An Extension of the Groth-Sahai Proof System

Sarah Meiklejohn
smeiklej@cs.brown.edu
Sc.B., Brown University, 2008

Thesis Advisor: Anna Lysyanskaya

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in the Department
of Computer Science at Brown University

Providence, RI 02912

May 1, 2009

Abstract

Non-interactive zero-knowledge proofs, particularly those constructed on top of bilinear groups, have been significantly studied in cryptography and used in a wide variety of applications in recent years. One very powerful suite of techniques for proofs over bilinear groups is the Groth-Sahai proof system, which provides efficient non-interactive witness-indistinguishable and zero-knowledge proofs without relying on any one assumption or language.

The Groth-Sahai suite of proofs has already been used in a number of applications, including group signature schemes, anonymous voting, and anonymous credentials. In this paper, we describe a technique that allows us to prove that two GS commitments open to the same value. We then use this technique to create a non-interactive zero-knowledge protocol that satisfies a stronger version of zero-knowledge than the original GS protocol. Finally, we use both these techniques to provide a new extension of the proof system that makes it into a non-interactive zero-knowledge proof of knowledge of an exponent. We then outline new building blocks based on this technique that can be used in a variety of applications. The main application of our technique will involve using it to construct a fully unforgeable non-interactive anonymous credentials scheme.

Contents

Chapter 1

Introduction

Non-interactive zero-knowledge (NIZK) proofs have become increasingly important in cryptographic theory and practice in the last few years. Initially introduced by Blum et al. [BFM88], there has been much exploration of various constructions and applications [FLS90, Dam92, KP98] and it has been shown that NIZK proofs exist for all languages in NP. Unfortunately, although known proofs do yield interesting theoretical results, most are too inefficient to be used in practice.

The root of this inefficiency is mainly the reliance on using one particular NP-complete language, such as 3-SAT or Circuit-SAT. To create a proof for a general statement it is first necessary to reduce the original statement to a statement in the appropriate language; while this is certainly feasible, it is often quite computationally expensive. Rather than attempt to fix this problem by trying to use a different NP-complete language, Groth and Sahai [GS08] focus instead on a set of *group-dependent* languages. Their work is based on the observation that many cryptographic protocols are based on finite abelian groups; they focus more specifically on groups that induce a bilinear map. Using these groups, they are able to construct efficient non-interactive zero-knowledge proofs that are not limited to one specific language, and furthermore are not even based on one specific cryptographic assumption. The generality with which these proofs are outlined means that in addition to being much more efficient than any previous constructions, their proofs are also highly flexible. Groth and Sahai outline proofs of satisfiability of many different kinds of equations, such as multi-exponentiation equations, multi-scalar multiplication equations, pairing product equations, and quadratic equations (in this paper, I talk just about the last two), but there is nothing to suggest that these same techniques can't be extended to prove satisfiability of other types of equations. Similarly, although the instantiations they highlight are based on specific $\mathbb{Z}_p$- and $\mathbb{Z}_n$-modules, the techniques can be extended to work for other groups that induce a bilinear map under a variety of cryptographic assumptions.

Because of their flexibility, there have been many applications of the proofs of Groth and Sahai. One application is Groth's construction of an efficient group signature scheme [Gro07]. Briefly, group signatures allow a member of a group to anonymously sign messages on behalf of the group. A group manager controls group membership and also has the power to open a signature and reveal the identity of the signer if a member has abused his privileges. Although group signatures have been studied for many years, most schemes given are either too inefficient to be used in practice or require the random oracle model to prove security. Using GS proofs, Groth constructs the first group signature scheme where the keys and signatures both contain a constant number of group elements and the proof of security does not require the random oracle model.

In an extension using the GS proofs for satisfiability of both multi-exponentiations and pairing product equations, Groth and Lu [GL07] give the first non-interactive proof for the correctness of a shuffle, where a shuffle is defined as a permutation and re-encryption of a set of ciphertexts. Because it is the first proof that works non-interactively, this construction has significant applications in the development of mix-nets useful for anonymous voting and broadcasting schemes.

Another application of GS proofs has been work in anonymous credentials. Belenkiy et al. [BCKL08] provide the first non-interactive construction of unforgeable anonymous credentials based on GS proofs and a signature scheme inspired by the Boneh-Boyen signature scheme [BB04]. Anonymous credentials is a very useful problem, as it allows a user Alice to prove to another user Bob that she has been given a certificate by a third user Carol in a secure manner; i.e. without Bob or Carol linking Alice's proof to Alice's certificate (and thus to Alice herself). In addition, if Alice then proves her possession of a credential to a fourth user Dave, the proofs that Alice gives to Bob and Dave cannot be linked to each other and so the authentication scheme remains anonymous.

In this thesis, I focus on further applications of GS proofs. More specifically, I focus on the use of GS proofs in the setting of proving satisfiability of quadratic equations, which can often give us better efficiency results than using pairing product equations. I also extend the GS proofs to allow the prover to construct a non-interactive zero-knowledge proof of knowledge of an exponent, whereas in the original system it is only possible to prove knowledge of a group element. While the proof of knowledge of an exponent is less efficient than the proof of knowledge of a group element, there are many applications where it is nevertheless useful. To demonstrate one of these applications, I show how this technique can be used to simplify the cryptographic assumptions used by Belenkiy et al. and construct an unforgeable non-interactive anonymous credentials scheme that uses the Boneh-Boyen signature scheme directly.

Chapter 2

Definitions and Notation

The following definitions and discussions will be essential in our exploration of Groth-Sahai proofs, and of non-interactive zero-knowledge proofs in general. We first go over some definitions from probability, and then discuss the idea of a complexity class (specifically P, NP, and NP-complete). In the third section, we discuss one-way functions and commitment schemes. In the fourth and fifth sections, we introduce interactive protocols and discuss what it means for an interactive protocol to be zero-knowledge or a proof of knowledge. Finally, we bring all this notation together to introduce non-interactive zero-knowledge proofs. Many of the definitions in this chapter are taken from [Gol01], and many of them appear in my undergraduate thesis [Mei08].

2.1 Probability notation

Definition 2.1.1. *A function $\mu : \mathbb{N} \to \mathbb{R}$ is called* negligible *if for all positive polynomials $p(\cdot)$ there exists a value N such that $\mu(n) < \frac{1}{p(n)}$ for all $n \geq N$.*

A good example of a negligible function is $f(x) = 2^{-x}$, or any other function that eventually gets very close to zero. These functions appear everywhere in cryptography, especially in definitions of security. It is often hard to guarantee that an event occurs with probability exactly 0, so saying that the probability an event occurs is negligible is often as close as we can get.

Definition 2.1.2. *Two distribution ensembles $\{D_n\}_{n\in\mathbb{N}}$ and $\{E_n\}_{n\in\mathbb{N}}$ are* computationally indistinguishable *if for all probabilistic polynomial time algorithms $\mathcal{A}$, we have*

$$|Pr_{x\leftarrow D_n}[\mathcal{A}(x) = 1] - Pr_{x\leftarrow E_n}[\mathcal{A}(x) = 1]| = \nu(n)$$

for some negligible function $\nu(\cdot)$.

All this definition says is that given a random element x and a reasonable amount of computation, we should still be able to do only negligibly better than a random guess when deciding if it came from D_n or E_n. There are also two important variants on this definition to consider. The first definition is called *statistical indistinguishability*, in which we remove the condition that $\mathcal{A}$ run in polynomial time (this is also sometimes called *information-theoretic* indistinguishability). The second is called *perfect indistinguishability* and it occurs when we require that $\nu(k) = 0$ for all k; that is that the distributions are in fact identical.

2.2 Complexity classes

The idea of a *complexity class* is a group of problems that have related levels of difficulty. In computer science, unlike in many other disciplines, we can actually define what it means for a problem to be "hard." Intuitively, the more steps a problem requires, the harder it is. The first complexity class we look at is the group of problems that can be solved using polynomially many steps.

Definition 2.2.1. *A language L is* recognizable in polynomial time*, or $L \in P$, if there exists a deterministic Turing machine M and polynomial $p(\cdot)$ such that*

1. *On input x, M halts after at most $p(|x|)$ steps, and*
2. *$M(x) = 1$ if and only if $x \in L$.*

For now, we don't need to worry about the Turing machine and what it means to take $p(|x|)$ steps; we will see all this formally in Section 2.4. All the definition really says is that if we are given an element x, after a reasonable amount of computation we should be able to say definitively if x is in the related language or not.

Next we see a group of languages known as NP, where each language L is *verifiable* in polynomial time (and not necessarily *recognizable* as it was with P).

Definition 2.2.2. *$L \in NP$ if there exists a binary relation $R_L \subseteq \{0,1\}^* \times \{0,1\}^*$ and a polynomial $p(\cdot)$ such that $R_L \in P$, and $x \in L$ if and only if there exists a y such that $|y| \leq p(|x|)$ and $(x, y) \in R_L$.*

In the above definition, the value y is referred to as a *witness* to the fact that $x \in L$. In general, the set of all witnesses can be denoted $R_L(x)$. The definition simply says that if $L \in$ NP, then for all $x \in L$ there must exist a witness y that demonstrates this fact, and that once given this valid witness we should be able to verify (in polynomial time) that x is in fact in L.

Next we look at what it means for a language to be NP-complete. To define this class, we first need to define what it means to be able to reduce one language to another.

Definition 2.2.3. *L is* polynomially reducible to L' *if there exists a polynomial-time computable function f such that $x \in L$ if and only if $f(x) \in L'$.*

Definition 2.2.4. *L is* NP-complete *if $L \in NP$ and every language in NP is polynomially reducible to it.*

Essentially, the problems that are NP-complete are the most difficult problems in NP, as a solution for one would generate a solution for all other problems in NP (because of this reducibility property).

2.3 One-way functions and commitment schemes

One-way functions are one of the most basic building blocks in cryptography. No one has yet shown that P $\neq$ NP so we cannot actually say that one-way functions exist, but assuming their existence is considered to be a relatively weak cryptographic assumption. The intuition behind a one-way function is something that is easy to compute but hard to invert.

Definition 2.3.1. *A function $f : \{0,1\}^* \to \{0,1\}^*$ is called* one-way *if it satisfies the following two conditions:*

1. *Easy to compute: there exists a polynomial-time algorithm $\mathcal{A}$ such that $\mathcal{A}(x) = f(x)$ for all $x \in \{0,1\}^*$.*
2. *Hard to invert: for all polynomial-time algorithms $\mathcal{A}$ there exists a negligible function $\nu(\cdot)$ such that*

$$Pr[x \leftarrow \{0,1\}^n, y = f(x), x' \leftarrow \mathcal{A}(y) : f(x') = y] < \mu(n).$$[1]

In addition, if the pre-image x is unique (so if $f(x) = f(x')$ it must hold that $x = x'$) we call f a *one-way permutation.*

There are many cryptographic primitives that are based on the assumption of one-way functions, but one that will be very important for us is the idea of a commitment scheme. A commitment scheme should be seen as the cryptographic equivalent of a sealed envelope. The sender S will take a certain value v and send a commitment of this value to the receiver R. The commitment acts as the envelope; with just the committed value the receiver still doesn't know the value v, but at any point the sender can open the envelope, reveal v, and prove that this was the value inside the envelope all along. More formally, let's say that both S and R receive as common input a security parameter 1^n and that $\mathbf{Com}(x; r)$ represents a commitment to the input $x \in \{0,1\}^k$ using randomness $r \in \{0,1\}^n$. Then:

[1]This experiment notation can be a bit confusing. In English, we are saying that, given a random x in the domain of f and $y = f(x)$, the probability that an algorithm $\mathcal{A}$ outputs a value x' such that $f(x') = y$ is negligible.

Definition 2.3.2. *A* perfectly binding commitment scheme for k-bit strings *is a probabilistic algorithm* $\boldsymbol{Com}$ *satisfying the following:*

1. *Perfect binding:* $\boldsymbol{Com}(x; r) \neq \boldsymbol{Com}(y; s)$ *for any* $x, y \in \{0,1\}^k$, $r, s \in \{0,1\}^n$.
2. *Computational hiding: for any* $x, y \in \{0,1\}^k$, *the two probability ensembles (where* U_n *represents the uniform distribution)* $\{\boldsymbol{Com}(x; U_n)\}_{n\in\mathbb{N}}$ *and* $\{\boldsymbol{Com}(y; U_n)\}_{n\in\mathbb{N}}$ *are computationally indistinguishable.*

Note that here we pick perfect binding, which means that once we have committed to a particular value, there is no other value to which the commitment can be opened. It is also possible to have perfect hiding, which means that the two probability ensembles are in fact identical. It is not possible, however, to have both these properties at once: if one is perfect, the other can only be computational.

2.4 Interactive machines

Definition 2.4.1 ([GMR85]). *An* interactive Turing machine *is a deterministic multi-tape Turing machine consisting of the following seven tapes: a read-only input tape, a read-only random tape, a read-and-write work tape, a write-only output tape, a pair of communication tapes where one is read-only and the other is write-only, and a read-and-write switch tape that consists of a single cell.*

This definition may seem extremely complicated, but seen in the context of an interactive protocol it is actually quite straightforward. The input and output tapes are fairly self-explanatory, and the work tape can be thought of as where the Turing machine keeps track of what it has done so far.

The switch tape works to signify that the machine is either idle or active. This makes the most sense in the context of pairs of machines, when we think about alternating moves - when one machine becomes idle the other becomes active. When active, it is that machine's turn in the protocol, and once it is finished it changes the switch cell to indicate that it is now the other machine's turn to move. If one machine halts without changing the switch cell, we know that the protocol is now terminated and we should examine the final output tapes. The machines can also share their communication tapes: the read-only tape for a machine represents the messages it has received from the other, and the write-only tape represents the messages it sends.

Finally, we need to consider the notion of randomness. The Turing machine we have described above is *probabilistic* because it has the addition of a random tape. Without the random tape, upon receiving the same input the machine will always output the same answer and is then known as *deterministic*. We will almost always talk about probabilistic polynomial time Turing machines (PPT for short).

Definition 2.4.2. *An interactive machine $\mathcal{A}$ has* time complexity $t : \mathbb{N} \to \mathbb{N}$ *if for all interactive machines $\mathcal{B}$ and strings x, it holds that when interacting with $\mathcal{B}$ on common input x, $\mathcal{A}$ always halts within $t(|x|)$ steps.*

In particular, we say that $\mathcal{A}$ is *polynomial-time* if there exists a polynomial $p(\cdot)$ such that $\mathcal{A}$ has time complexity p. We also need to discuss the idea of *expected polynomial time*, which means that the running time is 'polynomial on the average.' There are a number of ways to try to define this formally, but it is really best to use intuition and say that it means that the algorithm usually runs in polynomial time, but there might be cases where it runs slower (but also cases where it runs faster!).

Definition 2.4.3. *A two-party game between a prover P and a verifier V is called an* interactive argument system for a language L *if both the prover and the verifier run in probabilistic polynomial time (PPT) and it satisfies the following two properties:*

1. *Completeness: for every $x \in L$ the verifier will always accept after interacting with the prover on common input x.*
2. *Soundness: for some negligible function $\nu(\cdot)$ and for all $x \notin L$ and potentially cheating provers P^*, the verifier V will reject with probability at most $\nu(|x|)$ after interacting with P^* on common input x.*

We can note that if the prover P is allowed to be computationally unbounded, the above game becomes an *interactive proof system* instead of an argument system.

This definition should hopefully be intuitive. The completeness property just means that both parties can be satisfied that the protocol was run in a fair way, and the soundness property protects the verifier from a cheating prover. In the next section we see a way to protect the prover from a cheating verifier.

An important concepts associated with an interactive proof (or argument) system is the idea of a *view*. The view can be defined for either the prover or the verifier, and consists of the public and private inputs, the random tape, and a list of all the previous messages. The verifier's view will be denoted as $view_V^P(x)$, while the prover's view can be written $view_P^V(x)$.

2.5 Zero knowledge and proofs of knowledge

The zero-knowledge property is the most important thing we need to consider in this chapter. Informally, this property means that when a verifier is engaged in a zero-knowledge proof that a statement T is true, at the end of the protocol he learns nothing beyond the validity of the statement. This can be stated more formally as follows:

Definition 2.5.1. *An interactive proof system* (P, V) *is called* zero-knowledge *if for all PPT verifiers* V^* *there exists a PPT simulator* S_{V^*} *such that the ensembles* $\{view^P_{V^*}(x)\}_{x \in L}$ *and* $\{S_{V^*}(x)\}_{x \in L}$ *are computationally indistinguishable.*

At first glance, this definition may seem somewhat non-intuitive. All it is saying, however, is that at the end of an interaction, the verifier cannot be sure if he was talking to the actual prover P or some simulator S who did not know any valid witness. This may seem impossible, but we will see in our next section that the simulator is given some extra power, such as a trapdoor that allows it to open commitments in many different ways. We will use the notation V^* or P^* instead of just V or P to represent a potentially cheating verifier or prover.

