs accept.

Identification. The bank will be able to identify any user who generates to two valid e-coins (i.e. e-coins that pass the VerifyCointest) with the same serial number. Formally, for all non-uniform polynomial time $\mathcal{A}$ and every $n < poly(k)$ there exists a negligible function ν such that

$$\begin{aligned}&Pr[params \leftarrow \mathsf{CashSetup}(1^k); (pk_{\mathcal{I}}, sk_{\mathcal{I}}) \leftarrow \mathsf{BankKG}(params, n);\\ &\quad (coin_1, coin_2) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Withdraw}}(params,\cdot,sk_{\mathcal{I}})}(params, pk_{\mathcal{I}}) :\\ &\quad coin_1 = (S, info_1, pk_{\mathcal{M}1}) \wedge coin_2 = (S, info_2, pk_{\mathcal{M}2})\\ &\quad \wedge pk_{\mathcal{M}1}||info_1 \neq pk_{\mathcal{M}2}||info_2\\ &\quad \wedge \mathsf{VerifyCoin}(params, pk_{\mathcal{M}1}, pk_B, coin_1) = \mathsf{accept}\\ &\quad \wedge \mathsf{VerifyCoin}(params, pk_{\mathcal{M}2}, pk_B, coin_2) = \mathsf{accept}\\ &\quad \wedge \mathsf{Identify}(params, coin_1, coin_2) \notin T] < \nu(k)\end{aligned}$$

Oracle $\mathcal{O}_{\mathsf{Withdraw}}(params, pk_{\mathcal{U}}, sk_{\mathcal{I}})$ runs protocol Withdraw, with the oracle acting as the bank and the adversary playing the part of the user: $\mathsf{Withdraw}(\mathcal{A}(state), \mathcal{I}(params, pk_{\mathcal{U}}, sk_{\mathcal{I}}))$. The bank adds the resulting T_W to its database T.

9.2 NIZK Proofs for a More Complex Language

In our application, we use NIZKs about PRFs in two different places. The first is to prove that a given serial number has been computed correctly as $\mathsf{PRF}_s(x)$ according to a committed seed s and committed input x. That can be done using the NIZK protocol described in the previous section. However, we also need to be able to prove that the doublespending value T has been computed correctly. Given commitments to user secret key sk_U, PRF seed t, and input x, and given some double spending challenge ch, we need to prove that the given value $T = (g^{sk_U})^{ch}\mathsf{PRF}_t(x)$. Thus, we need to be able to prove that a certain function of a pseudorandom function has been computed correctly. We can generalize our above proof system to handle this as well.

Let $\mathcal{L}_T(params)$ be the set of tuples $C_s, C_x, C_{sk}, tag, ch$ such that tag is the correct output with respect to ch and the x, s, sk contained in C_s, C_x, C_{sk}. I.e.

$$\begin{aligned}\mathcal{L}_T = \{&C_s, C_x, C_{sk}, tag, ch \mid \exists x, s, sk, aux(x), aux(s), aux(sk) \text{ such that}\\ &C_s = \mathsf{Commit}(s, aux(s)) \wedge C_x = \mathsf{Commit}(x, aux(x)) \wedge C_{sk} = \mathsf{Commit}(sk, aux(sk))\\ &\wedge tag = (g^{sk})^{ch}\mathsf{PRF}_s(x)\}\end{aligned}$$

Note that the commitments here are commitments to elements of Z_p, thus, $\mathsf{Commit}(a, aux(a)) = \mathsf{GSCommit}$ $(h^a, aux(a))$ as described in Section 4.2.3.

Again, our proof system will use the Groth-Sahai Setup and SimSetup algorithms, and will satisfy the requirements for composable zero knowledge (see Section 2.1.3.), so that it can securely be combined with the P-signature scheme given in Section 4.2.3.

To see how this extension will work, first consider the NIZK proof system described in Section 5.3. Intuitively, it works by first proving that C'_y is a commitment to the correct output given C_s, C_x, and then proving that C'_y is a commitment to y. Here instead of this second step, we will keep the commitment C'_y, and then commit to tag as C'_{tag}, and prove that tag' is correct with respect to C'_y, C'_{sk}, and ch. Finally, we prove that C'_{sk} is a commitment to the same value as C_{sk} and that C'_{tag} is a commitment to tag.

Here we give a proof system for the language $\mathcal{L}_T$ presented in Section 9.2.

$\mathsf{Prove}(params, C_s, C_x, C_{sk}, tag, ch, s, aux(s), x, aux(x), sk, aux(sk))$. We first form new commitments $C'_s = \mathsf{GSCommit}(h^s, aux(s)')$, $C'_x = \mathsf{GSCommit}(h^x, aux(x)')$, and $C'_{sk} = \mathsf{GSCommit}(h^{sk}, aux(sk)')$. Then we compute zero knowledge proofs π_1 for (C_s,C'_s), π_2 for (C_x, C'_x), and π_3 for (C_{sk}, C'_{sk}), showing that both commitments in each pair commit to the same value using the techniques described in Section 3.5.

Next, we compute a commitment C'_y to $\mathsf{PRF}_s(x)$, and a commitment $C''_{sk} = \mathsf{GSCommit}(g^{sk}, aux(sk)'')$.

We then compute a GS witness indistinguishable proof π_4 that the value committed to in C'_y is the correct output given the seed in C'_s and the input in C'_x. I.e. that C'_y commits to Y, C'_s commits to S and C'_x commits to X such that $e(Y, SX) = e(g, h)$.

Next we compute a GS witness indistinguishable proof π_5 that the value committed to in C''_{sk} is correct with respect to C'_{sk}, i.e that C''_{sk} commits to K'' and C'_{sk} commits to K' such that $e(K'', h) = e(g, K')$.

We can also compute $C'_{tag} = {C''_{sk}}^{ch}C'_y$. Note that by the homomorphic properties of the commitment scheme, this means C'_{tag} should be a commitment to $(g^{sk})^{ch}y$ which is the correct value for tag.

Finally, we compute a zero knowledge proof π_6 that C'_{tag} is a commitment to tag as in Section 3.5.

The final proof is $\pi = (C'_s, C'_x, C'_{sk}, C'_y, C''_{sk}, C'_{tag}, \pi_1, \pi_2, \pi_3, \pi_4, \pi_5, \pi_6)$.

$\mathsf{VerifyProof}(params, C_s, C_x, C_{sk}, ch, \pi = (C'_s, C'_x, C'_{sk}, C'_y, C''_{sk}, C'_{tag}, \pi_1, \pi_2, \pi_3, \pi_4, \pi_5, \pi_6))$. Uses the Groth-Sahai verification procedure to verify $\pi_1, \pi_2, \pi_3, \pi_4, \pi_5, \pi_6$ with respect to $C_s, C_x, C_{sk}, info$, $C'_s, C'_x, C'_{sk}, C'_y, C''_{sk}, C'_z$.

9.2.1 Efficiency

The proof system above generates 6 new commitments (3 in G_1 and 3 in G_2), Groth-Sahai proofs for 2 pairing product equations one with $Q = 1$ and one with $Q = 2$, and 4 zero-knowledge proofs of equality of committed exponents (see Section 3.5), 1 for a pair of commitments to an element of G_1, and 3 for commitments to elements of G_2. Applying the efficiency formulas given in Sections 3.4 and 3.5, we get the following lemma:

Theorem 29. *When instantiated using the SXDH instantiation given in Section 3.3 the above proof system will have the following efficiency: Generating the proof will require* 148 *exponentiations in* G_1 *and* 140 *exponentiations in* G_2*. The resulting proof will consist of* 52 *elements of* G_1 *and* 48 *elements of* G_2*. Verifying the proof will involve computing* 204 *bilinear group pairings.*

When instantiated using the symmetric setting version of the DLIN instantiation given in Section 3.3, the above proof system will have the following efficiency: Generating the proof will require 558 *exponentiations in G. The resulting proof will consist of* 120 *elements in G. Verifying the proof will involve computing* 369 *bilinear group pairings.*

9.2.2 Security

Theorem 30. *The proof system* Setup, Prove, VerifyProof *is a secure composable zero knowledge proof system for the language* $\mathcal{L}_T(params)$ *described above, where* $params$ *is output by* Setup.

Proof. Correctness and Soundness follow from the corresponding properties of the underlying proof systems.

To prove zero knowledge, consider the following simulator algorithms:

$\mathsf{SimSetup}(1^k)$. runs the GS simulation setup to generate simulated parameters $params$ and trapdoor sim.