There are many different variations on zero-knowledge. For example, if the ensembles described in the definition are identical rather than just indistinguishable, the protocol is known as *perfect zero-knowledge*, which means that the simulator produced an exact replica of a possible conversation with the prover. Another stronger version of zero-knowledge is *auxiliary-input zero-knowledge*, which accounts for an auxiliary input y given to the verifier before the protocol begins. Formally,

Definition 2.5.2. *An interactive proof system* (P, V) *is* auxiliary-input zero-knowledge *if for all PPT verifiers* V^* *there exists a PPT simulator* S_{V^*} *and a value* y *such that* $|y| = p(|x|)$ *for some polynomial* $p(\cdot)$ *such that the ensembles* $\{view^P_{V^*}(x, y)\}_{x \in L}$ *and* $\{S_{V^*}(x)\}_{x \in L}$ *are computationally indistinguishable.*

This definition simply accounts for possibly malicious or cheating verifiers that have managed to find out some additional piece of information. In fact, this version of zero-knowledge has been so useful that it is now considered the standard definition rather than the definition given before.

Another important concept in proof systems is the idea of a *proof of knowledge*. In an ordinary proof system, all the prover needs to show is the validity of the statement, namely that $x \in L$. This is known as a proof of membership. A proof of knowledge takes the idea one step further, as the prover now needs to show that it actually *knows* a witness w proving the fact that $x \in L$. But what does it mean to say that a machine knows something? Formally,

Definition 2.5.3. *A proof (or argument) system* (P, V) *for a language* L *is a* proof of knowledge with knowledge error $\kappa(\cdot)$ *if it satisfies:*

1. *Completeness: if* P *has a* w *such that* $(x, w) \in R_L$, V *accepts.*
2. *Knowledge extraction: there exists a polynomial* $p(\cdot)$ *and a* knowledge extractor E *such that if the prover's probability of making the verifier accept is* $\epsilon(x)$, *after interacting with the prover the extractor outputs a solution* $s \in R_L(x)$ *in expected time* $\frac{p(|x|)}{\epsilon(x) - \kappa(x)}$.

What exactly does this mean? The knowledge extractor is analogous to a simulator: it has black-box access to the prover, and it also may have access to some trapdoor. At the end of the day, the knowledge extractor is successful if it manages to retrieve a valid witness w. This is known as a proof of knowledge because if an extractor is able to extract a valid witness from the prover, the prover must have had knowledge of such a witness all along.

There do exist proof systems which are capable of satisfying both the above definitions at the same time; these are known as *zero-knowledge proofs of knowledge* and must satisfy the completeness, soundness, zero-knowledge, and knowledge extraction properties outlined in Definitions 2.4.3, 2.5.2, and 2.5.3. We will see more on such proof systems in Chapters 3 and 4.

2.6 Non-interactive zero knowledge

One final property we want to consider is when the game run in Definition 2.4.3 has only one round: a single message sent from the prover to the verifier. In this case, we call the proof system *non-interactive*. Formally,

Definition 2.6.1 ([BFM88]). *A* non-interactive zero-knowledge proof system *for a language L and security parameter k consists of a setup algorithm G, a pair of PPT algorithms P and V, and a PPT simulator S such that the following three properties are satisfied:*

1. *Completeness: for all $x \in L$,*

$$Pr[\sigma \leftarrow G(1^k);\ \pi \leftarrow P(x, \sigma, k);\ V(x, \sigma, \pi) = accept] = 1.$$

2. *Soundness: for any $x \notin L$ and possibly cheating prover P^*, there exists a negligible function $\nu(\cdot)$ such that*

$$Pr[\sigma \leftarrow G(1^k);\ \pi \leftarrow P^*(x, \sigma, k);\ V(x, \sigma, \pi) = accept] \leq \nu(k).$$

3. *Zero-knowledge: there exists a PPT simulator S that outputs a distribution space such that the ensembles $\{S(x,k)\}_{x \in L}$ and $\{(\sigma, P(x, \sigma, k))\}_{x \in L}$ are indistinguishable.*

The proof in the definition above is in the *common reference string* (CRS) model, which means that a common reference string σ is generated by G and given to both the prover and the verifier. For some NIZK proof systems [BdSMP91, FLS90], it is enough for G to simply output a random string (in which case we call it the *shared random string* model), although in our case G will always output the description of a bilinear group.

While the definition above is a useful formulation of NIZK proofs, it has the problem that only one proof can be generated for each CRS σ. In terms of efficiency, it would be very useful to be able to prove multiple statements using the same σ. This formulation of

NIZK is called *multi-theorem* NIZK and was first introduced by Blum et al. [BdSMP91], who demonstrated a multi-theorem NIZK proof system for 3-SAT, which because 3-SAT is NP-complete further implies existence for all $L \in$ NP (assuming quadratic residuosity). In addition, Feige et al. [FLS90] demonstrated a multi-prover multi-theorem NIZK proof system, assuming trapdoor permutations. More recently, Groth, Ostrovsky, and Sahai [GOS06] gave a construction for Circuit-SAT that achieved perfect zero-knowledge.

2.6.1 Groth-Sahai non-interactive proofs

As discussed in Chapter 1, the proof systems mentioned above are quite inefficient because of their reliance on a single NP-complete language and the need to reduce any statement we might want to prove to this particular language. To generalize this notion, we are finally ready to discuss NIZK proof systems as defined by Groth and Sahai. First, we let R be some efficiently computable ternary relation. We look at elements in R of the form (gk, x, w) where gk is considered the setup, x is the statement, and w is the witness. We can also consider the language L consisting of statements in R for a fixed gk; because we will always take gk to be the description of a bilinear group this means a language L corresponding to some bilinear group.

The Groth-Sahai proof system consists of four PPT algorithms: a generation (setup) algorithm $\mathcal{G}$[2], a CRS generation algorithm K, a prover P, and a verifier V. The setup algorithm $\mathcal{G}$ will take in the security parameter and output information (gk, sk), where gk is some public information (in our case, the description of a bilinear group) and sk is some secret information (for example, it may be the factorization of the group order if it is composite). The CRS generation algorithm K will take (gk, sk) as input and output a common reference string σ. The prover P will then take (gk, σ, x, w) as input and output a proof π. Finally, the verifier will take input (gk, σ, x, π) and output 1 if the proof verifies and 0 otherwise. We require the following properties:

1. **Perfect completeness:** For all adversaries $\mathcal{A}$ and $(gk, x, w) \in R$,

$$\begin{aligned}\Pr[(gk, sk) \leftarrow \mathcal{G}(1^k);\ \sigma \leftarrow K(gk, sk);\ (x, w) \leftarrow \mathcal{A}(gk, \sigma);\\ \pi \leftarrow P(gk, \sigma, x, w)\ :\ V(gk, \sigma, x, \pi) = 1] = 1.\end{aligned}$$

2. **Perfect soundness:** For all adversaries $\mathcal{A}$ and $x \notin L$,

$$\Pr[(gk, sk) \leftarrow \mathcal{G}(1^k);\ \sigma \leftarrow K(gk,sk);\ (x, \pi) \leftarrow \mathcal{A}(gk, \sigma)\ :\ V(gk, \sigma, x, \pi) = 1] = 0.$$

3. **Perfect $L_{\mathbf{co}}$ soundness:** In some cases, we may want to consider a language L_{co} which depends on gk and σ. We can then provide an alternate definition of soundness

[2]This setup algorithm is not included in the original GS paper outlining pairing product equations, but it will be useful for our purposes and for comparison with quadratic equations.

where the adversary succeeds if it creates a valid proof for $x \in L_{\text{co}}$. Note if we set L_{co} to be the complement of L we end up with the version of soundness above. Formally then, we say that for all adversaries $\mathcal{A}$ and $(x, gk, \sigma) \notin L_{\text{co}}$,

$$\Pr[(x, \pi) \leftarrow \mathcal{A}(gk, \sigma) \ : \ V(gk, \sigma, x, \pi) = 1] = 0.$$

4. **CRS indistinguishability:** We need to use the idea of a simulated CRS in order to define witness indistinguishability and zero-knowledge. We require that for all PPT adversaries $\mathcal{A}$, there exists a negligible function $\nu(\cdot)$ such that

$$\begin{aligned} &|\Pr[(gk, sk) \leftarrow \mathcal{G}(1^k);\ \sigma \leftarrow K(gk, sk) \ : \ \mathcal{A}(gk, \sigma) = 1] \\ -\ &\Pr[(gk, sk) \leftarrow \mathcal{G}(1^k);\ \sigma \leftarrow S(gk, sk) \ : \ \mathcal{A}(gk, \sigma) = 1]| = \nu(k), \end{aligned}$$

   in other words that an adversary cannot distinguish between a real and and simulated CRS.

5. **Perfect witness indistinguishability:** We require that on a simulated CRS the witnesses of the prover are perfectly indistinguishable; in experiment notation we write this as

$$\begin{aligned} &\Pr[(gk, sk) \leftarrow \mathcal{G}(1^k);\ \sigma \leftarrow S(gk, sk);\ (x, w_0, w_1, s) \leftarrow \mathcal{A}(gk, \sigma); \\ &\qquad\qquad \pi \leftarrow P(gk, \sigma, x, w_0) \ : \ \mathcal{A}(\pi, s) = 1] \\ =\ &\Pr[(gk, sk) \leftarrow \mathcal{G}(1^k);\ \sigma \leftarrow S(gk, sk);\ (x, w_0, w_1, s) \leftarrow \mathcal{A}(gk, \sigma); \\ &\qquad\qquad \pi \leftarrow P(gk, \sigma, x, w_1) \ : \ \mathcal{A}(\pi, s) = 1] \end{aligned}$$

   for state information s and $(gk, x, w_0), (gk, x, w_1) \in R$.

6. **Perfect zero-knowledge:** Finally, we must consider the notion of composable zero-knowledge. We again require that the adversary cannot distinguish between a real and a simulated CRS. We now look at two simulators: S_1 to produce the simulation string and S_2 to produce the proof π. We also allow for S_1 to output some secret information td and require that even with access to td our adversary *still* cannot distinguish between the outputs of the real prover P and the simulator S_2.

   Formally, we write

$$\begin{aligned} &\Pr[(gk, sk) \leftarrow \mathcal{G}(1^k);(\sigma, td) \leftarrow S_1(gk, sk); \\ &\qquad\qquad (x, w, s) \leftarrow \mathcal{A}(gk, \sigma, td); \pi \leftarrow P(gk, \sigma, x, w) \ : \ \mathcal{A}(\pi, s) = 1] \\ =\ &\Pr[(gk, sk) \leftarrow \mathcal{G}(1^k);(\sigma, td) \leftarrow S_1(gk, sk); \\ &\qquad\qquad (x, w, s) \leftarrow \mathcal{A}(gk, \sigma, td); \pi \leftarrow S_2(gk, \sigma, td, x) \ : \ \mathcal{A}(\pi, s) = 1]. \end{aligned}$$

In addition, we will need to define the idea of a bilinear map (also called a bilinear pairing). Such maps have received a lot of attention in recent years [DBS04] and serve as the basis for a branch of cryptography called *pairing-based cryptography.*

Definition 2.6.2. *A function $e : G_1 \times G_2 \to G_T$, where G_1, G_2, and G_T are all cyclic, is called* a cryptographic bilinear map *if it satisfies the following three properties:*

1. *Computability: e is efficiently computable.*
2. *Bilinearity: for all $a \in G_1$, $b \in G_2$, $x, y \in \mathbb{Z}$, $e(a^x, b^y) = e(a, b)^{xy}$.*
3. *Non-degeneracy: if a is a generator of G_1 and b is a generator of G_2 then $e(a, b)$ is a generator of G_T.*

Because the groups are all cyclic, one consequence of the second property is that our map is bilinear in the typical mathematical sense, so that

$$e(uu', v) = e(u, v)e(u', v) \quad \text{and} \quad e(u, vv') = e(u, v)e(u, v')$$

for $u, u' \in G_1$ and $v, v' \in G_2$. In addition to these three properties, we will see in Chapter 4 that there are sometimes extra properties we will need to require of our bilinear maps.

Chapter 3

GS Proofs

As discussed in Chapter 1, the Groth-Sahai proofs are highly flexible and can be used for proving satisfiability of various kinds of equations under various cryptographic assumptions. Here we highlight a few of them that will be the most useful for our purposes. In Section 3.2 we outline how to prove satisfiability of quadratic equations, and in Section 3.3 we outline how to prove satisfiability of pairing product equations. In Section 3.5 we talk briefly about the instantiation of GS proofs under the Subgroup Decision assumption; while this instantiation is not necessarily useful for our applications, it is still interesting and demonstrates how GS proofs can be used even in composite-order groups. In Sections 3.6 and 3.7, we give detailed outlines of the instantiation of GS proofs under the SXDH and DLIN assumptions, both for quadratic and pairing product equations. Finally, in Chapter 4 we outline applications of GS proofs, including a new way to non-interactively prove knowledge of an exponent and a new construction of non-interactive anonymous credentials using the Boneh-Boyen signature scheme.

3.1 Notation disclaimer

One confusing part of the sections that will follow is the difference in notation. In both the quadratic equations and pairing product equations we will consider a bilinear map $f : G_1 \times G_2 \to G_T$. In the quadratic equations, it will be much more convenient to think of G_1 and G_2 as additive groups (as it corresponds more closely with the notation used in traditional elliptic curve literature), but to think of G_T as a multiplicative group. In the pairing product equations, however, we will use the usual multiplicative notation for all groups. Despite these differences, it is important to note that the commitment schemes used in both are in fact *identical* and it is only the proofs that are fundamentally different.

3.2 Quadratic equations

3.2.1 Introduction and definition

Definition 3.2.1. *Let $(\mathcal{R}, +, \cdot, 0, 1)$ be a finite commutative ring. Then an $\mathcal{R}$-*module A *is an abelian group $(A, +, 0)$ such that there exists an operator (scalar multiplication) $\mathcal{R} \times A \to A$ that maps (r, x) for $r \in \mathcal{R}$ and $x \in A$ to $rx \in A$. We also satisfy the following properties for all $r, s \in \mathcal{R}$, $x, y \in A$:*

- $(r + s)x = rx + sx$.
- $r(x + y) = rx + ry$.
- $r(sx) = (rs)x$.
- $1x = x$.

As an example, any cyclic group of order n can be viewed as a $\mathbb{Z}_n$-module. We will be considering $\mathcal{R}$-modules A_1, A_2, and A_T that have some associated bilinear map $f : A_1 \times A_2 \to A_T$. Using this map, we will focus on sets of quadratic equations of the form

$$\sum_{j=1}^{n} f(a_j, y_j) + \sum_{i=1}^{m} f(x_i, b_i) + \sum_{i=1}^{m}\sum_{j=1}^{n} \gamma_{ij} f(x_i, y_j) = t \tag{3.1}$$

where the x_i and b_i are drawn from A_1, the y_j and a_j are drawn from A_2, t is drawn from A_T, and the γ_{ij} are drawn from $\mathcal{R}$. To simplify this equation, we can use vector notation and define

$$\vec{x} \cdot \vec{y} = \sum_{i=1}^{n} f(x_i, y_i).$$

This allows us to rewrite Equation (3.1) as

$$\vec{a} \cdot \vec{y} + \vec{x} \cdot \vec{b} + \vec{x} \cdot \Gamma\vec{y} = t. \tag{3.2}$$

Note that Γ is a $m \times n$ matrix in $\mathcal{R}$, and that by the bilinear properties of f we have that $\vec{x} \cdot \Gamma\vec{y} = \Gamma^T\vec{x} \cdot \vec{y}$.

3.2.2 Commitment

To commit to elements from an $\mathcal{R}$-module A we define two homomorphisms $\tau : A \to B$ and $\rho : B \to A$. We will use n elements u_i drawn from B for the commitment, and require that $\rho(u_i) = 0$ for all i and that $\rho \circ \tau$ is the identity for all elements such that $\tau(x)$ is not in

$U = \langle u_1, \ldots, u_n \rangle$[1]. To commit to some $x \in A$, we first pick n random values r_i from $\mathcal{R}$ and then compute

$$c(x) = \tau(x) + \sum_{i=1}^{n} r_i u_i.$$

We can use the same vector notation as before to write $\vec{c} = \tau(\vec{x}) + R\vec{u}$ when committing to multiple values.

The key for our commitment will be $(B, \tau, \rho, u_1, \ldots, u_n)$. There are two cases:

- Hiding keys: in this case, the u_i generate the whole module B, which means that $U = \langle u_1, \ldots, u_n \rangle = B$. This implies that $\tau(A) \subseteq U$, which means that $c(x)$ is perfectly hiding.

- Binding keys: in this case, $U \neq B$. This means that c reveals some non-trivial information about x; in particular, it reveals the coset of U where $\tau(x)$ lives. This means that in order for our commitment to be binding, we must restrict the space of x to be the set of all x such that $\tau(x)$ is in the quotient group B/U; because $U \neq B$ both B/U and its inverse image $\tau^{-1}(B/U)$ are non-trivial. Note that for the quotient group to be well-defined U must be a normal submodule of B; this is fine since modules are by definition abelian, so every submodule is normal. One final thing to add to this argument is that, while it might not seem to be a very strong binding property, we often work in groups where each coset has a unique "representative element" and so our restriction of $\tau(x)$ to the space B/U really is binding. This will hopefully become more clear when we discuss instantiations later on.

  We can also note that because we are restricting our choices of x to be in the inverse image $\tau^{-1}(B/U)$, we have that $\rho(\vec{c}) = \rho(\tau(\vec{x}) + R\vec{u}) = \vec{x}$.

It is clear that we cannot have both these cases at the same time, as one implies that $\rho \circ \tau$ is the zero map for all $x \in A$ and the other one requires that it is the identity for elements $x \in \tau^{-1}(B/U)$. From Property 4 in Section 2.6, however, we know that the settings in which these keys are used need to be indistinguishable in order to get any witness indistinguishability or zero knowledge properties. This means that we will need to use an assumption that implies that these two cases are indistinguishable.

3.2.3 CRS setup

The common reference string (CRS) contains commitment keys that allow us to commit to elements in A_1 and A_2 (so these commitment keys will define B_1 and B_2, as well as our maps τ_1, τ_2, ρ_1, ρ_2, and some elements $u_1, \ldots, u_m$ and $v_1, \ldots, v_n$). The CRS must also

[1]Note that here we deviate from Groth and Sahai – they only require that $\rho \circ \tau$ is not the zero map.

$$\begin{tikzcd}
A_1 \arrow[d, shift right, "\tau_1"'] & \times & A_2 \arrow[r, "f"'] \arrow[d, shift right, "\tau_2"'] & A_T \arrow[d, shift right, "\tau_T"'] \\
B_1 \arrow[u, shift right, "p_1"'] & \times & B_2 \arrow[r, "F"'] \arrow[u, shift right, "p_2"'] & B_T \arrow[u, shift right, "p_T"']
\end{tikzcd}$$

Figure 3.1: Commutative Diagram for Groth-Sahai Proofs

define B_T and the maps τ_T, ρ_T, and F. We can extend our vector notation here as well to say that

$$\vec{X} * \vec{Y} = \sum_{i=1}^{n} F(X, Y).$$

for $\vec{X} \in B_1^n$ and $\vec{Y} \in B_2^n$. Finally, our CRS must define the set of matrices $H_1, \ldots, H_k$ that are a basis for the $\mathcal{R}$-module of all matrices H that satisfy $\vec{u} * H\vec{v} = 0$.

In general, we work with two different settings (note $U = \langle u_1, \ldots, u_m \rangle$ and $V = \langle v_1, \ldots, v_n \rangle$):

- Binding setting: the commitment keys are binding for $x_1 \in \tau_1^{-1}(B_1/U)$ and $x_2 \in \tau_2^{-1}(B_2/V)$, which means that $\rho_1 \circ \tau_1(x_1) = x_1$ and $\rho_2 \circ \tau_2(x_2) = x_2$ for all x_1, x_2 values to which we will be forming commitments. We will use this setting to prove soundness and extractability.

- Hiding setting: the commitment keys are hiding, which means $B_1 = U$ and similarly $B_2 = V$. This implies that $\rho_1 \circ \tau_1(x_1) = 0$ and $\rho_2 \circ \tau_2(x_2) = 0$ for all $x_1 \in A_1$ and $x_2 \in A_2$. We will use this setting to prove witness indistinguishability and zero-knowledge.

As with our commitment cases, we will need these two settings to be indistinguishable.

3.2.4 Proving committed values satisfy a quadratic equation

Remember that a quadratic equation is of the form

$$\vec{a} \cdot \vec{y} + \vec{x} \cdot \vec{b} + \vec{x} \cdot \Gamma \vec{y} = t$$

where $\vec{a} \in A_1^n$, $\vec{b} \in A_2^m$, $\Gamma \in \mathcal{R}_{m \times n}$ and $t \in A_T$ are constants and $\vec{x} \in A_1^m$, $\vec{y} \in A_2^n$ are the variables that satisfy the equation. We would like to prove that we know $\vec{c} = Com(\vec{x})$ and $\vec{d} = Com(\vec{y})$; in other words commitments whose openings satisfy the equation.

Suppose that to prove this we compute values $(\vec{\pi}, \vec{\psi})$ that satisfy the following equation:

$$\tau_1(\vec{a}) * \vec{d} + \vec{c} * \tau_2(\vec{b}) + \vec{c} * \Gamma\vec{d} = \tau_T(t) + \vec{u} * \vec{\pi}' + \vec{\psi}' * \vec{v}. \tag{3.3}$$

Why would such a proof be sound? To see this, remember that in the binding setting there exist maps ρ_1, ρ_2, and ρ_T such that $\rho_1(\vec{u}) = 0$ and $\rho_2(\vec{v}) = 0$. If we then apply these maps to the above equation, we end up with

$$a * \rho_2(\vec{d}) + \rho_1(\vec{c}) * \vec{b} + \rho_1(\vec{c}) * \Gamma\rho_2(\vec{d}) = \rho_T(\tau_T(t)).$$

Because the binding setting further requires that $\rho_1 \circ \tau_1$ is the identity, it will also be the case that $\rho_1(\vec{c}) = \rho_1(\tau_1(\vec{x})) + \rho_1(R\vec{u}) = \vec{x}$. Similarly, $\rho_2(\vec{d}) = \vec{y}$, and $\rho_T(\tau_T(t)) = t$. Since the witnesses are completely uncovered, this gives us perfect soundness.

Now that we have an idea of soundness, we of course need to figure out how to actually form this proof $(\vec{\pi}, \vec{\psi})$. One naive approach to forming a NIWI proof is to just plug commitments to $\vec{x}$ and $\vec{y}$ into the original equation. This turns out to be almost good enough. Remember that the commitments are of the form $\vec{c} = \tau_1(\vec{x}) + R\vec{u}$ and $\vec{d} = \tau_2(\vec{y}) + S\vec{v}$. Because $\vec{c} \in B_1^m$ and $\vec{d} \in B_2^n$ we need to map $\vec{a}$ into B_1^n and $\vec{b}$ into B_2^m in order to pair them with $\vec{d}$ and $\vec{c}$ respectively, which means that we end up with the left-hand side of Equation 3.3. Plugging in the values for our commitments and expanding them out, we get

$$\begin{aligned}
\text{(LHS of 3.3)} &= \tau_1(\vec{a}) * (\tau_2(\vec{y}) + S\vec{v}) + (\tau_1(\vec{x}) + R\vec{u}) * \tau_2(\vec{b}) + (\tau_1(\vec{x}) + R\vec{u}) * \Gamma(\tau_2(\vec{y}) + S\vec{v}) \\
&= \tau_1(\vec{a}) * \tau_2(\vec{y}) + \tau_1(\vec{a}) * S\vec{v} + \tau_1(\vec{x}) * \tau_2(\vec{b}) + R\vec{u} * \tau_2(\vec{b}) + \tau_1(\vec{x}) * \Gamma\tau_2(\vec{y}) \\
&\quad + R\vec{u} * \Gamma\tau_2(\vec{y}) + \tau_1(\vec{x}) * \Gamma S\vec{v} + R\vec{u} * \Gamma S\vec{v} \\
&= \tau_1(\vec{a}) * \tau_2(\vec{y}) + \tau_1(\vec{x}) * \tau_2(\vec{b}) + \tau_1(\vec{x}) * \Gamma\tau_2(\vec{y}) \\
&\quad + R\vec{u} * \tau_2(\vec{b}) + R\vec{u} * \Gamma\tau_2(\vec{y}) + R\vec{u} * \Gamma S\vec{v} \\
&\quad + \tau_1(\vec{a}) * S\vec{v} + \tau_1(\vec{x}) * \Gamma S\vec{v} \\
&= \tau_T(t) + \vec{u} * R^\top\tau_2(\vec{b}) + \vec{u} * R^\top\Gamma\tau_2(\vec{y}) + \vec{u} * R^\top\Gamma S\vec{v} \\
&\quad + S^\top\tau_1(\vec{a}) * \vec{v} + S^\top\Gamma^\top\tau_1(\vec{x}) * \vec{v} \\
&= \tau_T(t) + \vec{u} * (\underbrace{R^\top\tau_2(\vec{b}) + R^\top\Gamma\tau_2(\vec{y}) + R^\top\Gamma S\vec{v}}_{\vec{\pi}}) + (\underbrace{S^\top\tau_1(\vec{a}) + S^\top\Gamma^\top\tau_1(\vec{x})}_{\vec{\psi}}) * \vec{v}
\end{aligned}$$

Now if we form our proof $(\vec{\pi}', \vec{\psi}')$ as defined above we have that (LHS of 3.3) $= \tau_T(t) + \vec{u} * \vec{\pi}' + \vec{\psi}' * \vec{v}$, which is what we wanted for soundness. Unfortunately, this is not quite enough for witness indistinguishability.

Recall that to prove witness indistinguishability we work in the hiding setting, where the commitments are perfectly hiding. Consider as an example the symmetric case where $A_1 = A_2$ and $\vec{u} = \vec{v}$. Then $\vec{\pi}'$ is the only thing that could reveal any information about either $\vec{x}$ or $\vec{y}$. If $\vec{\pi}'$ is unique then we trivially have WI, so we only need to consider the case where we have two values $\vec{\pi}'_1$ and $\vec{\pi}'_2$ that satisfy the equation. This means that $\vec{u} * (\vec{\pi}'_1 - \vec{\pi}'_2) = 0$. To guarantee the same distribution regardless of our value, we can add $\sum_{i=1}^{k} r_i H_i \vec{v}$ to $\vec{\pi}'$ (remember that by the nature of the H_i when this is multiplied on the left by $\vec{u}$ we will get 0). This means that our new $\vec{\pi}'$ will satisfy the verification equation but also have the same distribution no matter what "real $\vec{\pi}'$" is contained within.

In the non-symmetric case, we still have $\vec{\psi}'$ to take care of. We can pick a matrix T at random to randomize $\vec{\psi}'$ so that we end up with $\vec{\psi}' + T\vec{u}$. We will therefore need to balance this term out in $\vec{\pi}'$ (so that it does not affect our series of equations from above), so we get

$$(\text{LHS of } 3.3) = \tau_T(t) + \vec{u} * \vec{\pi} + \vec{\psi} * \vec{v} = \tau_T(t) + \vec{u} * (\vec{\pi}' + \sum_{i=1}^{k} r_i H_i \vec{v} - T^\top \vec{v}) + (\vec{\psi}' + T\vec{u}) * \vec{v}.$$

This means that our protocol proceeds as follows:

Prover: Pick T and $r_1, \dots, r_k$ at random. Compute

$$\vec{\pi} = R^\top \tau_2(\vec{b}) + R^\top \Gamma \tau_2(\vec{y}) + R^\top \Gamma S \vec{v} - T^\top \vec{v} + \sum_{i=1}^{k} r_i H_i v$$
$$\vec{\psi} = R^\top \tau_1(\vec{a}) + S^\top \Gamma^\top \tau_1(\vec{x}) + T\vec{u}$$

and return $(\vec{\pi}, \vec{\psi})$.

Verifier: Accept if and only if

$$\tau_1(\vec{a}) * \vec{d} + \vec{c} * \tau_2(\vec{b}) + \vec{c} * \Gamma \vec{d} = \tau_T(t) + \vec{u} * \vec{\pi} + \vec{\psi} * \vec{v}.$$

Our construction above shows that we have already satisfied completeness, which means that we now only need to prove soundness and witness indistinguishability. For soundness, we recall from our discussion above that the idea is to apply the inverse map ρ for every τ. Because $\rho \circ \tau$ is the identity map we end up with perfect soundness.

Theorem 3.2.2. *In the witness indistinguishability setting defined in Section 3.2.3, all satisfying witnesses $\vec{x}$, $\vec{y}$, R, S yield proofs $\vec{\pi} \in \langle v_1, \dots, v_n \rangle^m$ and $\vec{\psi} \in \langle u_1, \dots, u_m \rangle^n$ that are uniformly distributed (conditioned on the satisfication of the verification equation).*

Proof. We know that $\tau_1(A_1) \subseteq \langle u_1, \dots, u_m \rangle$ and similarly that $\tau_2(A_2) \subseteq \langle v_1, \dots, v_n \rangle$. Then we know there exist matrices A, B, X, and Y over $\mathcal{R}$ such that $\tau_1(\vec{a}) = A\vec{u}$, $\tau_1(\vec{x}) = X\vec{u}$, $\tau_2(\vec{b}) = B\vec{v}$, and $\tau_2(\vec{y}) = Y\vec{v}$. This means we can rewrite our commitments as $\vec{c} = (X + R)\vec{u}$

and $\vec{d} = (Y + S)\vec{v}$. We can also rewrite our proofs $\vec{\pi}$ and $\vec{\psi}$ as

$$\vec{\psi} = (S^\top A + S^\top \Gamma^\top X + T)\vec{u} \quad \text{and}$$

$$\vec{\pi} = (R^\top B + R^\top \Gamma Y + R^\top \Gamma S - T^\top)\vec{v} + (\sum_{i=1}^{k} r_i H_i)\vec{v}.$$

Since T was chosen at random, we can think of $\vec{\psi}$ as a random variable given by $\vec{\psi} = \Psi\vec{v}$ for a random matrix Ψ. Similarly, we can think of $\vec{\pi} = \Pi\vec{v}$ where Π depends on Ψ.

We know we can write any two choices of Π as $\Pi_1 = \Pi_2 + \sum_{i=1}^{k} r_i H_i$. Based on how we form $\vec{\pi}$, we know that we get a uniform distribution over $\vec{\pi}$ if we condition on $\vec{\psi}$. Finally, because $\vec{\psi}$ is uniformly chosen, we know that we get a uniform distribution over $\vec{\pi}, \vec{\psi}$ for any witness. $\square$

3.2.5 NIWI proof

To summarize all that we have done above, we outline a fully composable NIWI proof for satisfiability of a set of quadratic equations.

- **Setup:** $(gk, sk) = ((\mathcal{R}, A_1, A_2, A_T, f), sk) \leftarrow \mathcal{G}(1^k)$.
- **Binding:** $\sigma = (B_1, B_2, B_T, F, \tau_1, \rho_1, \tau_2, \rho_2, \tau_T, \rho_T, \vec{u}, \vec{v}) \leftarrow K(gk, sk)$, where $\vec{u}$ and $\vec{v}$ are commitment keys for binding commitment schemes.
- **Hiding:** $\sigma = (B_1, B_2, B_T, F, \tau_1, \rho_1, \tau_2, \rho_2, \tau_T, \rho_T, \vec{u}, \vec{v}) \leftarrow S(gk, sk)$, where $\vec{u}$ and $\vec{v}$ are commitment keys for hiding commitment schemes.
- **Prover:** On input gk, σ, a set of quadratic equations $\{(\vec{a_i}, \vec{b_i}, \Gamma_i, t_i)\}_{i=1}^{N}$ and a satisfying witness $\vec{x}, \vec{y}$, pick R and S at random and then form the commitments $\vec{c} = \tau_1(\vec{x}) + R\vec{u}$ and $\vec{d} = \tau_2(\vec{y}) + S\vec{v}$. Now, for each quadratic equation pick T_i and $r_{i1}, \ldots, r_{ik}$ at random and compute

$$\vec{\pi}_i = R^\top \tau_2(\vec{b_i}) + R^\top \Gamma \tau_2(\vec{y}) + R^\top \Gamma S \vec{v} - T_i^\top \vec{v} + \sum_{j=1}^{k} r_{ij} H_j \vec{v} \tag{3.4}$$

$$\vec{\psi}_i = S^\top \tau_1(\vec{a_i}) + S^\top \Gamma^\top \tau_1(\vec{x}) + T_i \vec{u}. \tag{3.5}$$

  Output the proof $(\vec{c}, \vec{d}, \{\vec{\pi}_i, \vec{\psi}_i\}_{i=1}^{N})$.