$\mathsf{SimProve}(params, sim, C_x, C_s, C_{sk}, tag, ch)$. We first choose random $s', x', sk' \leftarrow Z_p$, random opening information $aux(s)', aux(x)', aux(sk)'$ and form new commitments $C'_s = \mathsf{GSCommit}(h^{s'}, aux(s)')$, $C'_x = \mathsf{GSCommit}(h^{x'}, aux(x)')$,and $C'_{sk} = \mathsf{GSCommit}(h^{sk'}, aux(sk)')$.

Then we use the GS NIZK simulator to compute simulated zero knowledge proof π_1 that C_s and C'_s are commitments to the same value and simulated proof π_2 that C_x and C'_x are commitments to the same value, and simulated proof π_3 that C_{sk} and C'_{sk} are commitments to the same value using the techniques described in Section 3.5.

Next, we compute a commitment C'_y to $\mathsf{PRF}_{s'}(x')$ and C''_{sk} to $g^{sk'}$.

Then we compute a GS witness indistinguishable proof π_4 that the value committed to in C'_y is the correct output given the seed in C'_s and the input in C'_x. (Note that this statement is true given our choice of C'_y, C'_s, C'_x.)

Similarly, we compute a GS witness indistinguishable proof π_5 that the value committed to in C''_{sk} is correct with respect to the value committed to in C'_{sk}.

We also compute $C'_{tag} = C'_{sk}(C'_y)^{ch}$.

Finally, we use the GS simulator to generate simulated proof π_6 that C'_{tag} is a commitment to tag as in Section 3.5.

The final proof is $\pi = (C'_s, C'_x, C'_{sk}, C'_y, C''_{sk}, C'_{tag}, \pi_1, \pi_2, \pi_3, \pi_4, \pi_5, \pi_6)$.

Note that when parameters are generated by SimSetup, the proofs π_4 and π_5 and the commitments $C'_y, C'_s, C'_x, C'_{sk}, C'_y, C''_{sk}, C'_{tag}$ generated by SimProve are distributed identically to those generated by Prove. Further, by the composable zero knowledge properties of the GS NIZK for equality of committed values, the simulated proofs $\pi_1, \pi_2, \pi_3, \pi_6$ will also be distributed identically to those generated by the honest Prove algorithm. Thus, SimSetup, SimProve as described here satisfy the definition of zero knowledge for Setup, Prove, VerifyProof. □

9.3 New Compact E-Cash Scheme

In this section, we construct a compact e-cash scheme using our multi-block P-signatures and sVRF protocols.

Compact e-cash as defined by Camenisch et al. [21] lets a user withdraw multiple e-coins simultaneously. There are three types of players: a bank $\mathcal{I}$ as well as many users $\mathcal{U}$ and merchants $\mathcal{M}$(though merchants are treated as a special type of user). Please refer to [21] or Section 9.1 for protocol specifications and a definition of security.[2] We now show how to construct compact e-cash.

$\mathsf{CashSetup}(1^k)$. Runs $\mathsf{SetupSig}(1^k)$ to obtain $params_{PK}$. Our construction is non-blackbox: we reuse the GS NIPK proof system parameters $params_{GS}$ that is contained in $params_{PK}$. The parameters $params_{GS}$ in turn contain the setup for a bilinear pairing $params_{BM} = (p, G_1, G_2, G_T, e, g, h)$ for a paring $e : G_1 \times G_2 \to G_T$ for groups of prime order p. The algorithm returns $params_{PK}$.

$\mathsf{BankKG}(params_{EC}, n)$. The bank creates two P-signature keypairs, $(pk_w, sk_w) \leftarrow \mathsf{KeygenSig}(params_{PK})$ for issuing wallets and $(pk_c, sk_c) \leftarrow \mathsf{KeygenSig}(params_{PK})$ for signing coin indices. Then the bank computes a P-signature on the n coin indices $\Sigma_1, \ldots, \Sigma_n$, where $\Sigma_i = \mathsf{Sign}(sk_c, i)$. The bank's secret-key is $sk_{\mathcal{I}} = (sk_w, sk_c)$ and the bank's public-key is $(pk_w, pk_c, \Sigma_1, \ldots, \Sigma_n)$.

$\mathsf{UserKG}(params_{EC})$. The user picks $sk_{\mathcal{U}} \leftarrow Z_p^*$ and returns $(pk_{\mathcal{U}} = e(g, h)^{sk_{\mathcal{U}}}, sk_{\mathcal{U}})$.

Merchants generate their keys in the same way but also have a publicly known identifier $id_{\mathcal{M}} = f(pk_{\mathcal{M}})$ associated with their public keys (f is some publicly known mapping).

$\mathsf{Withdraw}(\mathcal{U}(params_{EC}, pk_{\mathcal{I}}, sk_{\mathcal{U}}, n), \mathcal{I}(params_{EC}, pk_{\mathcal{U}}, sk_{\mathcal{I}}, n))$. The user obtains a wallet from the bank.

[2]The original [21] definition had an interactive Spend protocol, while we break it up into two non-interactive protocols: $\mathsf{SpendCoin}(params,\ W, pk_{\mathcal{M}}, info)$ and $\mathsf{VerifyCoin}(params, pk_{\mathcal{M}}, pk_{\mathcal{I}},\ coin)$. The merchant sends the user a $info$, the user runs SpendCoin and gives the resulting e-coin for the merchant to verify using VerifyCoin). We prefer to use a non-interactive spend protocols because often two-way communication is not available or impractical, e.g. when sending an e-coin by email.

1. The user picks $s', t' \leftarrow Z_p$ computes $comm_{sk} = \mathsf{Commit}(sk_{\mathcal{U}}, open_{sk_{\mathcal{U}}})$, $comm_{s'} = \mathsf{Commit}(s', open_{s'})$, and $comm_{t'} = \mathsf{Commit}(t', open_{t'})$ and sends $comm_{sk}$, $comm_{s'}$, and $comm_{t'}$ to the bank. The user proves in zero knowledge that he knows the opening to these values, and that $comm_{sk}$ corresponds to the secret key used for computing $pk_{\mathcal{U}}$.[3]
2. If the proofs verify the bank sends the user random values $s'', t'' \in Z_p$.
3. The user picks random $open_s, open_t$, commits to $comm_s = \mathsf{Commit}(s' + s'', open_s)$, and $comm_t = \mathsf{Commit}(t' + t'', open_t)$, sends $comm_s$ and $comm_t$ to the bank, and proves that they are formed correctly. Let $s = s' + s''$ and $t = t' + t''$.
4. The user and bank run $\mathsf{ObtainSig}(params_{PK}, pk_w, (comm_{sk}, comm_s, comm_t), (sk_{\mathcal{U}}, s, t)) \leftrightarrow \mathsf{IssueSig}(params_{PK}, sk_w, (comm_{sk}, comm_s, comm_t))$ respectively. The user obtains a P-signature σ on $(sk_{\mathcal{U}}, s, t)$. The user stores the wallet $W = (s, t, pk_{\mathcal{I}}, \sigma, n)$, the bank stores tracing information $T_W = pk_{\mathcal{U}}$.

$\mathsf{SpendCoin}(params_{EC}, (s, t, pk_{\mathcal{I}}, \sigma, J), pk_{\mathcal{M}}, info)$ The user calculates serial number $S = \mathsf{PRF}_s(J)$. The user needs to prove that he knows a signature σ on $(sk_{\mathcal{U}}, s, t)$ and a signature Σ_J on J such that $S = \mathsf{PRF}_s(J)$.

Next the user constructs a double-spending equation $T = (g^{id_{\mathcal{M}}\|info})^{sk_{\mathcal{U}}}\mathsf{PRF}_t(J)$.[4] The user proves that T is correctly formed for the $sk_{\mathcal{U}}, t, J$, signed in σ and Σ_J.

All these proofs need to be done non-interactively. We now give more details. The user runs the P-signature $\mathsf{ProveSig}$, first on σ and pk_w to obtain commitments and proof $((C_{ID}, C_s, C_t), \pi_1) \leftarrow \mathsf{ProveSig}(params_{PK}, pk_w, \sigma, (sk_{\mathcal{U}}, s, t))$ for $sk_{\mathcal{U}}, s, t$ respectively and second on Σ_J and pk_c to obtain commitment and proof $(C_J, \pi_2) \leftarrow \mathsf{ProveSig}(params_{PK}, pk_c, \Sigma_J, J)$ for J.

Then the user constructs non-interactive zero-knowledge proofs that indeed $(S, T, C_{ID}, C_s, C_t, C_J, id_{\mathcal{M}}\|info)$ are well formed.

This is done by computing two proofs π_S and π_T. π_S proves that $(C_s, C_J, S) \in \mathcal{L}_S$ and is computed as described in Section 5.3, where $\mathcal{L}_S$ is defined as:

$$\begin{aligned}\mathcal{L}_S = \{C_s, C_x, y \mid\ & \exists x, s, aux(x), aux(s) \text{ such that} \\ & C_s = \mathsf{Commit}(s, aux(s)) \wedge C_x = \mathsf{Commit}(x, aux(x)) \wedge y = F_s(x)\}\end{aligned}$$

π_T proves that $(C_t, C_J, C_{ID}, T, (id_{\mathcal{M}}|info)) \in \mathcal{L}_T$ and is computed as described in Section 9.2, where $\mathcal{L}_T$ is defined as:

$$\begin{aligned}\mathcal{L}_T = \{C_s, C_x, C_{sk}, tag, ch \mid\ & \exists x, s, sk, aux(x), aux(s), aux(sk) \text{ such that} \\ & C_s = \mathsf{Commit}(s, aux(s)) \wedge C_x = \mathsf{Commit}(x, aux(x)) \wedge C_{sk} = comm(sk, aux(xsk)) \\ & \wedge\ tag = (g^{sk})^{ch} F_s(x)\}\end{aligned}$$

[3]These and the rest of the proofs in the issue protocol can be done using efficient sigma protocols [28, 39] and their zero-knowledge compilers [38].

[4]The merchant is responsible for assuring that *info* is locally unique. Coins which have the same serial number and the same $id_{\mathcal{M}}\|info$ cannot be deposited and the damage lies with the merchant. The dangers that users get cheated by verifiers that do not accept coins with correct *info* can be mitigated using techniques such as endorsed e-cash [26].

The user outputs a $coin = (S, T, C_{ID}, C_s, C_t, C_J, \pi_1, \pi_2, \pi_S, \pi_T, id_{\mathcal{M}} \| info)$.

$\mathsf{VerifyCoin}(params_{EC}, pk_{\mathcal{M}}, pk_{\mathcal{I}}, coin)$. Parses $coin$ as $(S, (T, C_{ID}, C_s, C_t, C_J, \pi_1, \pi_2, \pi_S, \pi_T), id_{\mathcal{M}}' \| info)$ and checks that the following checks succeed: (1) Check that $id_{\mathcal{M}}' = f(pk_{\mathcal{M}})$. (2) $\mathsf{VerifySig}(params_{PK}, pk_w, \pi_1, (C_{ID}, C_s, C_t)) = \mathsf{accept}$. (3) $\mathsf{VerifySig}(params_{PK}, pk_c, \pi_2, C_J) = \mathsf{accept}$. (4) $\mathsf{VerifyProof}_{\mathcal{L}_S}(params_{GS}, (C_s, C_J, S), \pi_S) = \mathsf{accept}$. (5) $\mathsf{VerifyProof}_{\mathcal{L}_T}(params_{GS}, (C_t, C_J, C_{ID}, T, (id_{\mathcal{M}} | info)), \pi_T) = \mathsf{accept}$. Note that the merchant is responsible for assuring that $info$ is unique over all of his transactions. Otherwise his deposit might get rejected by the following algorithm.

$\mathsf{Deposit}(params_{EC}, pk_{\mathcal{I}}, pk_{\mathcal{M}}, coin, state_{\mathcal{I}})$. T he algorithm parses the coin as $coin = (S, T, C_{ID}, C_s, C_t, C_J, \pi_1, \pi_2, \pi_S, \pi_T, id_{\mathcal{M}} \| info)$ and performs the same checks as VerifyCoin. The bank maintains a database $state_{\mathcal{I}}$ of all previously accepted coins. The output of the algorithm is an updated database $state'_{\mathcal{I}} = state_{\mathcal{I}} \cup \{coin\}$ and the flag $result$, that is computed as follows:

(i) If the coin verifies and if no coin with serial number S is stored in $state_{\mathcal{I}}$, $result = \mathsf{accept}$ to indicates that the coin is correct and fresh. The bank deposits the value of the e-coin into the merchant's account and adds $coin$ to $state_{\mathcal{I}}$.

(ii) If the coin doesn't verify or if there is a coin with the same serial number and the same $id_{\mathcal{M}} \| info$ already stored in $state_{\mathcal{I}}$, $result = \mathsf{merchant}$ to indicate that the merchant cheated. The bank refuses to accept the e-coin because the merchant failed to properly verify it.

(iii) If the coin verifies but there is a coin with the same serial number S but different $id_{\mathcal{M}} \| info$ in $state_{\mathcal{I}}$, $result = \mathsf{user}$ to indicate that a user doublespend. The bank pays the merchant (who accepted the e-coin in good faith) and punishes the double-spending user.

$\mathsf{Identify}(params_{EC}, pk_{\mathcal{I}}, coin_1, coin_2)$. The algorithm allows the bank to identify a double-spender. Parse $coin_1 = (S, (T, C_{ID}, C_s, C_t, C_J, \pi_1, \pi_2, \pi_S, \pi_T), id_{\mathcal{M}1} \| info_1)$ and $coin_2 = (S', (T', C'_{ID}, C'_s, C'_t, C'_J, \pi'_1, \pi'_2, \pi'_S, \pi'_T), id_{\mathcal{M}2} \| info_2)$. The algorithm aborts if one of the coins doesn't verify, if $S \neq S'$, or if $id_{\mathcal{M}1} \| info_1 = id_{\mathcal{M}2} \| info_2$. Otherwise, the algorithm outputs $T_W = pk_{\mathcal{U}} = e((T/T')^{1/(id_{\mathcal{M}1} \| info_1 - id_{\mathcal{M}2} \| info_2), h)}$, which the bank compares to the trace information it stores after each withdrawal transaction.

9.4 Efficiency

Here we will consider the efficiency of the SpendCoin and VerifyCoin algorithms. A coin includes values $S, T \in G_1$, a P-signature proof for a signature on 3 messages, a P-signature proof for a signature on 1-message, a proof π_S generated using the proof system described in Section 5.3, and a proof π_T generated using the proof system described in Section 9.2. The most appropriate P-signature construction is the multi-block P-signature construction described in Section 4.2.3. The following theorem describes the efficiency of the resulting algorithms for spending and verifying e-coins:

Theorem 31. *When the underlying Groth-Sahai proofs are instantiated using the SXDH instantiation given in Section 3.3 the above e-cash system will have the following efficiency: Generating an e-coin will require*

463 *exponentiations in* G_1 *and* 460 *exponentiations in* G_2*. The resulting coin will consist of* 152 *elements of* G_1 *and* 150 *elements of* G_2*. Verifying the e-coin will involve computing* 624 *bilinear group pairings.*

When the underlying Groth-Sahai proofs are instantiated using the symmetric setting version of the DLIN instantiation given in Section 3.3, the above e-cash system will have the following efficiency: Generating an e-coin will require 1812 *exponentiations in* G*. The resulting coin will consist of* 365 *elements in* G*. Verifying the e-coin will involve computing* 1143 *bilinear group pairings.*

9.5 Proof of Security of Our Compact E-Cash Scheme

This proof makes use of the security definitions of P-Signatures as of [5] and the standard security notions of pseudo-random functions and zero-knowledge proof systems. We refer the reader to [5] for a definition of the *Signer privacy*, *User privacy*, *Correctness*, *Unforgeability*, and *Zero-knowledge* properties of a P-signature scheme and the corresponding simulator protocols (SimIssueSig, SimObtainSig,SimSetupSig, SimProveSig) and extraction algorithms (ExtractSetupSig, ExtractSig) of a P-signature scheme.

Theorem 32. *The e-cash scheme presented in Section 9.3 is a secure e-cash scheme given the security of the P-signature scheme, the PRF, and the NIZK proof system.*

Proof. We need to prove that CashSetup, BankKG, UserKG, SpendCoin, VerifyCoin, Deposit, Identify, and the interactive protocol Withdraw fulfill the *Correctness*, *Anonymity*, *Balance*, and *Identification* properties.

Correctness. Correctness is straight forward.

Anonymity. Consider the following simulator $\mathsf{Sim} = (\mathsf{SimCashSetup}, \mathsf{SimSpend})$:

$\mathsf{SimCashSetup}(1^k)$. Runs $\mathsf{SimSetupSig}(1^k)$ to obtain $params_{PK}, sim_P$. Our construction is non-blackbox: we reuse the GS-NIZK proof system parameters $params_{GS}$ that are contained in $params_{PK}$ and the GS NIZK simulation parameters sim_{GS} contained in sim_P. The parameters $params_{GS}$ in turn contain the setup for a bilinear pairing $params_{BM} = (p, G_1, G_2, G_T, e, g, h)$ for a paring $e : G_1 \times G_2 \to G_T$ for groups of prime order p. The algorithm returns $(params_{PK}, sim_P)$.