- **Verifier:** On input gk, σ, a set of quadratic equations, and a proof $\vec{c}$, $\vec{d}$, $\{\vec{\pi}_i, \vec{\psi}_i\}_{i=1}^{N}$, check for each equation that

$$\tau_1(\vec{a_i}) * \vec{d} + \vec{c} * \tau_2(\vec{b_i}) + \vec{c} * \Gamma_i \vec{d} = \tau_T(t_i) + \vec{u} * \vec{\pi}_i + \vec{\psi}_i * \vec{v}.$$

  Accept the proof if and only all these checks pass.

In addition, our discussions in the previous section have proved the following theorem:

Theorem 3.2.3. *The protocol outlined above is a non-interactive witness-instinguishable proof of satisfiability of a set of quadratic equations with perfect completeness, perfect soundness, and perfect composable witness-indistinguishability.*

3.2.6 The symmetric setting

One interesting case we can consider is when $A_1 = A_2$, our commitment keys $\vec{u}$ and $\vec{v}$ are the same, and our bilinear map F is *symmetric*, i.e. $F(x, y) = F(y, x)$ for all x, y. This case arises when we look at the DLIN assumption in Section 3.7 and it turns that we can improve efficiency by having our proof consist of one value $\vec{\phi}$ instead of both $\vec{\pi}$ and $\vec{\psi}$. To see this, note the verification equation will collapse to

$$\begin{aligned} \tau_1(\vec{a}) * \vec{d} + \vec{c} * \tau_2(\vec{b}) + \vec{c} * \Gamma\vec{d} &= \tau_T(t) + \vec{u} * \vec{\pi} + \vec{\psi} * \vec{u} \\ &= \tau_T(t) + \vec{u} * (\vec{\pi} + \vec{\psi}) \end{aligned}$$

since F is symmetric. This means that we end up with a proof of the form

$$\vec{\phi} = R^\top\tau_2(\vec{b}) + R^\top\Gamma\tau_2(\vec{y}) + R^\top\Gamma S\vec{u} - T^\top\vec{u} + \sum_{i=1}^{k} r_i H_i\vec{u} + S^\top\tau_1(\vec{a}) + S^\top\Gamma^\top\tau_1(\vec{x}) + T\vec{u}.$$

This is certainly a little messy, but it is possible to simplify it a little by noting that the symmetry of F implies $\vec{u} * (T - T^\top)\vec{u} = 0$, so that these terms will be cancelled when the proof is plugged into the verification equation. This means that we can use

$$\vec{\phi} = R^\top\tau_2(\vec{b}) + R^\top\Gamma\tau_2(\vec{y}) + R^\top\Gamma S\vec{u} + S^\top\tau_1(\vec{a}) + S^\top\Gamma^\top\tau_1(\vec{x}) + \sum_{i=1}^{k} r_i H_i\vec{u}. \tag{3.6}$$

3.2.7 Zero knowledge

It turns out that in some cases it not so hard to use the above NIWI proof system to create a proof system that satisfies the zero-knowledge property from Section 2.6.1. This formulation of zero-knowledge is slightly non-standard; we are not proving that values *in a set of commitments* satisfy the equations, but that a set of values satisfy the equations (so no commitments involved). This means the prover is allowed to form his own commitments, which means that the simulator in turn is allowed to form its own commitments. We address a stronger version of zero-knowledge in Chapter 4 in which the simulator is given a commitment that it does not necessarily know how to open, but for now we focus on this weaker version.

First, we consider the case of prime-order groups, which means that the job of the simulator is made much easier. Given the equations $\{\vec{a_i}, \vec{b_i}, \Gamma_i, t_i\}_{i=1}^N$ and witnesses $\vec{x}$ and $\vec{y}$, the prover proceeds as normal (so forms the commitments and proofs just as in Section 3.2.5). The simulator, on the other hand, does not know the witnesses. For each individual equation, however, the simulator is able to simply solve for $\vec{x}$ and $\vec{y}$. The simulator can then form a pair of commitments $\vec{c}$ and $\vec{d}$ to ones of these "witnesses" $\vec{x}$ and $\vec{y}$; these are not witnesses in the real sense as they do not satisfy all the equations simultaneously. This means that in order to allow the simulator to form the proofs, we need it to have access to a trapdoor td that allows it to open $\vec{c}$ and $\vec{d}$ to any $\vec{x}$ and $\vec{y}$ – in particular this means that the simulator can use $\vec{c}$ and $\vec{d}$ as commitments to each of the pairs $\vec{x}$ and $\vec{y}$ that it found for an individual equation. In other words, with this trapdoor the simulator can compute the proofs $\{\vec{\pi_i}, \vec{\psi_i}\}_{i=1}^N$ exactly as the prover would, using the same commitments $\vec{c}$ and $\vec{d}$ but different values of $\vec{x}$ and $\vec{y}$ for each equation.

Theorem 3.2.4. *The protocol described in Section 3.2.5 is a composable non-interactive zero-knowledge proof of satisfiability of a set of quadratic equations with perfect completeness, perfect soundness, and perfect composable zero-knowledge.*

Proof. First, the perfect completeness and perfect soundness properties follow directly from Theorem 3.2.3, as the protocol has not been altered in any way.

Why are these proofs also zero-knowledge? We use the simulator described above, which gets the set of equations and computes a satisfying witness $\vec{x}$ and $\vec{y}$ for each individual equation, and then uses commitments $\vec{c}$ and $\vec{d}$ to form proofs for these witnesses. Consider a hybrid distribution Hyb_i in which the commitments $\vec{c}$ and $\vec{d}$ are still sent, but the real witnesses are used for the first i equations and the simulator witnesses are used for the last $N-i$ equations (we can consider this distribution since we do not need to bound our hybrid's computational power; this means the commitments c_i and d_i can be opened to the simulator's x_i and y_i using brute force). To show that Hyb_i and Hyb_{i+1} are indistinguishable, we recall that the commitments are perfectly hiding and so neither c_i nor d_i will reveal any information about the witness contained within. As for the proofs, since each individual proof formed by the simulator is a valid proof (since the simulator knows a valid witness for that particular equation), Theorem 3.2.3 tells us that the distribution of $\vec{\pi_{i+1}}, \vec{\psi_{i+1}}$ will be identical for the prover and the simulator, and so the hybrid distributions are in fact perfectly indistinguishable. □

In a $\mathbb{Z}_n$-module G we cannot just solve for each equation, since a given element will not necessarily have an inverse. Therefore, the case of a composite order group is more work. In fact, there is currently no known method for simulating a proof for a generic quadratic equation, which means there is no known way to achieve this version of zero-knowledge. If our equations are of the form $(a, b, \Gamma, 1)$, however, then $\vec{x} = \vec{0}$ and $\vec{y} = \vec{0}$ are always witnesses (recall that we are using $t = 1$ because although A_1 and A_2 are considering additively, we use multiplicative notation for A_T and therefore 1 and not 0 represents the identity). Therefore, we can trivially guarantee zero knowledge in the case where $t = 1$.

3.2.8 Proof of knowledge

One final thing to note is that these proofs are also proofs of knowledge. As we will see in Sections 3.6 and 3.7, it is often possible for the CRS generator K to output a trapdoor that makes it possible to compute the ρ maps. This means that an extractor given this trapdoor is able to compute $\rho_1(\tau_1(\vec{x})) = \vec{x}$ and $\rho_2(\tau_2(\vec{y})) = \vec{y}$ and thus fully recover both the witnesses, making the proofs perfect proofs of knowledge. It turns out, however, that the ρ maps are only efficiently computable for elements $X \in A$. We also consider commitments to exponents $x \in \mathcal{R}$. For these elements, computing ρ is infeasible (even with a trapdoor) if we assume the hardness of the Discrete Log Assumption. Therefore, we won't be able to see proofs of knowledge of exponents until Chapter 4.

3.3 Pairing product equations

Here we will be using multiplicative notation instead of additive notation. This means that an $\mathcal{R}$-module will now be defined as an abelian group $(A, \cdot, 1)$ such that for $r, s \in \mathcal{R}$ and $u, v \in A$ we have $u^{r+s} = u^r u^s$ and $(uv)^r = u^r v^r$.

3.3.1 Introduction and definition

In general, we are interested in proving a pairing product equation of the form

$$\prod_{q=1}^{N} f(a_q \prod_{i=1}^{m} x_i^{\alpha_{qi}}, b_q \prod_{j=1}^{n} y_j^{\beta_{qj}}) = t \tag{3.7}$$

where $a_q \in A_1$, $b_q \in A_2$, $\alpha_{qi}, \beta_{qj} \in \mathcal{R}$, and $t \in A_T$ are the equation constants and $x_i \in A_1$, $y_j \in A_2$ are the variables we are trying to prove satisfy the equation. As with quadratic equations, we will do this using commitments to the variables (which will be outlined in the next section). This means that for $r_{ik}, s_{jl} \leftarrow \mathcal{R}$ we have committed to the values x_i and y_j as

$$c_i = \tau_1(x_i) \prod_{k=1}^{m} u_k^{r_{ik}} \quad \text{and} \quad d_j = \tau_2(y_j) \prod_{l=1}^{n} v_l^{s_{jl}}.$$

Because our commitments are homomorphic, we have that

$$\begin{aligned} a_q \prod_{i=1}^{N} c_i^{\alpha_{qi}} &= a_q \prod_{i=1}^{N} (\tau_1(x_i) \prod_{k=1}^{m} u_k^{r_{ik}})^{\alpha_{qi}} \\ &= a_q \prod_{i=1}^{N} \tau_1(x_i)^{\alpha_{qi}} \cdot \prod_{k=1}^{m} u_k^{\sum_{i=1}^{I} \alpha_{qi} r_{ki}}. \end{aligned}$$

In particular, this means that anyone can compute a commitment to $a_q \prod_i \tau_1(x_i)^{\alpha_{qi}}$ and that, by the same reasoning, anyone can compute a commitment to $b_q \prod_j \tau(y_j)^{\beta_{qj}}$. Looking back at Equation (3.7), this means that anyone can compute commitments to the values on the left-hand side and so they can be treated as single module elements. Putting all this together, we see that this general equation can be reduced to a simplified version of the form

$$\prod_{q=1}^{N} f(x_q, y_q) = t \tag{3.8}$$

If we use vector notation similar to that in Section 3.2.1, we can define

$$\vec{x} \bullet \vec{y} = \prod_{q=1}^{N} f(x_q, y_q)$$

to rewrite Equation (3.8) as

$$\vec{x} \bullet \vec{y} = t.$$

3.3.2 Commitment

We will here require the same homomorphisms as for the quadratic equations, namely maps $\tau : A \to B$ and $\rho : B \to A$. To commit to an element $x \in A$ we can let $u_1, \ldots, u_n$ be elements in B. We choose random $r_1, \ldots, r_n \in \mathcal{R}$ and form the commitment as

$$c = \tau(x) \prod_{i=1}^{n} u_i^{r_i}.$$

We require the same properties of our maps τ and ρ. Recall from Section 3.2.2 that when $U = B$ we get perfect hiding because $\rho(u_i) = 0$ for all i, and when $U \neq B$ we get binding for all $x \in \tau^{-1}(B/U)$ because $\rho \circ \tau$ is the identity for such values x.

These τ and ρ maps correspond to our notation used in the quadratic equations section, but not to the original Groth-Sahai notation. The maps will hopefully serve to highlight the similarities between the quadratic equations and pairing product equations settings, as well as allow us to more easily integrate our discussion of commitment to group elements and commitment to exponents.

3.3.3 CRS setup

Let A_1, A_2, A_T, and B_1, B_2, and B_T be $\mathcal{R}$-modules, where the relations between the modules are the same as in the quadratic equations setting and can be found in Figure 3.1.

We let $u_1, \ldots, u_m$ be elements in B_1 and $v_1, \ldots, v_n$ be elements in B_2, and denote $U = \langle u_1, \ldots, u_m \rangle$ and $V = \langle v_1, \ldots, v_n \rangle$. The bilinear map F will produce mn elements (not necessarily distinct) in B_T. This gives rise to a linear map $\mu_{\vec{u},\vec{v}} : \mathcal{R}^{mn} \to B_T$ such that

$$(\alpha_{11}, \ldots, \alpha_{mn}) \mapsto \prod_{i=1}^{m} \prod_{j=1}^{n} F(u_i, v_j)^{\alpha_{ij}}.$$

We know that $(0, \ldots, 0)$ is always in the kernel of μ, but there may be other elements as well which we denote as $h_1, \ldots, h_k$. This means that any vector $(\alpha_{11}, \ldots, \alpha_{mn})$ such that

$$\prod_{i=1}^{m} \prod_{j=1}^{n} F(u_i, v_j)^{\alpha_{ij}} = 1$$

can therefore be written as a linear combination of these h_i (in other words, they form a basis for the kernel). Our CRS will consist of descriptions of the modules and the commitment keys $\vec{u}$ and $\vec{v}$, as well as the bilinear map F and these elements $h_1, \ldots, h_k$ that generate the kernel of $\mu_{\vec{u},\vec{v}}$.

3.3.4 Proving committed values satisfy a pairing product equation

Suppose that we have commitments $\vec{c}$ to $\vec{x}$ in A_1 and commitments $\vec{d}$ to $\vec{y}$ in A_2, so commitments of the form

$$c_q = \tau(x_q) \prod_{i=1}^{m} u_i^{r_{qi}} \quad \text{and} \quad d_q = \tau(y_q) \prod_{j=1}^{n} v_j^{s_{qj}}$$

for random $r_{qi}, s_{qj} \in \mathcal{R}$.

We want to prove that the values contained within our commitments satisfy the simplified pairing product equation in Equation 3.8. Suppose we have a proof $\vec{\pi}, \vec{\psi}$ such that

$$\prod_{q=1}^{N} F(c_q, d_q) = \tau_T(t) \cdot \prod_{i=1}^{m} F(u_i, \pi_i) \prod_{j=1}^{n} F(\psi_j, v_j). \tag{3.9}$$

As with the quadratic equations setting, we first want to consider why such a proof would be sound. To see this, we again extend our vector notation to define

$$\vec{c} * \vec{d} = \prod_{q=1}^{N} F(c_q, d_q)$$

and write Equation 3.9 even more succinctly as

$$\vec{c} * \vec{d} = \tau_T(\vec{t})(\vec{u} * \vec{\pi})(\vec{\psi} * \vec{v}). \tag{3.10}$$

Any proofs that satisfy this equation will imply that

$$\rho_1(\tau_1(\vec{x})) \bullet \rho_2(\tau_2(\vec{y})) = \rho_T(\tau_T(t)),$$

which because $\rho_1 \circ \tau_1$, $\rho_2 \circ \tau_2$, and $\rho_T \circ \tau_T$ are all the identity will further imply that $\vec{x} \bullet \vec{y} = t$, which is what we were originally trying to prove. So we have perfect soundness.

Now, we need to actually construct our proofs. We use the following series of equalities for $t_{ij}, t_l \leftarrow \mathcal{R}$:

$$\begin{aligned}
\prod_{q=1}^{N} F(c_q, d_q) \cdot \tau_T(t^{-1}) &= \prod_{q=1}^{N} F(\tau_1(x_q) \prod_{i=1}^{m} u_i^{r_{qi}}\ ,\ \tau_2(y_q) \prod_{j=1}^{n} v_j^{s_{qj}}) \cdot \tau_T(t^{-1}) \\
&= \prod_{q=1}^{N} F(\tau_1(x_q), \tau_2(y_q)) \cdot \tau_T(t^{-1}) \prod_{q=1}^{N} \prod_{i=1}^{m} F(u_i^{r_{qi}}, \tau_2(y_q) \prod_{j=1}^{n} v_j^{s_{qj}}) \cdot \\
&\quad \prod_{q=1}^{N} \prod_{j=1}^{n} F(\tau_1(x_q), v_j^{s_{qj}}) \\
&= 1 \cdot \prod_{i=1}^{m} F(u_i\ , \prod_{q=1}^{N} d_q^{r_{qi}}) \prod_{j=1}^{n} F(\prod_{q=1}^{N} \tau_1(x_q)^{s_{qj}}, v_j) \\
&= \prod_{i=1}^{m} F(u_i\ , \prod_{j=1}^{n} v_j^{t_{ij}} \prod_{q=1}^{N} d_q^{r_{qi}}) \prod_{j=1}^{n} F(\prod_{i=1}^{m} u_i^{-t_{ij}} \prod_{q=1}^{N} \tau_1(x_q)^{s_{qj}}, v_j) \\
&= \prod_{i=1}^{m} F(u_i\ , \underbrace{\prod_{j=1}^{n} v_j^{t_{ij}} \prod_{q=1}^{N} d_q^{r_{qi}}}_{\pi_i}) \prod_{j=1}^{n} F(\underbrace{\prod_{i=1}^{m} u_i^{-t_{ij} + \sum_{l=1}^{k} t_l h_{lij}} \prod_{q=1}^{N} \tau_1(x_q)^{s_{qj}}}_{\psi_j}, v_j).
\end{aligned}$$

If we then define π_i and ψ_j as indicated above, we get our verification equation from Equation 3.9.