$\mathsf{SimSpend}(params, sim, pk_{\mathcal{I}}, pk_{\mathcal{M}}, info)$.

- The simulator uses $\mathsf{SimProveSig}(params_{PK}, sim_P, pk_w, 3)$ to compute $((C_{ID}, C_s, C_t), \pi_1)$
- The simulator uses $\mathsf{SimProveSig}(params_{PK}, sim_P, pk_c, 1)$ to compute (C_J, π_2).
- The simulator picks a random serial number and double spending tag $S, T \leftarrow G_1$ and simulates the non-interactive zero-knowledge proofs π_S and π_T using the zero-knowledge simulator for $\mathcal{L}_S$ and $\mathcal{L}_T$.

We consider a sequence of 5 Games:

Game 1. Corresponds to the game $\mathcal{A}$ plays when interacting with $\mathcal{O}_{\mathsf{Spend}}(params, pk_{\mathcal{I}}, \cdot, \cdot)$.

Game 2. As Game 1, except that $\mathsf{CashSetup}$ is replaced by $\mathsf{SimCashSetup}$ to obtain sim_P.

Game 3. As Game 2, except that the oracle uses sim_P and $\mathsf{SimProveSig}$ to compute $((C_{ID}, C_s, C_t), \pi_1)$.

Game 4. As Game 3, except that the oracle uses sim_P and $\mathsf{SimProveSig}$ to compute (C_J, π_2).

Game 5. As Game 4, except that the oracle uses sim_{GS} and the zero-knowledge simulator for languages L_S and L_T.

Game 6. As Game 5, except that S and T are now chosen at random. This corresponds to the game with $\mathcal{O}_{\mathsf{SimSpend}}(params, pk_{\mathcal{I}}, \cdot, \cdot, \cdot)$.

Games 1 and 2 are indistinguishable by the properties of the P-signature scheme.

A non-negligible probability to distinguish between Games 2 and 3 and between Games 3 and 4 allows to break the zero-knowledge property of the P-signature scheme. A non-negligible probability to distinguish between Games 4 and 5 breaks the zero knowledge property of the proof system.

A non-negligible probability to distinguish between Games 5 and 6 allows to break the pseudorandomness of the PRF through the following reduction. The reduction either gets oracle access to two pseudorandom functions $\mathsf{PRF}_s(.)$ and $\mathsf{PRF}_t(.)$ or to two random functions.[5] It can simulate all the rest of the spend without knowing s, t given the simulators. In one case it's Game 5, in the other case it's Game 6. If a distinguisher can distinguish between the two games we can break the pseudorandomness of the PRF.

As Game 6 corresponds to the view generated by the simulator, the success probability of an adversary in breaking the *anonymity* property is bounded by the sum of the distinguishing advantages in the above games. This advantage is negligible.

Balance. A successfully deposited coin can be parsed as $coin = (S, (T, C_{ID}, C_s, C_t, C_J, \pi_1, \pi_2, \pi_S, \pi_T), id_{\mathcal{M}} \| info)$. We consider multiple games.

Game 1. The first game is the same as the balance definition with the oracles using the real protocol.

Game 2. As Game 1 except that in $\mathsf{CashSetup}$ algorithm $\mathsf{SetupSig}$ is replaced with $\mathsf{ExtractSetupSig}$ to obtain td.

Game 3. As Game 2 except that in $\mathcal{O}_{\mathsf{Deposit}}$ the game checks every deposited coin and uses td and the $\mathsf{ExtractSig}$ algorithm to extract y_s, y_t and y_{ID} from C_{ID}, C_s, C_t, π_1 and y_J from C_J, π_2. It aborts if the triple (y_s, y_t, y_J) already appeared in a previously deposited coin.

Game 4. As Game 3 except that it also aborts if the value y_J is not in $\{F(1), \dots, F(n)\}$.

Game 5. As Game 4 except that it abort if the number of deposited coins with different (y_s, y_t) pairs is bigger than $withdrawals$.

Games 1 and 2 are indistinguishable as $\mathsf{SetupSig}$ and $\mathsf{ExtractSetupSig}$ are indistinguishable.

Games 2 and 3 are indistinguishable because Game 3 aborts only with negligible probability. An abort can occur only if one of the $F(y_s)^{-1}$, $F(y_t)^{-1}$, $F(y_J)^{-1}$ does not correspond to the opening of C_s, C_t, C_J

[5]A standard hybrid argument can be used to show that S and T can be replaced one after the other.

in which case we found a forgery for one of the two P-signatures or if we broke the soundness of the proof system used to prove language $\mathcal{L}_S$ and $\mathcal{L}_T$. We guess which of the three options is the case to do a reduction and break the unforgeability of the P-signature scheme or the soundness of the proof system.

A distinguisher between Game 3 and 4 allows to break the unforgeability of the P-signature scheme as the only time Game 4 aborts is when it obtains a signature on a value $F^{-1}(y_J) > n$. As such a J value was never signed we obtain a P-signature forgery.

A distinguisher between Game 4 and 5 allows to break the unforgeability of the P-signature scheme as the number of different (y_s, y_t) pairs with corresponding signatures pairs is greater than the number of correctly generated signatures be $\mathcal{O}_{\mathsf{Withdraw}}$. We create a reduction and guess which (y_s, y_t) is the forgery to break P-signature unforgeability with probability at least $1/(\mathit{withdrawals} + 1)$.

In Game 5, we can bound the number of successful deposits to at most $\mathit{withdrawals} \cdot n$. The success probability of $\mathcal{A}$ is bounded by the sum of the distinguishing probabilities between the Games 1 to 5. This probability is negligible.

Identification. A successful adversary $\mathcal{A}$ in the identification game outputs two coins $(\mathit{coin}_1, \mathit{coin}_2)$ that verify and have the same serial number S but different $\mathit{id}_{\mathcal{M}} \| \mathit{info}$. We consider multiple games.

Game 1. Is the same as the original security game.

Game 2. As Game 1 but in CashSetup algorithm SetupSig is replaced with ExtractSetupSig to obtain td.

Game 3. As Game 2 but the game parses the coins $\mathit{coin}_1 = (S, (T, C_{ID}, C_s, C_t, C_J, \pi_1, \pi_2, \pi_S, \pi_T),$ $pk_{\mathcal{M}1}, \mathit{info}_1)$ and $\mathit{coin}_2 = (S, (T', C'_{ID}, C'_s, C'_t, C'_J, \pi'_1, \pi'_2, \pi'_S, \pi'_T), \mathit{id}_{\mathcal{M}2} \| \mathit{info}_2)$ obtained from the adversary and uses td and the ExtractSig algorithm to extract y_{ID}, y_s, y_t, y_J from $(C_{ID}, C_s, C_t, \pi_1)$, (C_J, π_2) and $y'_{ID}, y'_s, y'_t, y'_J$ from $(C'_{ID}, C'_s, C'_t, \pi_1), (C_J, \pi_2)$. Game 3 aborts if the values y_J or y'_J are not in $\{F(1), \ldots, F(n)\}$.

Game 4. As Game 3 but it also aborts if $y_s \neq y'_s$.

Game 5. As Game 4 but it also aborts if $y_J \neq y'_J$.

Game 6. As Game 5 but it aborts if $y_t \neq y_t$ or $y_{ID} \neq y'_{ID}$.

Games 1 and 2 are indistinguishable as SetupSig and ExtractSetupSig are indistinguishable.

A distinguisher between Game 2 and 3 allows to break the unforgeability of the P-signature scheme as the only time Game 3 aborts is when it obtains a signature on a value $F^{-1}(y_J) > n$. As such a J value was never signed we obtain a P-signature forgery.

Games 3 and 4 are indistinguishable. The abort in Game 4 can only happens with probability greater than n^2/p ($\sim$ the probability of collision if S is computed by a random function) in one of the following 4 cases: (i. and ii.) one of the $F(y_s)^{-1}$, $F(y'_s)^{-1}$ does not correspond to the opening of C_s, C'_s respectively (in this case we can find a forgery for the P-signatures scheme), iii. we break the soundness of the proof system for language $\mathcal{L}_S$, or iv. we break the pseudorandomness of PRF.

The reduction to the pseudo-randomness works as follows: We have a polynomial sized domain, so an adversary in the pseudorandomness game can compute the output on every element in the domain. By pseudorandomness, this should look like a completely random polynomially sized subset of Ga. So the probability that two randomly chosen seeds will produce intersecting ranges should be negligible. Otherwise we can build a reduction which breaks the pseudorandomness property without even seeing the seed: we are given oracle access to the PRF (or a random function). We query it on all points in the domain. Then we choose another random seed and compute that on all points in the domain. If there is an intersection with the first set, we output "pseudo random" otherwise we output "random".

Note that $n < poly(k)$ and thus the Games 3 and 4 are indistinguishable, or otherwise we guess which of the 4 cases mentioned above holds do the appropriate reduction.

Games 4 and 5 are indistinguishable for the same reason except that the probability of a collision for perfectly random functions corresponds to the probability of distinguishing a random function from a random permutation given only $n < poly(k)$ queries.

Games 5 and 6 are indistinguishable as aborts in 6 occur only with negligible probability. In Game 6 we abort if $y_t \neq y_t$ or $y_{ID} \neq y'_{ID}$. As the seed s is chosen at random, it is highly unlikely that two withdrawn wallets contain the same seed. Consequently $y_t = y_t$ and $y_{ID} = y_{ID}$ or we break the unforgeability of the P-signature scheme.

In Game 6 the probability of $e((T/T')^{1/(id_{\mathcal{M}1} \| info_1 - id_{\mathcal{M}2} \| info_2)}, h) \notin DB_T$ is bounded by the soundness error of the proof protocol or the probability that y_{ID} was never signed by the bank or does not correspond to the commitment C_{ID}. If the probability of the first is non-negligible we break the soundness of the proof protocol, if the probability of the latter is non-negligible we break the unforgeability of our P-signature scheme.

As Games 1 to 6 are computationally indistinguishable, $\mathcal{A}$'s success probability in the real game is also negligible.

Weak Exculpability. A successful adversary $\mathcal{A}$ in the weak exculpability game outputs two coins ($coin_1$, $coin_2$) that verify and have the same serial number S but different $id_{\mathcal{M}} \| info$.

While $\mathcal{A}$ knows the users public key $pk_{\mathcal{U}} = e(g^{sk_{\mathcal{U}}}, h)$ it is hard to compute $g^{sk_{\mathcal{U}}}$ from $pk_{\mathcal{U}}$ alone without knowing $sk_{\mathcal{U}}$. As no additional information about $g^{sk_{\mathcal{U}}}$ is revealed until $\mathcal{U}$ reuses an e-token, an adversary computing $g^{sk_{\mathcal{U}}}$ can be used to asymmetric computation Diffie Hellman (asymmetric CDH) assumption: given random g^a, h^b compute g^{ab}. Asymmetric CDH is implied by q-BDDH and DLIN.

More formally we define a sequence of games to eliminate all sources of information that an adversary may potentially have about honest user's keys besides the public key learned through a $\mathcal{O}_{\mathsf{GetKey}}$ query:

Game 1. Here the adversary plays the same game as in the weak exculpability definition.

Game 2. As Game 1, but CashSetup is replaced with SimCashSetup.

Game 3. As Game 2, but in $\mathcal{O}_{\mathsf{Withdraw}}$ algorithm ObtainSig is replaced with the P-signature simulator SimObtainSig.

Game 4. As Game 3, but in $\mathcal{O}_{\mathsf{Spend}}$ algorithm SpendCoin is replaced with SimSpend.

Games 1 and 2 are indistinguishable by the properties of the P-signature scheme (and the anonymity of the e-cash scheme itself). If Games 2 and 3 can be distinguish we break the user privacy of the P-signature scheme. Games 3 and 4 are indistinguishable based on the anonymity of the e-cash scheme.

In order to break the asymmetric CDH assumption we do the following reduction. We answer a random $\mathcal{O}_{\mathsf{GetKey}}$ with $e(g^a, h^b)$. Then we use the simulators $\mathsf{SimCashSetup}$, $\mathsf{SimObtainSig}$, and $\mathsf{SimSpend}$ to simulate all interactions with the adversary where this user is involved. A successful adversary outputs g^{ab} as $(T/T')^{1/(id_{\mathcal{M}1} \| info_1 - id_{\mathcal{M}2} \| info_2)}$.

$\square$

Appendix A

Generic Group Security for BB-SDH and BB-CDH

A.0.1 BB-HSDH

We provide more confidence in the BB-HSDH assumption by proving lower bounds on the complexity of the problem in the generic group model. Note that this also establishes hardness of the HSDH assumption for generic groups as conjectured by [14].

Definition 35 (BB-HSDH)**.** *Let $c_1 \dots c_q \leftarrow Z_p$. On input $g, g^x, u \in G_1$, $h, h^x \in G_2$ and $\{g^{1/(x+c_\ell)}, c_\ell\}_{\ell=1\dots q}$, it is computationally infeasible to output a new tuple $(g^{1/(x+c)}, h^c, u^c)$.*

Let $e : G_1 \times G_2 \to G_T$ be a bilinear map over groups G_1, G_2, G_T of prime order p. In the generic group model, we encode group elements of G_1, G_2, G_T as unique random strings. Group operations are performed by an oracle, that operates on these strings and keeps an internal representation of the group. We set $\alpha : Z_p \to \{0,1\}^*$ to be the opaque encoding of elements of G_1. Let g be a generator of G_1; $\alpha(a)$ maps $a \in Z_p$ to the string representation of $g^a \in G_1$. The function $\beta : Z_p \to \{0,1\}^*$ maps $a \in Z_p$ to the string representation of $h^a \in G_2$ and the function $\tau : Z_p \to \{0,1\}^*$ maps $a \in Z_p$ to $e(g,h)^a \in G_T$.

We can represent all operations in terms of the random maps α, β, τ. Note that the maps do not need to be explicitly given. It is sufficient if the oracles create them using lazy evaluation. Let $a, b \in Z_p$. We define oracle queries for the following operations:

Group Operation. $\alpha(a) \cdot \alpha(b) = \alpha(a+b)$. This is because $\alpha(a) \cdot \alpha(b) = g^a \cdot g^b = g^{a+b} = \alpha(a+b)$. The same holds for the group operation in β and τ.

Exponentiation by a Constant. $\alpha(b)^a = \alpha(ab)$. This is because $\alpha(b)^a = (g^b)^a = g^{ab} = \alpha(ab)$. The same holds for multiplication by a constant in β and τ.

Pairing. $e(\alpha(a), \beta(b)) = \tau(ab)$. This is because $e(\alpha(a), \beta(b)) = e(g^a, h^b) = e(g,h)^{ab} = \tau(ab)$.

When an adversary tries to break the BB-HSDH assumption to the generic group model, the adversary does not get g, g^x, u, h, h^x and $\{g^{1/(x+c_\ell)}, c_\ell\}_{\ell=1\dots q}$ as input. Instead, we encode these values using the random maps α, β, τ. The generators g, h and $e(g,h)$ become $\alpha(1)$, $\beta(1)$ and $\tau(1)$ respectively. We encode g^x as $\alpha(x)$, $g^{1/x+c_\ell}$ as $\alpha(1/(x+c_\ell)$ and h^x as $\beta(x)$. Since g is a generator of G_1, there exists a $y \in Z_p$ such that $g^y = u$. So we choose y at random and set $u = \alpha(y)$.

To break the BB-HSDH assumption, the adversary needs to output a triple (A, B, C) of the form $(g^{1/x+c}$, $h^c, u^c)$ for some $c \in Z_p$. Normally, we can test that the triple is well-formed using the bilinear map: $e(A, h^x B) = e(g,h) \wedge e(C,h) = e(u,B)$.

In the generic group model we require that he outputs random representations $(\alpha_A, \beta_B, \alpha_C)$. The adversary can either compute these values using the group oracles, or pick them at random, which he could also do by doing a generic exponentiation with a random constant. Thus it is meaningful to say that the adversary succeeds if $\alpha_A = \alpha(1/(x+c)) \wedge \beta_B = \beta(c) \wedge \alpha_C = \alpha(yc)$.