3.3.5 NIWI proof

To summarize all that we have done above, we outline a NIWI proof for satisfiability of a set of pairing product equations.

- **Setup:** $(gk, sk) = ((\mathcal{R}, A_1, A_2, A_T, f), sk) \leftarrow \mathcal{G}(1^k)$.

- **Binding:** $\sigma = (B_1, B_2, B_T, F, \tau_1, \rho_1, \tau_2, \rho_2, \tau_T, \rho_T, \vec{u}, \vec{v}) \leftarrow K(gk, sk)$, where $\vec{u}$ and $\vec{v}$ are commitment keys for binding commitment schemes.
- **Hiding:** $\sigma = (B_1, B_2, B_T, F, \tau_1, \rho_1, \tau_2, \rho_2, \tau_T, \rho_T, \vec{u}, \vec{v}) \leftarrow S(gk, sk)$, where $\vec{u}$ and $\vec{v}$ are commitment keys for hiding commitment schemes.
- **Prover:** As input we get gk, σ, a set of pairing product equations $\{t_i\}_{i=1}^N$ and a satisfying witness $\vec{x}, \vec{y}$. For each x_i form a commitment $c_i = \tau_1(x_i)\prod_j u_j^{r_j}$ for $r_j \leftarrow \mathcal{R}$. Similarly, for each y_i form a commitment $d_i = \tau_2(y_i)\prod_j v_j^{s_j}$ for $s_j \leftarrow \mathcal{R}$. Now, for each pairing product equation pick t_{ij} and t_l at random and compute

$$\vec{\pi}_i = \prod_{j=1}^{n} v_j^{t_{ij}} \prod_{q=1}^{N} d_q^{r_{qi}} \tag{3.11}$$

$$\vec{\psi}_i = \prod_{j=1}^{m} u_j^{-t_{ij}+\sum_{l=1}^{k} t_l h_{ijl}} \prod_{q=1}^{N} \tau_1(x_q)^{s_{qi}}. \tag{3.12}$$

  Output the proof $(\vec{c}, \vec{d}, \{\vec{\pi}_i, \vec{\psi}_i\}_{i=1}^N)$.

- **Verifier:** On input gk, σ, a set of equations, and a proof $\vec{c}$, $\vec{d}$, $\{\vec{\pi}_i, \vec{\psi}_i\}_{i=1}^N$, check for each equation that

$$\vec{c} * \vec{d} = \tau_T(t_i)(\vec{u} * \vec{\pi}_i)(\vec{\psi}_i * \vec{v}).$$

  Accept the proof if and only if all these checks pass.

Theorem 3.3.1. *A proof system constructed as above, where the proof is $\vec{\pi}, \vec{\psi}$ and the verification equation is Equation (3.9), satisfies perfect completeness, perfect soundness, and perfect composable witness indistinguishability.*

Proof. Perfect completeness follows from Equation 3.9 and its derivation, and perfect soundness follows from the discussion at the beginning of the section.

This means that all that is really left to prove is witness indistinguishability; it turns out that the proof of witness indistinguishability is quite similar to that for quadratic equations. We first recall that for witness indistinguishability our commitments are perfectly hiding, which means that $U = B_1$ and $V = B_2$. This means there exists constants X, Y, R and S such that $\tau_1(\vec{x}) = X\vec{u}$ and $\tau_2(\vec{y}) = Y\vec{v}$, which furthermore means we can write our commitments as $\vec{c} = XR\vec{u}$ and $\vec{d} = YS\vec{v}$. Extending this idea to our proofs, we can now write them as

$$\vec{\pi} = TYSR\vec{v} \quad \text{and} \quad \vec{\psi} = T'TXRS\vec{u}.$$

Since T and T' are completely random (as they represent the randomness introduced by the t_{ij} and t_l respectively), we can think of $\vec{\pi}$ and $\vec{\psi}$ as being random variables $\vec{\pi} = \Pi\vec{v}$ and

$\vec{\psi} = \Psi\vec{u}$. Conditioned on the joint satisfication of the verification equation, we can see that we get a uniform distribution over these random variables for any witness, meaning that our proofs are perfectly witness indistinguishable. □

3.3.6 The symmetric setting

One immediate thing we can see is that if we have a setting in which $A_1 = A_2$, we use the same commitment keys $\vec{u}$ and $\vec{v}$, and our bilinear map F is symmetric, we can collapse the verification equation to

$$\begin{aligned}\prod_{q=1}^{N} F(c_q, d_q) \cdot \tau_T(t)^{-1} &= \prod_{i=1}^{m} F(u_i, \pi_i) \prod_{j=1}^{m} F(\psi_j, u_j) \\ &= \prod_{i=1}^{m} F(u_i, \pi_i \psi_i).\end{aligned}$$

This will prove to be more efficient since it means our proof will consist of a single element $\vec{\phi} = \vec{\pi} \cdot \vec{\psi}$. To write this out more fully, we have for each equation a proof of the form

$$\vec{\phi_i} = \prod_{j=1}^{m} u_j^{\sum_{l=1}^{k} t_l h_{lji}} \prod_{q=1}^{N} d_q^{r_{qi}} x_q^{s_{qi}}. \tag{3.13}$$

3.3.7 Zero Knowledge

As with the quadratic equations setting, we only outline here how to achieve zero-knowledge in the non-standard sense (so where the prover and simulator are allowed to form their commitments themselves). In the case of prime-order groups, the technique is the same as with quadratic equations. The simulator can simply solve for $\vec{x}$ and $\vec{y}$ in every individual equation. When it comes to forming commitments, the simulator will form commitments $\vec{c}$ and $\vec{d}$ to one particular pair $\vec{x}$ and $\vec{y}$. When it comes time to form the proofs, the simulator will require a trapdoor that allows it to open $\vec{c}$ and $\vec{d}$ to the individual $\vec{x}$ and $\vec{y}$ that it found for each equation. Using this trapdoor, it can form the proofs exactly as the prover would.

Because the techniques are exactly the same as in the quadratic equations setting, we reference Theorem 3.2.4 for a proof of the following theorem:

Theorem 3.3.2. *The above protocol is a composable non-interactive zero-knowledge proof for satisfiability of a set of pairing product equations with perfect completeness, perfect soundness, and perfect composable zero-knowledge.*

Again, we run into problems when attempting to extend these ideas to composite-order groups, as it is not necessarily feasible to find elements $\vec{x}$ and $\vec{y}$ such that $\vec{x} \bullet \vec{y} = t$. Therefore, we can again only guarantee zero-knowledge for a composite-order group in the trivial case when $t = 1$.

3.3.8 Proof of knowledge

As with the quadratic equations, it is often possible for the CRS generation algorithm K to output some information that would allow us to extract elements from the commitments and therefore from the proofs. We will see more on this in Sections 3.6 and 3.7.

3.4 NIWI proof for satisfiability of a set of equations

Just to tie together all we have done in the previous sections, we outline here a non-interactive witness-indistinguishable proof for proving satisfiability of a set of equations. The protocol is essentially just a mix of the protocols seen in Section 3.2.5 and 3.3.5, and runs as follows:

- **Setup:** $(gk, sk) = ((\mathcal{R}, A_1, A_2, A_T, f), sk) \leftarrow \mathcal{G}(1^k)$.
- **Binding:** We need to consider separate commitment schemes here. This means that $\sigma = (B_1, B_2, B_T, F, \tau_1', \rho_1', \tau_1, \rho_1, \tau_2', \rho_2', \tau_2, \rho_2, \tau_T', \rho_T', \tau_T, \rho_T, \vec{u}, \vec{v}) \leftarrow K(gk, sk)$, where τ_i' and ρ_i' represent the maps used for group elements, τ_i and ρ_i represent the maps used for exponents, and $\vec{u}$ and $\vec{v}$ are commitment keys for binding commitment schemes.
- **Hiding:** Again, we need to consider separate commitment schemes. This means that $\sigma = (B_1, B_2, B_T, F, \tau_1', \rho_1', \tau_1, \rho_1, \tau_2', \rho_2', \tau_2, \rho_2, \tau_T', \rho_T', \tau_T, \rho_T, \vec{u}, \vec{v}) \leftarrow S(gk, sk)$, where τ_i' and ρ_i' represent the maps used for group elements, τ_i and ρ_i represent the maps used for exponents, and $\vec{u}$ and $\vec{v}$ are commitment keys for hiding commitment schemes.
- **Prover:** As input we get gk, σ, a set of N equations and a satisfying witness $\vec{x}, \vec{X}, \vec{y}, \vec{Y}$, where $x_i, y_j \in \mathcal{R}$ and $X_i \in A_1$, $Y_j \in A_2$.
  1. For each x_i form a commitment $c_i = \tau_1(x_i) \prod_j u_j^{r_j}$ for $r_j \leftarrow \mathcal{R}^2$. Similarly, for each y_i form a commitment $d_i = \tau_2(y_i) + \prod_j v_j^{s_j}$ for $s_j \leftarrow \mathcal{R}$. For each group element X_i form a commitment $C_i = \tau_1'(X_i) \prod_j u_j^{r_j'}$ and for Y_i form a commitment $D_i = \tau_2'(Y_i) \prod_j v_j^{s_j'}$.
  2. For each pairing product equation (t_j), invoke the prover from Section 3.3.5 to get a proof $\vec{\pi}_j, \vec{\psi}_j$.

[2]Here we choose to switch back to the multiplicative notation, but of course the commitment is the same no matter what notation we use.

3. Similarly, for each quadratic equation $(\vec{a_k}, \vec{b_k}, \Gamma_k, t_k)$, invoke the prover from Section 3.2.5 to get a proof $\vec{\pi_k}, \vec{\psi_k}$.
4. Output the proof $(\vec{c}, \vec{d}, \vec{C}, \vec{D}, \{\vec{\pi}_i, \vec{\psi_i}\}_{i=1}^N)$.

- **Verifier:** On input gk, σ, a set of equations, and a proof $\vec{c}$, $\vec{d}$, $\vec{C}$, $\vec{D}$, $\{\vec{\pi}_i, \vec{\psi_i}\}_{i=1}^N$:

  1. For each pairing product equation (t_j), check that
  $$\vec{C} * \vec{D} = \tau_T'(t_j)(\vec{u} * \vec{\pi_j})(\vec{\psi_j} * \vec{v}).$$
  2. For each quadratic equation $(\vec{a_k}, \vec{b_k}, \Gamma_k, t_k)$, check that
  $$\vec{a_k} * \vec{d} + \vec{c} * \vec{b_k} + \vec{c} * \Gamma_k \vec{d} = \tau_T(t) + \vec{u} * \vec{\pi_k} + \vec{\psi_k} * \vec{v}.$$
  3. Accept the proof if and only if all these checks pass.

3.5 Subgroup Decision

Although we do not fully outline here the instantiation based on the Subgroup Decision assumption, it is worth discussing. We first recall the assumption:

Assumption 3.5.1 ([BGN05])**.** *Assuming a generation algorithm $\mathcal{G}$ that outputs a tuple (p, q, G, G_T, e) such that $e : G \times G \to G_T$ and G and G_T are both groups of order $n = pq$, it is computationally infeasible to distinguish between an element of G and an element of G_p. More formally, for all PPT adversaries $\mathcal{A}$ there exists a negligible function $\nu(\cdot)$ such that*

$$\begin{aligned}|Pr[(p,q,G,G_T,e) \leftarrow \mathcal{G}(1^k);\ n = pq;\ x \leftarrow G\ :\ \mathcal{A}(n,G,G_T,e,x) = 0]\\ - Pr[(p,q,G,G_T,e) \leftarrow \mathcal{G}(1^k);\ n = pq;\ x \leftarrow G\ :\ \mathcal{A}(n,G,G_T,e,x^q) = 0]| < \nu(k)\end{aligned}$$

where $\mathcal{A}$ outputs a 1 if it believes $x \in G_p$ and 0 otherwise.

In order to use this assumption, the instantiation uses a group G of order $n = pq$. For the commitments, we have $A = B = G$ and use one commitment key: an element u chosen to either generate G or have order q; Subgroup Decision tells us that these choices are indistinguishable. For group elements X, we use τ as the identity. For ring elements x, we use $\tau(x) = g^x$ (or xg if working additively), where g is a generator of G. To define our ρ map, we use an element $\lambda \in \mathbb{Z}_n$ such that $\lambda \equiv 1 \bmod p$ and $\lambda \equiv 0 \bmod q$ and define $\rho(c) = \lambda c$.

The problem with this instantiation is that it is longer the case that $\rho \circ \tau$ is the identity; instead, $\rho \circ \tau$ maps X to its p-component. This means that our commitments cannot be perfectly binding, which further implies that our proofs cannot be perfectly sound. If we recall Property 3 from Section 2.6.1 we see that we can still create proofs that are L_{co} sound, so this instantiation does still have many interesting and useful properties. For our purposes, however, the problem lies in the extractability properties. In Section 4.2 we would like to prove knowledge of an exponent $x \in \mathcal{R}$, and in our applications of this proof technique we will also need to prove knowledge of group elements. To apply these proofs in a useful way, we need them to be *perfectly* extractable, which means that extracting just the p-component will not suffice. We could try to remedy this by restricting the space of X to G_p (which is what Groth and Sahai suggest) but this causes problems for a prover who might not know the factorization of n, as he does not know p and therefore has no way to efficiently sample from G_p.

3.6 SXDH

Here we use the symmetric external Diffie-Hellman assumption (SXDH), which involves a prime order bilinear group defined by $(p, G_1, G_2, G_T, f, g_1, g_2)$, where g_1 and g_2 generate G_1 and G_2 respectively, $f(g_1, g_2)$ generates G_T, and all the G_i are cyclic groups of order p. This assumption states that it is hard to distinguish between (g^a, g^b, g^{ab}) and (g^a, g^b, g^c) (for a random c) in both G_1 and G_2. To define this more formally, we first need to define the decisional Diffie-Hellman assumption:

Assumption 3.6.1 ([NR97])**.** *Assuming a generation algorithm $\mathcal{G}$ that outputs a tuple (p, G, g) such that G is of order p and has generator g, it is computationally infeasible to distinguish between (g, g^a, g^b, g^c) and (g, g^a, g^b, g^{ab}) for $a, b, c \leftarrow \mathbb{Z}_p$. More formally, for all PPT adversaries $\mathcal{A}$ there exists a negligible function $\nu(\cdot)$ such that*

$$\begin{aligned}|&Pr[(p,G,g) \leftarrow \mathcal{G}(1^k);\ a,b,c \leftarrow \mathbb{Z}_p\ :\ \mathcal{A}(g,g^a,g^b,g^c) = 0]\\ &- Pr[(p,G,g) \leftarrow \mathcal{G}(1^k);\ a,b \leftarrow \mathbb{Z}_p\ :\ \mathcal{A}(g,g^a,g^b,g^{ab}) = 0]| < \nu(k)\end{aligned}$$

where $\mathcal{A}$ outputs a 1 *if it believes $c = ab$ and* 0 *otherwise.*

This assumption is only defined for a single group G, but the SXDH assumpion [Sco02] says the decisional Diffie-Hellman problem is hard in both G_1 and G_2.

It is worth spending a little bit of time to discuss how practical it is to use the SXDH assumption. First of all, it is clear that the assumption breaks down in the symmetric case when $G_1 = G_2 = G$, since it would be easy to check if $f(g^a, g^b) = f(g, g^c)$ and thus decide if $c = ab$ or not. Currently, the groups G_1 and G_2 for which SXDH is still considered to be hard are defined as follows: let E be an MNT elliptic curve defined over a finite field $\mathbb{F}_q$ such that $E(\mathbb{F}_q)$ has a subgroup of (large) prime order p and small embedding degree d. Define G_1 to be this subgroup. If we apply the Frobenius map on $\mathbb{F}_{q^d}/\mathbb{F}_q$ to obtain the trace-0 subgroup of $E(\mathbb{F}_{q^d})$, we can now define G_2 to be this subgroup. In this setting (and if we use f to be the Tate pairing), SXDH is still assumed to be a reasonable assumption.

The reason the problem defined by SXDH can be assumed to be hard in groups defined as above is that there exists no efficiently computable distortion map from G_1 to G_2 (there does exist a distortion map from G_2 to G_1 though). This is a result of Verheul [Ver04] and while it does not prove that the SXDH assumption holds in such groups, it does rule out all currently known attacks on DDH (which involve combining distortion maps with pairings).