Theorem 33 (BB-HSDH is Hard in Generic Group Model)**.** *Let G_1, G_2, G_T be groups of prime order p, (where p is a k-bit prime) with bilinear map $e : G_1 \times G_2 \to G_T$. We choose maps α, β, τ at random. There exists a negligible function $\nu : \mathbb{N} \to [0,1]$ such that for every PPTM $\mathcal{A}$:*

$$\Pr[x, y, \{c_\ell\}_{\ell=1\dots q} \leftarrow Z_p; (\alpha_A, \beta_B, \alpha_C) \leftarrow \mathcal{A}(\alpha(1), \alpha(x), \alpha(y), \beta(1), \beta(x), \{\alpha(1/(x+c_\ell)), c_\ell\}_{\ell=1\dots q}) :$$
$$\exists c : \alpha_A = \alpha(1/x+c) \wedge \beta_B = \beta(c) \wedge \alpha_C = \alpha(yc)] \leq \nu(k)$$

Proof. Let $\mathcal{A}$ be a PPTM that can break the BB-HSDH assumption. We create an environment $\mathcal{E}$ that interacts with $\mathcal{A}$ as follows:

$\mathcal{E}$ maintains three lists: $L_\alpha = \{(F_{\alpha,s}, \alpha_s) : s = 0, \dots, S_\alpha - 1\}$, $L_\beta = \{(F_{\beta,s}, \beta_s) : s = 0, \dots, S_\beta - 1\}$, and $L_\tau = \{(F_{\tau,s}, \tau_s) : s = 0, \dots, S_\tau - 1\}$. The $F_{\alpha,s}, F_{\beta,s}, F_{\tau,s}$ contain emphrational functions; their numerators and denominators are polynomials in $Z_p[x,c]$. $\mathcal{E}$ uses $F_{\alpha,s}, F_{\beta,s}, F_{\tau,s}$ to store the group action queries that $\mathcal{A}$ makes and $\alpha_s, \beta_s, \tau_s$ to store the results. Thus $\alpha_s = \alpha(F_{\alpha,s})$, $\beta_s = \beta(F_{\beta,s})$ and $\tau_s = \tau(F_{\tau,s})$.

$\mathcal{E}$ chooses random strings $\alpha_0, \alpha_1, \alpha_2, \{\alpha_{2+\ell}\}_{\ell=1\dots q}, \beta_0, \beta_1, \tau_0 \in \{0,1\}^*$, and sets the corresponding polynomials as:

$$\begin{array}{llll} F_{\alpha,0} = 1 & F_{\alpha,1} = x & F_{\alpha,2} = y & F_{\alpha,2+\ell} = 1/(x+c_\ell) \\ F_{\beta,0} = 1 & F_{\beta,1} = x & & \\ F_{\tau,0} = 1 & & & \end{array}$$

$\mathcal{E}$ sets $S_\alpha = 3 + q$, $S_\beta = 2$ and $S_\tau = 1$. Then $\mathcal{E}$ sends the strings to $\mathcal{A}$. Whenever $\mathcal{A}$ calls the group action oracle, $\mathcal{E}$ updates its lists.

Multiplication. $\mathcal{A}$ inputs α_s and α_t. $\mathcal{E}$ checks that α_s and α_t are in its list L_α, and returns $\perp$ if they are not. Then $\mathcal{E}$ computes $F = F_{\alpha,s} + F_{\alpha,t}$. If F is already in the list L_α, then $\mathcal{E}$ returns the appropriate α_v. Otherwise, $\mathcal{E}$ adds chooses a random α_{S_α}, sets $F_{\alpha,S_\alpha} = F$ and adds this new tuple to the list. $\mathcal{E}$ increments the counter S_α by 1. $\mathcal{E}$ performs a similar operation if the inputs are in G_2 or G_T.

Division. $\mathcal{A}$ inputs α_s and α_t. $\mathcal{E}$ checks that α_s and α_t are in its list L_α, and returns $\perp$ if they are not. Then $\mathcal{E}$ computes $F = F_{\alpha,s} - F_{\alpha,t}$. If F is already in the list L_α, then $\mathcal{E}$ returns the appropriate α_v. Otherwise,

$\mathcal{E}$ adds chooses a random α_{S_α}, sets $F_{\alpha,S_\alpha} = F$ and adds this new tuple to the list. $\mathcal{E}$ increments the counter S_α by 1. $\mathcal{E}$ performs a similar operation if the inputs are in G_2 or G_T.

Exponentiation by a Constant $\mathcal{A}$ inputs α_s and a constant $a \in Z_p$. $\mathcal{E}$ checks that α_s is in its list L_α, and returns $\perp$ if it is not. Then $\mathcal{E}$ computes $F = F_{\alpha,s} \cdot a$. If F is already in the list L_α, then $\mathcal{E}$ returns the appropriate α_v. Otherwise, $\mathcal{E}$ adds chooses a random α_{S_α}, sets $F_{\alpha,S_\alpha} = F$ and adds this new tuple to the list. $\mathcal{E}$ increments the counter S_α by 1. $\mathcal{E}$ performs a similar operation if the inputs are in G_2 or G_T.

Pairing. $\mathcal{A}$ inputs α_s and β_t. $\mathcal{E}$ checks that α_s and β_t are in its lists L_α and L_β, respectively, and returns $\perp$ if they are not. Then $\mathcal{E}$ computes $F = F_{\alpha,s} \cdot F_{\beta,t}$. If F is already in the list L_τ, then $\mathcal{E}$ returns the appropriate τ_v. Otherwise, $\mathcal{E}$ adds chooses a random τ_{S_τ}, sets $F_{\tau,S_\tau} = F$ and adds this new tuple to the list. $\mathcal{E}$ increments the counter S_τ by 1.

At the end of the game, $\mathcal{A}$ outputs $(\alpha_A, \beta_B, \alpha_C)$. These values must correspond to bivariate polynomials $F_{\alpha,A}$, $F_{\beta,B}$ and $F_{\alpha,C}$ in our lists. (If one of these values is not in our lists, then $\mathcal{A}$ must have guessed a random group element; he might as well have asked the oracle to perform exponentiation on a random constant and added a random value to the list. Thus we ignore this case.)

Since $\mathcal{A}$ must have computed these polynomials as a result of oracle queries, they must be of the form $a_0 + a_1 x + a_2 y + \sum_{\ell=1}^{q} a_3/(x + c_\ell)$. If $\mathcal{A}$ is to be successful,

$$F_{\alpha,A}(x + F_{\beta,B}) = 1 \text{ and} \tag{A.1}$$

$$F_{\alpha,C} = yF_{\beta,B}. \tag{A.2}$$

For Equation (2) to hold identically in $Z_p[x, y]$, $F_{\alpha,C}$ and $F_{\beta,B}$ either need to be 0 or $F_{\beta,B}$ needs to be a constant, because the only possible term for y in the polynomials is $a_2 y$. In both cases, the term $(x + F_{\beta,B})$ in Equation (2) has degree 1, and Equation (1) can only be satisfied identically in $Z_p[x, y]$ if $F_{\alpha,A}$ has degree $\geq > p - 2$. We know that the degree of $F_{\alpha,A}$ is at most q and conclude that there exists an assignment in Z_p to the variables x and y for which the Equations (1) and (2) do not hold. Since Equation (1) is a non-trivial polynomial equation of degree $\leq 2q$, it admits at most $2q$ roots in Z_p.

Analysis of $\mathcal{E}$'s Simulation. At this point $\mathcal{E}$ chooses a random $x^*, y^* \in \mathbb{Z}_p^*$, and now sets $x = x^*$ and $y = y^*$. $\mathcal{E}$ now tests (in equations A.3, A.4, A.5, and A.6) if its simulation was perfect; that is, if the instantiation of x by x^* or y by y^* does *not* create any equality relation among the polynomials that was not revealed by the random strings provided to $\mathcal{A}$. Thus, $\mathcal{A}$'s overall success is bounded by the probability that any of the following holds:

$$F_{\alpha,i}(x^*, y^*) - F_{\alpha,j}(x^*, y^*) = 0 \text{ in } Z_p, \text{ for some } i, j \text{ such that } F_{\alpha,i} \neq F_{\alpha,j}, \tag{A.3}$$

$$F_{\beta,i}(x^*, y^*) - F_{\beta,j}(x^*, y^*) = 0 \text{ in } Z_p, \text{ for some } i, j \text{ such that } F_{\beta,i} \neq F_{\beta,j}, \tag{A.4}$$

$$F_{\tau,i}(x^*, y^*) - F_{\tau,j}(x^*, y^*) = 0 \text{ in } Z_p, \text{ for some } i, j \text{ such that } F_{\tau,i} \neq F_{\tau,j}, \tag{A.5}$$

$$F_{\alpha,A}(x^*, y^*)(x^* + F_{\beta,B}(x^*, y^*)) = 1 \wedge F_{\alpha,C}(x^*, y^*) = y^* F_{\beta,B}(x^*, y^*) \text{ in } Z_p. \tag{A.6}$$

Each polynomial $F_{\alpha,i}$, $F_{\beta,i}$ and $F_{\tau,i}$ has degree at most q, q and $2q$, respectively.

For fixed i and j, we satisfy equations A.3 and A.4 with probability $\leq q/(p-1)$ and equations A.5 with probability $\leq 2q/(p-1)$. We can bound the probability that Equation A.6 holds by $\leq 2q/(p-1)$

Now summing over all (i, j) pairs in each case, we bound $\mathcal{A}$'s overall success probability

$$\epsilon \leq 2\binom{S_\alpha}{2}\frac{q}{p-1} + \binom{S_\tau}{2}\frac{2q}{p-1} + \frac{2q}{p-1} \leq \frac{2q}{p-1}(\binom{S_\alpha}{2} + \binom{S_\tau}{2} + 1).$$

Let q_G be the total number of group oracle queries made, then we know that $\S_\alpha + S_\beta + S_\tau = q_G + q + 6$. We obtain that $\epsilon \leq (q_G + q + 6)^2 \frac{2q}{p-1} = O(q_G^2 q/p + q^3/p)$.

□

The following corollary restates the above result:

Corollary. *Any adversary that breaks the BB-HSDH assumption with constant probability $\epsilon > 0$ in generic bilinear groups of order p such that $q < O(\sqrt[3]{p})$ requires $\Omega(\sqrt[3]{\epsilon p/q})$ generic group operations.*

A.0.2 BB-CDH

We provide more confidence in the BB-CDH assumption by proving that it is implied by the SDH assumption [11]. We require either a homomorphism $\phi : G_1 \to G_2$ or an extended SDH challenge. As SDH is secure in the generic group model, BB-CDH is too.

Definition 36 (BB-CDH). *On input $g, g^x, g^y \in G_1$, $h, h^x \in G_2$, $c_1, \ldots, c_q \leftarrow Z_p$ and $g^{\frac{1}{x+c_1}}, \ldots, g^{\frac{1}{x+c_q}}$, it is computationally infeasible to output a g^{xy}.*

On input a SDH challenge $(g_1, g_1^z, g_1^{z^2}, \ldots, g_1^{z^{q+1}}, g_2, g_2^z)$, our reduction tries to compute $g^{\frac{1}{z}}$.

There is a technical subtlety. We also want to give our reduction $g_2^{z^2}, \ldots, g_2^{z^{q+1}}$. In [11], SDH is proven for a setting where homomorphisms between G_1 and G_2 exist. The generic group proof still holds for a setting where instead the elements $g_2^{z^2}, \ldots, g_2^{z^{q+1}}$ are given as part of the challenge.

Now we pick random $c_1, \ldots, c_q \leftarrow Z_p$ set $g = g_1^{z\prod_{i=1}^q(1+c_iz)}$, $X = g^x = g_1$, $Y = g^y = g_1^r$, $h = g_2^{z\prod_{i=1}^q(1+c_iz)}$, $Z = h^x = g_2$.

Note that implicitly $x = 1/z$ and $y = r/z$. As $z\prod_{i=1}^q(1+c_iz)$ is a polynomial of maximum degree $q+1$ the reduction can compute g and h.

Now we need to compute $g^{1/(x+c_i)} = g^{1/(\frac{1}{z}+c_i)} = g^{z/(1+c_iz)}$. Substituting g with $g_1^{z\prod_{i=1}^q(1+c_iz)}$ we get

$$g^{z/(1+c_iz)} = (g_1^{z\prod_{j=1}^q(1+c_jz)})^{z/(1+c_iz)}$$
$$g_1^{z^2\prod_{j=1,j\neq i}^q(1+c_jz)}$$

The polynomial $z^2\prod_{j=1,j\neq i}^q(1+c_jz)$ is again of maximum degree $q+1$. Thus the reduction can compute the $g^{1/(x+c_i)}$ values.

We are ready to query our BB-CDH adversary to obtain g^{xy}. We know that

$$\begin{aligned} g^{xy} &= g^{r/(z^2)} \\ &= (g_1^{z\prod_{i=1}^q(1+c_iz)})^{r/z^2}) \\ &= g_1^{r(\sum_{i=1}^q a_iz^{q-i})+1/z} \end{aligned}$$

The value $r(\sum_{i=1}^{q} a_i z^{q-i})$ is a polynomial of degree $q-1$ in z. We can compute $g_1^{r(\sum_{i=1}^{q} a_i z^{q-i})}$ and obtain $g_1^{1/z}$ through division.

The following corollary follows from the generic group proof of the SDH assumption:

Corollary. *Any adversary that breaks the BB-CDH assumption with constant probability $\epsilon > 0$ in generic bilinear groups of order p such that $q < O(\sqrt[3]{p})$ requires $\Omega(\sqrt[3]{\epsilon p/q})$ generic group operations.*

Bibliography

[1] Lucas Ballard, Matthew Green, Breno de Medeiros, and Fabian Monrose. Correlation-Resistant Storage. Johns Hopkins University, CS Technical Report # TR-SP-BGMM-050705. `ht\tp://spar.isi.jhu.edu/~mgreen/correlation.pdf`, 2005.

[2] Endre Bangerter, Jan Camenisch, and Anna Lysyanskaya. A cryptographic framework for the controlled release of certified data. In *Twelfth International Workshop on Security Protocols*, Lecture Notes in Computer Science. Springer Verlag, 2004.

[3] Mira Belenkiy, Jan Camenisch, Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Hovav Shacham. Delegatable anonymous credentials. (in submission).

[4] Mira Belenkiy, Melissa Chase, Chris Erway, John Jannotti, Alptekin Küpçü, Anna Lysyanskaya, and Eric Rachlin. Making p2p accountable without losing privacy. In *Proceedings of the Sixth Workshop on Privacy in the Electronic Society (WPES)*. ACM Press, 2007.

[5] Mira Belenkiy, Melissa Chase, Markulf Kohlweiss, and Anna Lysyanskaya. P-signatures and non-interactive anonymous credentials. In *TCC '08*, 2008.

[6] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *First ACM Conference on Computer and Communication Security*, pages 62–73. Association for Computing Machinery, 1993.

[7] Manuel Blum, Alfredo De Santis, Silvio Micali, and Guiseppe Persiano. Non-interactive zero-knowledge. *SIAM Journal of Computing*, 20(6):1084–1118, 1991.

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

[9] Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM Journal on Computing*, 13(4):850–863, November 1984.

[10] Dan Boneh and Xavier Boyen. Efficient selective id secure identity based encryption without random oracles. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology — EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*. Springer, 2004.

[11] Dan Boneh and Xavier Boyen. Short signatures without random oracles. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology — EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 54–73. Springer, 2004.

[12] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In Matthew K. Franklin, editor, *Advances in Cryptology — CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 41–55. Springer Verlag, 2004.

[13] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-dnf formulas on ciphertexts. In *TCC 2005*, volume 3378 of *Lecture Notes in Computer Science*, pages 325–341. Springer, 2005.

[14] Xavier Boyen and Brent Waters. Full-domain subgroup hiding and constant-size group signatures. In *Public Key Cryptography*, pages 1–15, 2007.

[15] Stefan Brands. *Rethinking Public Key Infrastructure and Digital Certificates— Building in Privacy*. PhD thesis, Eindhoven Institute of Technology, Eindhoven, The Netherlands, 1999.

[16] Ernie Brickell, Jan Camenisch, and Liqun Chen. Direct anonymous attestation. Technical Report Research Report RZ 3450, IBM Research Division, March 2004.

[17] Ernie Brickell, Jan Camenisch, and Liqun Chen. Direct anonymous attestation. In *Proc. 11th ACM Conference on Computer and Communications Security*, pages 225–234. acm press, 2004.

[18] Jan Camenisch and Els Van Herreweghen. Design and implementation of the *idemix* anonymous credential system. In *Proc. 9th ACM Conference on Computer and Communications Security*. acm press, 2002.

[19] Jan Camenisch and Els Van Herreweghen. Design and implementation of the *idemix* anonymous credential system. Technical Report Research Report RZ 3419, IBM Research Division, May 2002.

[20] Jan Camenisch, Susan Hohenberger, Markulf Kohlweiss, Anna Lysyanskaya, and Mira Meyerovich. How to win the clonewars: efficient periodic n-times anonymous authentication. In *CCS '06: Proceedings of the 13th ACM conference on Computer and communications security*, pages 201–210, New York, NY, USA, 2006. ACM Press.

[21] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Compact E-cash. In Ronald Cramer, editor, *Advances in Cryptology — Eurocrypt 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 302–321. Springer, 2005.

[22] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Balancing accountability and privacy using e-cash. In *SCN*, 2006.

[23] Jan Camenisch and Anna Lysyanskaya. Efficient non-transferable anonymous multi-show credential system with optional anonymity revocation. In Birgit Pfitzmann, editor, *Advances in Cryptology — EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 93–118. Springer Verlag, 2001.