Type of proof	G_1	G_2	Total
NIWI pairing product equations	4	4	8
NIZK pairing product equations	20	20	40
NIWI quadratic equations (in $\mathbb{Z}_p$)	2	2	4
NIZK quadratic equations	18	18	36

Figure 3.2: Costs for each type of equation under the SXDH assumption

3.6.1 Quadratic equations

Commitment

Here we will define $A = G$ (so A has prime order p) and $B = G^2$, both of which are modules over $\mathbb{Z}_p$. The commitment key will contain $u_1 = (g, h)$, where $h = \alpha g$ for some random $\alpha \in \mathbb{Z}_p^*$, and u_2, where u_2 is such that either $u_2 = tu_1$ or $u_2 = tu_1 - (0, g)$ for a random $t \in \mathbb{Z}_p^*$.

To commit to some $X \in G$ we define $\tau(X) = (0, X)$[3]. We pick randomness $r_1, r_2 \in \mathbb{Z}_p$ and form our commitment $c(X) = \tau(X) + r_1u_1 + r_2u_2$. We also define $\rho(c_1, c_2) = c_2 - \alpha c_1$, so that $\rho \circ \tau$ is the identity map. In the case where $u_2 = tu_1$ we have $c(X) = (0, X) + r_1(g, h) + r_2t(g, h) = ((r_1 + r_2t)g, (r_1 + r_2t)h + X)$, which we can recognize as an ElGamal encryption [ElG84] (in additive form) and so the commitment is perfectly binding. If instead u_1 and u_2 are linearly independent then they form a basis for $B = G^2$, so $\tau(A) \subseteq \langle u_1, u_2\rangle$ and therefore the commitment is perfectly hiding. Note that the commitment keys are computationally indistinguishable, since any algorithm that could distinguish between $(tg, t\alpha g)$ and $(tg, t\alpha g - 1)$ given $(g, \alpha g)$ could also distinguish between $(tg, t\alpha g)$ and (tg, rg) for random r, and this would violate SXDH.

To commit to an exponent, we define u to be either tu_1 or $tu_1 + (0, g)$ with $\tau(x) = xu$ and $\rho(c_1, c_2) = \log_g(c_2 - \alpha c_1)$. We then compute $c(x) = \tau(x) + ru_1$. On a binding key we use $u = tu_1 + (0, g)$, which means $c(x) = x(tu_1 + (0, g)) + ru_1 = ((xt + r)g, (xt + r)h + xg)$, which we again recognize (in additive form) as an ElGamal encryption of xg.

[3]This is what we meant back in Section 3.2.2 when we said that the coset where $\tau(X)$ lived would have a 'representative element.' If we saw an element of B that looked like $b = (b_1, b_2)$ for $b_1 \neq 0$, we could immediately know that $b \neq \tau(X)$ for any $X \in A$.

Note that here the decryption key α is enough to allow us to compute ρ for group elements. Since $\rho(c(X)) = X$ for $X \in G$, access to α would allow us to extract the value inside the commitment, which means that we can make our proofs perfect proofs of knowledge of group elements. For exponents, however, there is no possible trapdoor that will allow us to efficiently compute the discrete log, which means that all we can extract is xg.

Setup

Here we set $B_1 = G_1^2$, $B_2 = G_2^2$, and $B_T = G_T^4$. We can also define F such that

$$F\left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}\right) = \begin{pmatrix} f(x_1,y_1) & f(x_1,y_2) \\ f(x_2,y_1) & f(x_2,y_2) \end{pmatrix}.$$

We have $A_1 = A_2 = A_T = \mathbb{Z}_p$ and $f(x,y) \equiv xy \bmod p$ if we are using exponents, and $A_1 = G_1$, $A_2 = G_2$, $A_T = G_T$, f the Tate pairing if we are using group elements. We can define

$$\tau_T'(z) = \begin{pmatrix} 1 & 1 \\ 1 & z \end{pmatrix} \quad \text{and} \quad \rho_T'\left(\begin{pmatrix} z_{11} & z_{12} \\ z_{21} & z_{22} \end{pmatrix}\right) = z_{22} \cdot z_{12}^{-\alpha_1}(z_{21} \cdot z_{11}^{-\alpha_1})^{-\alpha_2}$$

where the α_i come from the commitment keys – recall that $u_1 = (g_1, \alpha_1 g_1)$ and $v_1 = (g_2, \alpha_2 g_2)$. If we are using group elements, we use the maps τ_T' and ρ_T'; it is easy to check that $\rho_T' \circ \tau_T'$ is the identity. To use exponents, we can expand on these maps to define $\tau_T(z) = \tau_T'(f(g,g)^z)$ and $\rho_T(z) = \log_g(f^{-1}(\rho_T'(z)))$, where $f^{-1}(f(g,z)) := z$. We can see that $\rho_T \circ \tau_T$ is the identity map as well, and also that $F(u_1, Hv_1)$ has no non-trivial solution. This means we don't have to specify any generators H_i.

NIWI proof

- **Setup:** $gk = (p, G_1, G_2, G_T, e, g_1, g_2) \leftarrow \mathcal{G}(1^k)$.
- **Binding:** On input gk compute σ as outlined in the previous section, using (u_1, u_2, v_1, v_2) such that $u_2 = t_1 u_1$ and $v_2 = t_2 v_1$ for random $t_1, t_2 \in \mathbb{Z}_p$.
- **Hiding:** On input gk compute σ as outlined in the previous section, using (u_1, u_2, v_1, v_2) such that $u_2 = t_1 u_1 - (0, g_1)$ and $v_2 = t_2 v_2 - (0, g_2)$ for random $t_1, t_2 \in \mathbb{Z}_p$.
- **Prover:** On input gk, σ, a set of equations, and a witness $\vec{x}, \vec{y}$:
    1. Commit to $\vec{x}$ as $\vec{c} = \tau_1(\vec{x}) + \vec{r}u_1$. Similarly, commit to $\vec{y}$ as $\vec{d} = \tau_2(\vec{y}) + \vec{s}v_1$.
    2. For each quadratic equation $\{\vec{a_i}, \vec{b_i}, \Gamma_i, t_i\}$ form

$$\pi_i = \vec{r}^\top \tau_2(\vec{b_i}) + \vec{r}^\top \Gamma_i \tau_2(\vec{y}) + (\vec{r}^\top \Gamma_i \vec{s} - T_i)v_1$$
$$\psi_i = \vec{s}^\top \tau_1(\vec{a_i}) + \vec{s}^\top \Gamma_i^\top \tau_1(\vec{x}) + T_i u_1.$$

- **Verifier:** On input gk, σ, a set of equations, and a proof $\vec{c}$, $\vec{d}$, $\{(\pi_i, \psi_i)\}_{i=1}^{N}$, check for each equation that

$$\tau_1(\vec{a_i}) * \vec{d} + \vec{c} * \tau_2(\vec{b_i}) + \vec{c} * \Gamma\vec{d} = \tau_T'(t_i) + F(u_1, \pi_i) + F(\psi_i, v_1).$$

3.6.2 Pairing product equations

Commitment

This will be identical to the commitment scheme for the quadratic equations (as mentioned in Section 3.1), but for the convenience of the reader we outline it here using the multiplicative notation.

We again consider $A = G$ and $B = G^2$ for our group G. We define $u_1 = (g, h)$ and $u_2 = (v, w)$ as two elements in B, where $h = g^\alpha$ for $\alpha \in \mathbb{Z}_p$ and either $u_2 = u_1^s$ for some s or u_1 and u_2 are linearly independent (so $u_2 = (g^s, g^{s\alpha-1})$ will work). The DDH assumption tells us that distinguishing between these keys is hard. To commit to an element $X \in A$ we define $\tau(X) = (1, X)$ and $\rho(c_1, c_2) = c_2/c_1^\alpha$. We then compute $c = u_1^{r_1}u_2^{r_2}\tau(X) = (g^{r_1}w^{r_2}, h^{r_1}v^{r_2}X)$ for random r_1, r_2. In the binding setting, we have $c = (g^{r_1+sr_2}, h^{r_1+sr_2}X)$, which we can recognize as an ElGamal encryption (this time in the usual multiplicative form!). In the hiding setting, c is perfectly hiding.

To commit to a ring element x, we use $A = \mathbb{Z}_p$ and the same u_1 and u_2 as before, but we now define $\tau(x) = u^x$, where $u = (g^s, g^{s\alpha+1})$ for a binding key and u_1^s for a hiding key (note $u = u_2 \cdot (1, g)$). This means that $\rho(c_1, c_2) = \log_g(c_2/c_1^\alpha)$ and $c = \tau(x)u_1^r$. Note that in the hiding setting we have a perfectly hiding Pedersen commitment, whereas in the binding setting we have that $c = (g^{sx+r}, g^{\alpha(sx+r)+x})$, which means that x is determined uniquely and the commitment is therefore binding.

As with the quadratic equations setting, $\rho \circ \tau$ is the identity and all we need to compute ρ for an element $X \in A$ is the value α such that $g^\alpha = h$. This means that we can have perfect proofs of knowledge for group elements, but for ring elements all we can extract efficiently is g^x (since there is no trapdoor that makes computing the discrete log efficient).

Setup

Here we again have three groups G_1, G_2, and G_T all of prime order p and generators g_1 and g_2 for G_1 and G_2 respectively. Our $\mathbb{Z}_p$ modules, as described in the previous section, will be $B_1 = G_1^2$, $B_2 = G_2^2$, and $B_T = G_T^4$. Our bilinear map is the same as with quadratic equations, where

$$F\left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}\right) = \begin{pmatrix} f(x_1, y_1) & f(x_1, y_2) \\ f(x_2, y_1) & f(x_2, y_2) \end{pmatrix}.$$

We can also define our τ_T and ρ_T maps to be as they were in the quadratic equations scenario. It is easy to see here (again, as with the quadratic equations) that $\mu_{\vec{u},\vec{v}}$ has a trivial kernel when u_1 and u_2 are linearly independent in B_1 and v_1 and v_2 are linearly independent in B_2. This means that we do not need to include any h_i elements.

NIWI proof

- **Setup:** $gk = (p, G_1, G_2, G_T, f, g_1, g_2) \leftarrow \mathcal{G}(1^k)$.
- **Binding:** On input gk form σ as described in the previous section, using (u_1, u_2, v_1, v_2) such that $u_1 = u_2^r$ and $v_1 = v_2^s$ for random $r, s \in \mathbb{Z}_p$.
- **Hiding:** On input gk form σ as described in the previous section, using (u_1, u_2, v_1, v_2) such that $u_2 = u_1^r * (1, g_1^{-1})$ and $v_2 = v_1^s * (1, g_2^{-1})$ for random $r, s \in \mathbb{Z}_p$.
- **Prover:** On input gk, σ, a set of pairing product equations $\{t_i\}_{i=1}^N$, and a witness $\vec{x}, \vec{y}$:
  1. Commit to each value $x_1, \ldots, x_N$ as $c_q = \tau_1(x_q) \prod u_i^{r_{qi}}$ and to each value $y_1, \ldots, y_N$ as $d_q = \tau_2(y_q) \cdot \prod v_j^{s_{qj}}$ for randomly chosen $r_{qi}, s_{qj} \in \mathbb{Z}_p$.
  2. For each equation pick $t_{i1}, t_{i2} \leftarrow \mathbb{Z}_p$, then for $j = 1, 2$ form and send

$$\pi_{ij} = v_1^{t_{j1}} v_2^{t_{j2}} \prod_{q=1}^{N} d_q^{r_{qi}}$$

$$\psi_{ij} = u_1^{-t_{j1}} u_2^{-t_{j2}} \prod_{q=1}^{N} (1, x_q)^{s_{qi}}$$

- **Verifier:** On input gk, σ, a set of pairing product equations $\{t_i\}_{i=1}^N$, and a proof $\vec{c}$, $\vec{d}$, $\{(\vec{\pi_i}, \vec{\psi_i})\}_{i=1}^N$, check for each equation that

$$\prod_{q=1}^{N} F(c_q, d_q) = \begin{pmatrix} 1 & 1 \\ 1 & t_i \end{pmatrix} F(u_1, \pi_{i1}) F(u_2, \pi_{i2}) F(\psi_{i1}, v_1) F(\psi_{i2}, v_2).$$

3.7 DLIN

Here we use the decisional linear assumption (DLIN) which states that for a prime order bilinear group (p, G, G_T, f, g) (so $G_1 = G_2 = G$ this time) we have that for $(g^\alpha, g^\beta, g^{r\alpha}, g^{s\beta}, g^t)$ with random $\alpha, \beta, r, s \in \mathbb{Z}_p$ it is hard to tell whether $t = r + s$ or t is random. More formally, we define this as follows:

Assumption 3.7.1 ([BBS04]). *Assuming a generation algorithm $\mathcal{G}$ that outputs a tuple (p, G, g) such that G is of order p and has generator g, it is computationally infeasible to distinguish between (a, b, c, a^r, b^s, c^t) and $(a, b, c, a^r, b^s, c^{r+s})$ for $a, b, c \leftarrow G$ and $r, s, t \leftarrow \mathbb{Z}_p$. More formally, for all PPT adversaries $\mathcal{A}$ there exists a negligible function $\nu(\cdot)$ such that*

$$|Pr[(p, G, g) \leftarrow \mathcal{G}(1^k);\ a, b, c \leftarrow G;\ r, s \leftarrow \mathbb{Z}_p\ :\ \mathcal{A}(a, b, c, a^r, b^s, c^{r+s}) = 0]$$
$$- Pr[(p, G, g) \leftarrow \mathcal{G}(1^k);\ a, b, c \leftarrow G;\ r, s, t \leftarrow \mathbb{Z}_p\ :\ \mathcal{A}(a, b, c, a^r, b^s, c^t) = 0]| < \nu(k)$$

where $\mathcal{A}$ outputs a 1 if it believes $t = r + s$ and 0 otherwise.

Again, it is worthwhile to take a minute and discuss the relative strengths and weaknesses of using this assumption. It can be shown that DLIN is at least as hard as DDH; that is that given an algorithm that breaks the DLIN assumption it is possible to construct an algorithm that breaks the DDH assumption. The converse has not been shown, however, which means that it may be possible to be less restrictive in our choice of curves when operating under the DLIN assumption than when operating under the SXDH assumption. The downside, as we see in the following figure, is that DLIN requires more group elements per proof.

Type of proof	G	Total
NIWI pairing product equations	9	9
NIZK pairing product equations	45	45
NIWI quadratic equations (in $\mathbb{Z}_p$)	6	6
NIZK quadratic equations	42	42

Figure 3.3: Costs for each type of equation under the DLIN assumption

3.7.1 Quadratic equations

Commitment

We'll start by letting $f = \alpha g$ and $h = \beta g$ for random $\alpha, \beta \in \mathbb{Z}_p^*$. We use $\mathbb{Z}_p$-modules $A = G$ and $B = G^3$. We will have three u_i in our commitment, where $u_1 = (f, 0, g)$, $u_2 = (0, h, g)$, and $u_3 = ru_1 + su_2$ or $u_3 = ru_1 + su_2 - (0, 0, g)$ for random $r, s \in \mathbb{Z}_p$. The former will produce a binding key, and the latter a hiding key. The DLIN assumption tells us that these are indistinguishable, since an adversary who could distinguish $(rf, sh, (r + s)g)$ from $(rf, sh, (r + s - 1)g)$ could also distinguish $(rf, sh, (r + s)g)$ from (rf, sh, tg) for random t.

To commit to some $X \in A$, we define $\tau(X) = (0, 0, X)$. We then pick three random values r_i and compute $c(X) = \tau(X) + \sum r_i u_i$. As usual, if the u_i are linearly independent we end up with a perfectly hiding commitment scheme. In the other case, we end up with

$$c(X) = ((r_1 + rr_3) \cdot f, (r_2 + sr_3) \cdot h, (r_1 + r_2 + (r + s)r_3) \cdot g + X).$$

We can recognize this as a BBS encryption of X, where $\rho(c_1, c_2, c_3) = c_3 - \frac{1}{\alpha}c_1 - \frac{1}{\beta}c_2$ corresponds to the decryption function. Since $\rho \circ \tau$ is the identity map, this is perfectly binding.

To commit to an exponent $x \in \mathcal{R}$ we define $u = u_3 + (0, 0, g)$, $\tau(x) = xu$, and $\rho(c_1, c_2, c_3) = \log_g(c_3 - \frac{1}{\alpha}c_1 - \frac{1}{\beta}c_2)$. The commitment is very similar – we compute $c(x) = xu + r_1u_1 + r_2u_2$. The hiding case is just the same as above, and in the binding case we have a BBS encryption of xg.

Here the decryption keys α, β provide us with enough information to fully extract $X \in A$ from its commitment. As before, this means that we can turn our proofs into perfect proofs of knowledge of group elements but for exponents can only hope to extract xg.

Setup

Here we have $A_1 = A_2 = A_T = \mathbb{Z}_p$ for exponents and $A_1 = A_2 = G$, $A_T = G_T$ for group elements, and $B_1 = B_2 = G^3$, $B_T = G_T^9$. We use two different bilinear maps F and $\tilde{F}$ defined as follows:

$$\tilde{F}\left(\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}, \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}\right) = \begin{pmatrix} f(x_1, y_1) & f(x_1, y_2) & f(x_1, y_3) \\ f(x_2, y_1) & f(x_2, y_2) & f(x_2, y_3) \\ f(x_3, y_1) & f(x_3, y_2) & f(x_3, y_3) \end{pmatrix}.$$

The map F is defined as $F(x, y) = \frac{1}{2}\tilde{F}(x, y) + \frac{1}{2}\tilde{F}(y, x)$. We also define our bilinear map as $f(x, y) = xy \bmod p$ when using exponents, and f as the Weil or Tate pairing for group elements. Note that for $\tilde{F}$ we have no non-trivial matrices H, while for F we have the matrices

$$H_1 = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, H_2 = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{pmatrix}, \text{ and } H_3 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{pmatrix}.$$

Now all that remains is to define our maps $\tau_T(\cdot)$ and $\rho_T(\cdot)$. We can first define

$$\tau'_T(z) = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & z \end{pmatrix}$$

and

$$\rho'_T\left(\begin{pmatrix} z_{11} & z_{12} & z_{13} \\ z_{21} & z_{22} & z_{23} \\ z_{31} & z_{32} & z_{33} \end{pmatrix}\right) = (z_{33}z_{13}^{-\alpha}z_{23}^{-1/\beta})(z_{31}z_{11}^{-1/\alpha}z_{21}^{-1/\beta})(z_{32}z_{12}^{-1/\alpha}z_{22}^{-1/\beta})^{-1/\beta}.$$

We can see that $\rho'_T \circ \tau'_T$ is the identity map; these are the maps we will use when dealing with group elements. We can further define $\tau''_T(z) = \tau'_T(f(g,z))$ and $\rho''_T(z) = f^{-1}(\rho'_T(z))$, where $f^{-1}(f(g,z)) := z$. Finally, we define the maps we will be using for exponents as $\tau_T(z) = \tau'_T(f(g,g)^z)$ and $\rho_T(z) = \log_g(\rho''_T(z))$.

NIWI proof

- **Setup:** $gk = (p, G, G_T, f, g) \leftarrow \mathcal{G}(1^k)$
- **Binding:** On input gk form σ as described in the previous section, using (u_1, u_2, u_3) such that $u_3 = t_1u_1 + t_2u_2$ for random $t_1, t_2 \in \mathbb{Z}_p$.
- **Hiding:** On input gk form σ as described in the previous section, using (u_1, u_2, u_3) such that $u_3 = t_1u_1 + t_2u_2 - (0,0,g)$ for random $t_1, t_2 \in \mathbb{Z}_p$.
- **Prover:** On input gk, σ, a set of equations $\{\vec{a_i}, \vec{b_i}, \Gamma_i, t_i\}_{i=1}^N$, and a witness $\vec{x}, \vec{y}$:
  1. Commit to $\vec{x}$ as $\vec{c} = \tau(\vec{x}) + R\vec{u}$ and to $\vec{y}$ as $\vec{d} = \tau(\vec{y}) + S\vec{v}$.
  2. For each equation form and send

$$\vec{\phi_i} = R^\top \tau_2(\vec{b_i}) + R^\top \Gamma \tau_2(\vec{y}) + R^\top \Gamma S \vec{u} + S^\top \tau_1(\vec{a_i}) + S^\top \Gamma^\top \tau_1(\vec{x}) + \sum_{j=1}^{k} r_j H_j \vec{u}.$$

- **Verifier:** On input gk, σ, a set of equations $\{\vec{a_i}, \vec{b_i}, \Gamma_i, t_i\}_{i=1}^N$, and a proof $\vec{c}$, $\vec{d}$, $\{\vec{\phi_i}\}_{i=1}^N$, check for each equation that

$$\tau(\vec{a_i}) * \vec{d} + \vec{c} * \tau(\vec{b_i}) + \vec{c} * \Gamma \vec{d} = \tau_T(t_i) + \vec{u} * \vec{\phi_i}.$$

3.7.2 Pairing product equations

Commitment

Let f, g, and h be three generators of G such that $f = g^\alpha$ and $h = g^\beta$. The $\mathbb{Z}_p$-module we will work with here is G^3, and we use commitment keys $u_1 = (f, 1, g)$, $u_2 = (1, h, g)$, and u_3 such that $u_3 = (f^{r_u}, h^{s_v}, g^{t_w})$, where either $t_w = r_u + s_v$ or t_w is random. The DLIN assumption tells us that these choices of keys are indistinguishable.

To commit to a group element we use $A = G$ and $B = G^3$. For $X \in A$ we define $\tau(X) = (1, 1, X)$ and $\rho(c_1, c_2, c_3) = c_1^{-1/\alpha} c_2^{-1/\beta} c_3$ and then compute $c = \tau(X) u_1^r u_2^s u_3^t$ for random $r, s, t \in \mathbb{Z}_p$. In the case where these elements are all linearly independent (so t_w is random) they generate the whole module B and we end up with a perfectly hiding commitment. In the other scenario, we know that $\rho \circ \tau$ is the identity and so our commitment is perfectly binding.

To commit to an element $x \in \mathbb{Z}_p$, we use $A = \mathbb{Z}_p$ and $B = G^3$. We further define $u = u_3 \cdot (1, 1, g)$, $\tau(x) = u^x$ and $\rho(c_1, c_2, c_3) = \log_g(c_1^{-1/\alpha} c_2^{-1/\beta} c_3)$. To commit, we pick $r, s \in \mathbb{Z}_p$ randomly and compute $c = \tau(x) u_1^r u_2^s$. In this case the scenarios are reversed: if $t_w = r_u + s_v$ then this is a perfectly hiding commitment scheme, and if they are linearly independent it is binding.

As with the quadratic equations, α and β are enough to compute ρ for group elements X, but for exponents we can only extract g^x.

Setup

As with quadratic equations, we have $\mathbb{Z}_p$ modules $B_1 = B_2 = G^3$ and $B_T = G_T^9$, and a bilinear map F defined as

$$\left(\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}, \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} \right) \mapsto \begin{pmatrix} f(x_1, y_1) & f(x_1, y_2) f(x_2, y_1) & f(x_1, y_3) f(x_3, y_1) \\ 0 & f(x_2, y_2) & f(x_2, y_3) f(x_3, y_2) \\ 0 & 0 & f(x_3, y_3) \end{pmatrix}.$$

In this case, the corresponding map $\mu_{\vec{u}}$ has a non-trivial kernel that will need to be specified in the CRS, namely the elements

$$h_1 = (0, 1, 0, -1, 0, 0, 0, 0, 0),\ h_2 = (0, 0, 1, 0, 0, 0, -1, 0, 0),\ \text{and } h_3 = (0, 0, 0, 0, 0, 1, 0, -1, 0).$$

NIWI proof

- **Setup:** $gk = (p, G, G_T, f, g) \leftarrow \mathcal{G}(1^k)$.
- **Binding:** On input gk, form σ as described in the previous section, using (u_1, u_2, u_3) such that $u_1 = (f, 1, g)$, $u_2 = (1, h, g)$, and $u_3 = (f^{r_u}, h^{s_v}, g^{r_u + s_v})$ for random $r_u, s_v \in \mathbb{Z}_p$, where f, g, h are generators of G.
- **Hiding:** On input gk, form σ as described in the previous section, using (u_1, u_2, u_3) so the u_i elements are linearly independent in G^3.
- **Prover:** On input gk, σ, a set of equations $\{t_i\}_{i=1}^N$, and a witness $\vec{x}, \vec{y}$:
  1. Commit to each value $x_1, \ldots, x_Q$ as $c_q = \tau(x_q) \prod_i u_i^{r_{qi}}$ and to each value $y_1, \ldots, y_Q$ as $d_q = \tau(y_q) \prod_j u_j^{s_{qj}}$ for randomly chosen $r_{qi}, s_{qj} \in \mathbb{Z}_p$.
  2. For each equation and for $j = 1, 2, 3$ form

$$\phi_{ij} = \prod_{k=1}^{3} u_k^{\sum_{l=1}^{3} t_l h_{ljk}} \prod_{q=1}^{Q} (1, 1, x_q)^{s_{qi}} d_q^{r_{qi}}.$$

- **Verifier:** On input gk, σ, a set of equations $\{t_i\}_{i=1}^N$, and a proof $\vec{c}$, $\vec{d}$, $\{\vec{\phi_i}\}_{i=1}^N$, check for each equation that

$$\vec{c} * \vec{d} = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & t_i \end{pmatrix} (\vec{u} * \vec{\phi_i}).$$

Chapter 4

Extensions and Applications

4.1 Proving two commitments open to the same value

In this section, we use the ideas of Belenkiy et al. [BCKL08] to prove in zero knowledge that two commitments $D_1 = Com(y_1; \vec{r})$ and $D_2 = Com(y_2; \vec{s})$ are commitments to the same value for $y_1, y_2 \in A_2$ (the proof for $x_1, x_2 \in A_1$ is analogous). As we see in the next section, one of the main applications of this technique will be to create NIZK proofs that satisfy a more robust definition of zero-knowledge. To get this stronger notion of zero-knowledge, we first need a stronger requirement on our bilinear map f. Looking back at Property 3 from Definition 2.6.2, we see that we require there to be *some* values g_1 and g_2 such that $f(g_1, g_2)$ is not the identity, but that we do not require this for all elements. To strengthen this requirement, we now say that $f(a, b) = 1$ if and only if either $a = 1$ or $b = 1$. We will refer to this property as *strong non-degeneracy*[1].

Using this stronger definition of a bilinear map, we first form a commitment C to g_1, a generator for the module A_1. Then we prove that the values x and y, contained in the commitments C and D_1/D_2 respectively, are such that $f(x, y) = 1$ and $f(x/g_1, g_2) = 1$. Since $f(a, b) = 1$ if and only if either $a = 1$ or $b = 1$, these proofs show that $y = 1$ and therefore the values contained in D_1 and D_2 must be the same. More precisely, the second proof shows us that C really is a commitment to g_1; it then follows from the first proof (and the fact that $g_1 \neq 1$) that the value contained in D_1/D_2 must be 1 and so the commitments contain the same value.

Since the prover knows the openings for the commitments, he will have no trouble forming the proofs in the usual way. The simulator, on the other hand, will first form $C = Com(g_1; \vec{r})$ just as the prover would (since g_1 and g_2 are public this will not be a problem). It then needs a trapdoor that allows it to open C to 1; in other words a way

[1]It is fairly easy (using a distortion map) to create modified versions of the Weil and Tate pairings that satisfy this property, so it is not unreasonable to expect it from the bilinear maps we use.

to compute randomness $\vec{r}'$ such that $C = Com(1; \vec{r}')$. For the second proof, the simulator has formed the commitments itself and so can form the proof $(\vec{\pi}, \vec{\psi})$ normally, using C as a commitment to g_1. It turns out that if the simulator uses C as a commitment to 1 for the first proof, it will be able to form the proof normally here as well. To see this, we first look at $\vec{\pi_i}$ in Equation (3.11). Other than the public commitment keys v_j, $\vec{\pi_i}$ involves knowing the commitment values d and randomness r_i. Since the equation here is $f(x, y) = 1$ where x is the value contained in C, the simulator knows the randomness r_i (since it formed C itself). In addition, since D_1 and D_2 are public, the simulator can just directly compute $d = D_1/D_2$. As for the second part of our proof (Equation 3.12), $\vec{\psi_j}$ involves knowing x from the commitment C and the randomness s_j from the commitment D_1/D_2. The randomness would otherwise be a problem, but since the simulator is using C as a commitment to $x = 1$ it can form

$$\begin{aligned}
\vec{\psi_j} &= \tau_1(x)^{s_j} \prod_{i=1}^{m} u_i^{\sum_{l=1}^{k} t_l h_{li}} \prod_{i=1}^{m} u_i^{-t_i} \\
&= 1^{s_j} \prod_{i=1}^{m} u_i^{\sum_{l=1}^{k} t_l h_{li}} \prod_{i=1}^{m} u_i^{-t_i} \\
&= \prod_{i=1}^{m} u_i^{\sum_{l=1}^{k} t_l h_{li}} \prod_{i=1}^{m} u_i^{-t_i},
\end{aligned}$$

since τ_1 is a homomorphism. So the simulator can compute $\vec{\psi_j}$ without having to know the randomness s_j.

To summarize, here is the zero-knowledge protocol for proving that two commitments D_1 and D_2 are commitments to the same value:

- **Setup:** $(gk, sk) = ((\mathcal{R}, A_1, A_2, A_T, f, g_1, g_2), sk) \leftarrow \mathcal{G}(1^k)$, where f is strongly non-degenerate (so $f(a, b) = 1$ if and only if $a = 1$ or $b = 1$).

- **Binding:** $\sigma = (B_1, B_2, B_T, F, \vec{u}, \vec{v}) \leftarrow K(gk, sk)$, where $\vec{u}$ and $\vec{v}$ are keys for binding commitment schemes in A_1 and A_2 respectively.

- **Hiding:** $(\sigma = (B_1, B_2, B_T, F, \vec{u}, \vec{v}), td) \leftarrow S_1(gk, sk)$, where $\vec{u}$ and $\vec{v}$ are keys for hiding commitment schemes in A_1 and A_2 respectively, and td is a trapdoor that allows the simulator S_2 to open a commitment to g_1 as a commitment to a 1.

- **Prover:** On input gk, σ, commitments D_1, D_2, and their openings $(y_1, \vec{r})$, $(y_2, \vec{s})$, do the following:

  1. Form a commitment $C = Com(g_1)$.

2. Denote the value contained inside D_1/D_2 as y and the value contained in C as x. Then form a proof $\vec{\pi}_1, \vec{\psi}_1$ as outlined in Section 3.3.5 for the pairing product equation $f(x, y) = 1$ (using commitments C and D_1/D_2).
3. Next, form a proof $\vec{\pi}_2, \vec{\psi}_2$ (again, as in Section 3.3.5) to prove that $f(x/g_1, g_2) = 1$. Because the commitments are homomorphic the prover can use $C/\tau_1(g_1)$ as a commitment to x/g_1 and $\tau_2(g_2)$ as a commitment to g_2 (with randomness 0).
4. Send the proofs $\vec{\pi}_1, \vec{\psi}_1, \vec{\pi}_2, \vec{\psi}_2$ as well as the commitment C.

- **Verifier:** On input gk, σ, commitments D_1, D_2 and a proof $C, \vec{\pi}_1, \vec{\psi}_1, \vec{\pi}_2, \vec{\psi}_2$, check the following (see Section 3.3.1 if you don't remember the vector notation):

$$F(C, D_1/D_2) = (\vec{u} * \vec{\pi_1})(\vec{\psi_1} * \vec{v}) \quad \text{and}$$
$$F(C/\tau_1(g_1), \tau_2(g_2)) = (\vec{u} * \vec{\pi_2})(\vec{\psi_2} * \vec{v}).$$

  Accept iff both checks pass.

- **Simulator:** On input gk, σ, td, and commitments D_1 and D_2 do the following:

1. Form a commitment $C = Com(g_1; \vec{r})$. Use td to compute $\vec{r}'$ such that $C = Com(1; \vec{r}')$.
2. Denote the value contained in D_1/D_2 as y and the value contained in C as x. Form a proof $\vec{\pi}_1, \vec{\psi}_1$ as outlined in Section 3.3.5 to prove that $f(x, y) = 1$, using randomness $\vec{r}'$ (in other words, using C as a commitment to 1).
3. Next, form the proof $\vec{\pi}_2, \vec{\psi}_2$ exactly as in Step 3 of the prover, this time using randomness $\vec{r}$ (so using C as a commitment to g_1).
4. Send the proofs $\vec{\pi}_1, \vec{\psi}_1, \vec{\pi}_2, \vec{\psi}_2$ as well as the commitment C.

Theorem 4.1.1. *The protocol outlined above is a non-interactive zero-knowledge proof that the values contained inside two commitments D_1 and D_2 are the same with perfect completeness, perfect soundness, and perfect zero-knowledge.*

Proof. Let's start with completeness. For a valid pair of proofs, the values contained within D_1 and D_2 really will be the same, which means by the homomorphic properties of our commitments that the value contained in D_1/D_2 will be 1. Similarly, the prover will have formed the commitments correctly, which means that the value contained in C will be g_1. This means that both the first and second proofs are correct, since $f(g_1, 1) = 1$ and $f(g_1/g_1 = 1, g_2) = 1$.