[24] Jan Camenisch and Anna Lysyanskaya. A signature scheme with efficient protocols. In Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano, editors, *Security in Communication Networks, Third International Conference, SCN 2002*, volume 2576 of *Lecture Notes in Computer Science*, pages 268–289. Springer Verlag, 2003.

[25] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In Matthew K. Franklin, editor, *Advances in Cryptology — CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 56–72. Springer Verlag, 2004.

[26] Jan Camenisch, Anna Lysyanskaya, and Mira Meyerovich. Endorsed e-cash. In *IEEE Symposium on Security and Privacy*, pages 101–115, 2007.

[27] Jan Camenisch and Victor Shoup. Practical verifiable encryption and decryption of discrete logarithms. In Dan Boneh, editor, *Advances in Cryptology — CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 126–144, 2003.

[28] Jan Camenisch and Markus Stadler. Proof systems for general statements about discrete logarithms. Technical Report TR 260, Institute for Theoretical Computer Science, ETH Zürich, March 1997.

[29] Sébastien Canard, Aline Gouget, and Emeline Hufschmitt. A handy multi-coupon system. In *ACNS*, pages 66–81, 2006.

[30] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004.

[31] Melissa Chase and Anna Lysyanskaya. On signatures of knowledge. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 78–96, 2006.

[32] Melissa Chase and Anna Lysyanskaya. A direct construction of simulatable vrfs with applications to multi-theorem nizk. In *Advances in Cryptology — CRYPTO '07*, 2007.

[33] David Chaum. Blind signatures for untraceable payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *Advances in Cryptology — Proceedings of CRYPTO '82*, pages 199–203. Plenum Press, 1983.

[34] David Chaum. Blind signature systems. In David Chaum, editor, *Advances in Cryptology — CRYPTO '83*, page 153. Plenum Press, 1984.

[35] David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10):1030–1044, October 1985.

[36] David Chaum, Amos Fiat, and Moni Naor. Untraceable electronic cash. In Shafi Goldwasser, editor, *Advances in Cryptology — CRYPTO '88*, volume 403 of *Lecture Notes in Computer Science*, pages 319–327. Springer Verlag, 1990.

[37] Jean-Sibastien Coron. On the exact security of full domain hash. In Mihir Bellare, editor, *Advances in Cryptology — CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 229–235. Springer Verlag, 2000.

[38] Ivan Damgård. Efficient concurrent zero-knowledge in the auxiliary string model. In Bart Preneel, editor, *Advances in Cryptology — EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 431–444. Springer Verlag, 2000.

[39] Ivan Damgård. On Σ-protocols. Available at `http://www.daimi.au.dk/~ivan/Sigma.ps`, 2002.

[40] Ivan Damgård, Kasper Dupont, and Michael Østergaard Pedersen. Unclonable group identification. In *EUROCRYPT*, pages 555–572, 2006.

[41] Ivan Bjerre Damgård. Payment systems and credential mechanism with provable security against abuse by individuals. In Shafi Goldwasser, editor, *Advances in Cryptology — CRYPTO '88*, volume 403 of *Lecture Notes in Computer Science*, pages 328–335. Springer Verlag, 1990.

[42] Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. Non-interactive zero-knowledge proof systems. In Carl Pomerance, editor, *Advances in Cryptology — CRYPTO '87*, volume 293 of *Lecture Notes in Computer Science*, pages 52–72. Springer-Verlag, 1988.

[43] Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In *Public Key Cryptography 2005*, pages 416–432, 2005.

[44] Cynthia Dwork and Moni Naor. Zaps and their applications. In *FOCS*, pages 283–293, 2000.

[45] Cynthia Dwork, Moni Naor, Omer Reingold, and Larry J. Stockmeyer. Magic functions. *J. ACM*, 50(6):852–921, 2003.

[46] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple noninteractive zero knowledge proofs under general assumptions. *SIAM Journal on Computing*, 29(1):1–28, 1999.

[47] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology — CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer Verlag, 1987.

[48] Eiichiro Fujisaki and Tatsuaki Okamoto. A practical and provably secure scheme for publicly verifiable secret sharing and its applications. In Kaisa Nyberg, editor, *Advances in Cryptology — EUROCRYPT '98*, volume 1403 of *Lecture Notes in Computer Science*, pages 32–46. Springer Verlag, 1998.

[49] S. Galbraith and V. Rotger. Easy decision diffie-hellman groups. *Journal of Computation and Mathematics*, 7:201–218, 2004.

[50] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, October 1986.

[51] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity and a method of cryptographic protocol design. In *Proc. 27th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 174–187. IEEE Computer Society Press, 1986.

[52] Shafi Goldwasser and Yael Tauman Kalai. On the (in)security of the Fiat-Shamir paradigm. In *Proc. 44th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 102–115. IEEE Computer Society Press, 2003.

[53] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. In *Proc. 27th Annual Symposium on Foundations of Computer Science*, pages 291–304, 1985.

[54] Shafi Goldwasser, Silvio Micali, and Ronald Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, April 1988.

[55] Jens Groth. Simulation-sound nizk proofs for a practical language and constant size group signatures. In *ASIACRYPT*, pages 444–459, 2006.

[56] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero knowledge for NP. In *Advances in Cryptology — Eurocrypt '06*, Lecture Notes in Computer Science. Springer-Verlag, 2006.

[57] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. http://eprint.iacr.org/2007/155, 2007.

[58] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal of Computing*, 28(4):1364–1396, 1999.

[59] Markus Jakobsson, Kazue Sako, and Russell Impagliazzo. Designated verifier proofs and their applications. In Ueli Maurer, editor, *Advances in Cryptology — EUROCRYPT '96*, volume 1233 of *Lecture Notes in Computer Science*. Springer, 1996.

[60] Stanislaw Jarecki and Vitaly Shmatikov. Hancuffing big brother: an abuse-resilient transaction escrow scheme. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology — EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 590–608. Springer, 2004.

[61] Stanislaw Jarecki and Vitaly Shmatikov. Efficient two-party secure computation on committed inputs. In *EUROCRYPT*, pages 97–114, 2007.

[62] Jonathan Katz. Efficient and non-malleable proofs of plaintext knowledge and applications. In *EUROCRYPT*, pages 211–228, 2003.

[63] Joe Kilian and Erez Petrank. An efficient noninteractive zero-knowledge proof system for np with general assumptions. *J. Cryptology*, 11(1):1–27, 1998.

[64] Anna Lysyanskaya. *Signature schemes and applications to cryptographic protocol design*. PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, September 2002.

[65] Anna Lysyanskaya, Ron Rivest, Amit Sahai, and Stefan Wolf. Pseudonym systems. In Howard Heys and Carlisle Adams, editors, *Selected Areas in Cryptography*, volume 1758 of *Lecture Notes in Computer Science*. Springer Verlag, 1999.

[66] Silvio Micali, Michael Rabin, and Salil Vadhan. Verifiable random functions. In *Proc. 40th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 120–130. IEEE Computer Society Press, 1999.

[67] Moni Naor. Bit commitment using pseudorandomness. *Journal of Cryptology*, 4(2):51–158, 1991.

[68] Pascal Paillier. Public-key cryptosystems based on composite residuosity classes. In Jacques Stern, editor, *Advances in Cryptology — EUROCRYPT '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–239. Springer Verlag, 1999.

[69] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology – CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer Verlag, 1992.

[70] Alfredo De Santis, Giovanni Di Crescenzo, and Giuseppe Persiano. Randomness-efficient non-interactive zero-knowledge (extended abstract). In *ICALP*, pages 716–726, 1997.

[71] Alfredo De Santis, Giovanni Di Crescenzo, and Giuseppe Persiano. Necessary and sufficient assumptions for non-interactive zero-knowledge proofs of knowledge for all NP relations. In Ugo Montanari, José P. Rolim, and Emo Welzl, editors, *Proc. 27th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 1853 of *Lecture Notes in Computer Science*, pages 451–462. Springer Verlag, 2000.

[72] Mike Scott. Authenticated id-based key exchange and remote log-in with insecure to ken and pin number. `http://eprint.iacr.org/2002/164`, 2002.

[73] Isamu Teranishi and Kazue Sako. k-times anonymous authentication with a constant proving cost. In *Public Key Cryptography*, pages 525–542, 2006.

[74] Eric R. Verheul. Evidence that xtr is more secure than supersingular elliptic curve cryptosystems. *J. Cryptology*, 17(4):277–296, 2004.

[75] Victor K. Wei. More compact e-cash with efficient coin tracing. Cryptology ePrint Archive, Report 2005/411, 2005. `http://eprint.iacr.org/`.

[76] Andrew C. Yao. How to generate and exchange secrets. In *Proc. 27th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 162–167, 1986.