To show perfect soundness, we first analyze our second equation $f(x/g_1, g_2) = 1$. Since we know from gk that $g_2 \neq 1$, the strong non-degeneracy property of f tells us that $x/g_1 = 1$. By the perfect soundness from Theorem 3.3.2, this means that $x = g_1$, so that C is a commitment to g_1. Now, we plug this information into our first equation to see that $f(x,y) = f(g_1,y) = 1$. Again, by the perfect soundness from Theorem 3.3.2 and the strong non-degeneracy of f, this implies that $y = y_1/y_2 = 1$ and so it must be the case that $y_1 = y_2$ and the values contained in D_1 and D_2 are the same.

Finally, we must prove zero-knowledge. To do this, we recall that in the hiding setting, the proofs are perfectly witness indistinguishable and the commitments are perfectly hiding. The prover and the simulator both form the commitment $C = Com(g_1)$; because the commitment the simulator sends is really a commitment to g_1 the commitments received by the verifier will be perfectly indistinguishable. For the proofs, we notice that the simulator does have a valid pair of witnesses for each individual statement and so his proofs for these statements will be valid proofs (and so they will pass the verifier's checks). This means by the perfect WI property of the proofs that the proofs of the simulator will be perfectly indistinguishable from the proofs of the prover and so the verifier will be unable to distinguish between input from the prover and input from the simulator. $\square$

4.1.1 Proving satisfiability of quadratic equations in zero-knowledge

One implication of the above construction is the ability to prove satisfiability of a set of equations in a more robust version of zero-knowledge[2]. If we look back at Section 3.2.7 we can see that our prover (and therefore simulator) is forming the commitments as he forms the proofs. In many applications, however, it makes much more sense for the prover to already have a set of commitments and then form the proofs separately when the time comes. We therefore need a stronger notion of zero-knowledge in which the prover has a set of publicly available commitments; then upon forming a proof for satisfiability of a set of equations he can form new commitments and prove that they are the same as the old ones. Although this adds an extra step in the proof (and therefore makes it less efficient) it has the bonus that now we can simulate the proof quite easily. The simulator will receive the same commitments as the prover, but it will form its own unrelated commitments $\vec{C'}, \vec{D'}$ to values that it knows will work for the set of equations. When it comes time to prove that these are commitments to the same values as $\vec{c}$ and $\vec{d}$ the simulator will fake this proof and then proceed with the values it has chosen.

To make this explicit, we outline a new version of the GS proofs that achieves full zero knowledge:

- **Setup:** $(gk, sk) = ((\mathcal{R}, A_1, A_2, A_T, f, g_1, g_2), sk) \leftarrow \mathcal{G}(1^k)$, where f is strongly non-degenerate (as defined in the previous section).
- **Binding:** $\sigma = (B_1, B_2, B_T, F, \vec{u}, \vec{v}) \leftarrow K(gk, sk)$, where $\vec{u}$ and $\vec{v}$ are keys for binding commitment schemes in A_1 and A_2 respectively.

[2]For simplicity of exposition we only highlight the proof for satisfiability of quadratic equations here, but the technique is easily extended to proofs for a general set of equations.

- **Hiding:** $(\sigma = (B_1, B_2, B_T, F, \vec{u}, \vec{v}), td) \leftarrow S_1(gk, sk)$, where $\vec{u}$ and $\vec{v}$ are keys for hiding commitment schemes in A_1 and A_2 respectively, and td is a trapdoor that allows S_2 to open a commitment $c \in B_1$ or $d \in B_2$ (that it formed itself) as a commitment to any value.

- **Prover:** On input gk, σ, a set of quadratic equations, and commitments $\vec{c}$, $\vec{d}$ and their openings $(\vec{x}, R)$, $(\vec{y}, S)$, do the following:

  1. Form a new pair of commitments $\vec{C} = Com(\vec{x}; R')$ and $\vec{D} = Com(\vec{y}; S')$.
  2. Using the techniques of the previous section, prove that $\vec{c}$ and $\vec{C}$ are commitments to the same value $\vec{x}$, and that $\vec{d}$ and $\vec{D}$ are commitments to the same value $\vec{y}$. Call this proof π_s.
  3. Now, use the commitments $\vec{C}$ and $\vec{D}$ to form the WI proofs as in Section 3.4. Call this proof π_e.
  4. Send the proofs π_s, π_e, as well as the commitments $\vec{C}$ and $\vec{D}$.

- **Verifier:** On input gk, σ, a set of quadratic equations, commitments $\vec{c}, \vec{d}$, and a proof of the form $\vec{C}, \vec{D}, \pi_s, \pi_e$, do the following:

  1. Perform the checks from the previous section using the proof π_s to see that $\vec{c}$ and $\vec{C}$ are commitments to the same values, and that $\vec{d}$ and $\vec{D}$ are commitments to the same values.
  2. Next, perform the checks from Section 3.4 using π_e to see that $\vec{C}$ and $\vec{D}$ (and therefore $\vec{c}$ and $\vec{d}$) satisfy the given set of equations.
  3. Accept iff these checks pass.

- **Simulator:** On input gk, σ, td, a set of quadratic equations, and commitments $\vec{c}$ and $\vec{d}$ do the following:

  1. For each quadratic equation, use the techniques of Section 3.2.7 to either solve for $\vec{x}$ and $\vec{y}$ (in the case of a prime-order group) or simply use $\vec{x} = \vec{0}$ and $\vec{y} = \vec{0}$ if the equation is of a form with $t = 1$ (in the case of a composite-order group).
  2. Using one pair of values $\vec{x}$ and $\vec{y}$ that we found for a particular equation, form commitments $\vec{C}$ and $\vec{D}$. Invoke the simulator from the previous section using td (here, we use td to allow us to open a commitment $c \in B_1$ to g_1 as a commitment to 1) to get a (fake) proof π_s that $\vec{c}$ and $\vec{C}$ are commitments to the same value, as are $\vec{d}$ and $\vec{D}$.
  3. Next, invoke the prover from Section 3.4 to get a proof π_e for the equations, using td to get openings for $\vec{C}$ and $\vec{D}$ to the witnesses $\vec{x}$ and $\vec{y}$ for each equation.

4. Send the proofs π_s, π_e, as well as the commitments $\vec{C}$ and $\vec{D}$.

In terms of efficiency, this adds an extra 32 elements if operating under the SXDH assumption and an extra 36 elements if operating under the DLIN assumption, since each commitment pair requires two WI proofs for pairing product equations, and there are two sets of commitments. We can look back at Figures 3.2 and 3.3 to understand now where these extra group elements come from. It would be possible to reduce these costs somewhat if we were able form the proof π_s in the quadratic equations setting (since in that setting each proof costs either 4 or 6 elements instead of 8 or 9), but there is currently no known method for doing this.

4.2 Proving knowledge of an exponent

One major drawback of the proofs seen in Sections 3.2 and 3.3 is that, while they are proofs of knowledge of group elements, they are not proofs of knowledge of exponents. Recall that in every instantiation, the best we could hope to extract from a commitment to $x \in \mathcal{R}$ was the value g^x (or xg if working additively). If we assume the Discrete Log problem is hard, there is no way to extract x given this value. In many applications of non-interactive zero-knowledge proofs, however, it is often necessary (or at least very useful!) to be able to extract the exponent. For example, one of the necessary properties in anonymous credentials is *unforgeability*, which requires that no PPT adversary $\mathcal{A}$ can create a proof for a message on which he has not previously obtained a signature or a proof. Belenkiy et al. [BCKL08] prove unforgeability by using a signature scheme reminiscent of the Boneh-Boyen signature scheme [BB04] under a slightly non-standard assumption. In Section 4.3 we show how using our technique for proving knowledge of an exponent we can construct an unforgeable non-interactive anonymous credentials scheme directly based on the Boneh-Boyen signature scheme, thus getting around many of the extra steps and assumptions.

4.2.1 Proving two commitments open to the same bit

Before we can outline our non-interactive proof of knowledge (NIZKPoK) of an exponent, we first need to be able to prove that two commitments $c \in B_1$ and $d \in B_2$ open to the same bit $b = 0$ or $b = 1$. To do this, we extend a technique of Groth, Ostrovsky, and Sahai [GOS06] to work in the asymmetric setting. If we define $x_1 \in A_1$ to be the group element contained in c and $x_2 \in A_2$ to be the group element contained in d, we see that we first need to prove that x_1 and x_2 have the same discrete logarithm, and then show that this common discrete logarithm is in fact 0 or 1. To prove the first part of this, we prove satisfiability of the pairing product equation $f(x_1, g_2) = f(g_1, x_2)$, which can also be written as $f(x_1, g_2) \cdot f(g_1, 1/x_2) = 1$. Then, to prove that their common discrete logarithm must either be 0 or 1, we prove satisfiability of the equation $f(x_1, x_2/g_2) = 1$. The full protocol runs as follows:

- **Setup:** $(gk, sk) = ((\mathcal{R}, A_1, A_2, A_T, f, g_1, g_2), sk) \leftarrow \mathcal{G}(1^k)$.
- **Binding:** $(\sigma = (B_1, B_2, B_T, F, \vec{u}, \vec{v}), td_e) \leftarrow K(gk, sk)$, where $\vec{u}$ and $\vec{v}$ are keys for binding commitment schemes in A_1 and A_2 respectively, and td_e is a trapdoor that allows the extractor E to compute the maps ρ_1 and ρ_2 for group elements.
- **Hiding:** $(\sigma = (B_1, B_2, B_T, F, \vec{u}, \vec{v}) \leftarrow S(gk, sk)$, where $\vec{u}$ and $\vec{v}$ are keys for hiding commitment schemes in A_1 and A_2 respectively.
- **Prover:** On input gk, σ, and a value $b \in \{0, 1\}$:
  1. Form commitments $c \in B_1$ and $d \in B_2$ to b. Using the openings for these commitments and defining $x_1 \in A_1$ as the value in c and $x_2 \in A_2$ as the value in d, form a proof as in Section 3.3.5 for the pairing product equation $f(x_1, g_2) \cdot f(g_1, 1/x_2) = 1$. Call this proof π_e.
  2. Next, use the openings for the commitments to form another proof (again, as in Section 3.3.5) for the pairing product equation $f(x_1, x_2/g_2) = 1$. Call this proof π_b.
  3. Send the commitments c and d, as well as the proofs π_e and π_b.
- **Verifier:** On input gk, σ, and a proof c, d, π_e, π_b, do the following:
  1. Invoke the verifier from Section 3.3.5 on input c, d, π_e to check the pairing product equation $f(x_1, g_2) \cdot f(g_1, 1/x_2) = 1$.
  2. Invoke the verifier from Section 3.3.5 on input c, d, π_b to check the pairing product equation $f(x_1, x_2/g_2) = 1$.
  3. Accept iff both the above checks pass.
- **Extractor:** On input gk, σ, td_e, and a proof c, d, π_e, π_b, do the following:
  1. Compute $\rho_1(c)$ to obtain the witness x_1 inside the commitment c.
  2. Similarly, compute $\rho_2(d)$ to obtain x_2.
  3. If $x_1 = 1$ and $x_2 = 1$, output $b = 0$. If $x_1 = g_1$ and $x_2 = g_2$, output $b = 1$. Otherwise, output $\perp$.

Theorem 4.2.1. *The above protocol for proving two commitments open to a* 0 *or* 1 *is a witness-indistinguishable non-interactive proof of knowledge that satisfies perfect completeness, perfect soundness, perfect witness indistinguishability, and perfect extractability.*

Proof. To show perfect completeness, we have that $x_1 = g_1^x$ and $x_2 = g_2^x$, where $x = 0$ or $x = 1$. Then for the first equation we have

$$\begin{aligned} f(x_1, g_2)f(g_1, 1/x_2) &= f(g_1^x, g_2)f(g_1, 1/g_2^x) \\ &= f(g_1, g_2)^x f(g_1, g_2)^{-x} \\ &= f(g_1, g_2)^{x-x} \\ &= 1, \end{aligned}$$

and for the second equation we have

$$\begin{aligned} f(x_1, x_2/g_2) &= f(g_1^x, g_2^x/g_2) \\ &= f(g_1, g_2)^{x(x-1)}. \end{aligned}$$

In the case that $x = 0$, this becomes $f(g_1, g_2)^{0(-1)} = 1$. In the case that $x = 1$, this becomes $f(g_1, g_2)^{1(1-1)} = 1$. Since equality holds in either case, we get perfect completeness.

To show perfect soundness of the protocol, let $x_1 = g_1^x$ and $x_2 = g_2^y$. Then we have the following derivation:

$$\begin{aligned} f(x_1, g_2)f(g_1, 1/x_2) &= f(g_1^x, g_2)f(g_1, 1/g_2^y) \\ &= f(g_1, g_2)^x f(g_1, g_2^{-y}) \\ &= f(g_1, g_2)^x f(g_1, g_2)^{-y} \\ &= f(g_1, g_2)^{x-y}. \end{aligned}$$

So, if it is the case that $f(x_1, g_2)f(g_1, 1/x_2) = 1$, it must be the case that $f(g_1, g_2)^{x-y} = 1$, which further implies by the non-degeneracy of f and the soundness from Theorem 3.3.1 that $x - y = 0$, $x = y$. So, any two values x_1 and x_2 satisfying the first pairing product equation really must have the same discrete logarithm. We can now plug this information into the second equation to see that

$$\begin{aligned} f(x_1, x_2/g_2) &= f(g_1^x, g_2^x/g_2) \\ &= f(g_1^x, g_2^{x-1}) \\ &= f(g_1, g_2)^{x(x-1)}. \end{aligned}$$

Again, the non-degeneracy of our bilinear map and Theorem 3.3.1 tell us that in order for the equation $f(x_1, x_2/g_2) = 1$ to hold, it must be the case that $x(x-1) = 0$, or $x^2 = x$. Since this is only true when $x = 0$ or $x = 1$, this second equation tells us that the common discrete log of x_1 and x_2 really must be 0 or 1.

Perfect witness indistinguishability follows directly from Theorem 3.3.1, so all we have left to show is perfect extractability. This follows from the fact that, in the binding setting, $\rho_1 \circ \tau_1$ and $\rho_2 \circ \tau_2$ are both required to be the identity map. In addition, it is also the case that $\rho_1(u_i) = 0$ for all i and $\rho_2(v_j) = 0$ for all j. This means that $\rho_1(c) = x_1$ and $\rho_2(d) = x_2$, where $x_1 = g_1^b$ and $x_2 = g_2^b$. Because we are only interested in extracting $b \in \{0, 1\}$, we can ignore all other possible values for b (as the proof will not verify for other values of b anyway). For $b = 0$, it will be the case that $x_1 = 1$ and $x_2 = 1$, which means our extractor as defined above will output the correct bit. For $b = 1$ it will be the case that $x_1 = g_1$ and $x_2 = g_2$, which means our extractor will be correct in this case as well and we are done. $\square$

4.2.2 Proof of knowledge for an exponent

To prove knowledge of the exponent contained in a given commitment $c \in B_1$[3], we consider as usual an $\mathcal{R}$-module A_1 with a binding key $\vec{u}$ for forming commitments. We can then commit to an exponent $x \in \mathcal{R}$ by first considering the binary representation of x, which we can write as $x = \sum_{j=1}^{k} x_j \cdot 2^j$ for $k = \lfloor \log_2(x) \rfloor$. First, we form commitments in A_1 to each bit of x; call these commitments $\{c_i\}_{i=1}^{k}$. Because we are not necessarily in the symmetric setting, we will also need to form commitments in A_2 to each bit of x. Next, we use the techniques of the previous section to prove that each of these pairs of commitments opens to either a 0 or a 1. Finally, we compute $c' = \prod_{i=1}^{k} c_i \cdot 2^i$. Since the commitments are homomorphic, this will be a commitment to $x = \sum_{j=1}^{k} x_j \cdot 2^j$ as long as we have formed our commitments correctly. So, all that's left to do is use the techniques of Section 4.1 to prove that c and c' are commitments to the same value, which is our original exponent x.

- **Setup:** $(gk, sk) = ((\mathcal{R}, A_1, A_2, A_T, f, g_1, g_2), sk) \leftarrow \mathcal{G}(1^k)$, where f is strongly non-degenerate.
- **Binding:** $(\sigma = (B_1, B_2, B_T, F, \vec{u}, \vec{v}), td_e) \leftarrow K(gk, sk)$, where $\vec{u}$ and $\vec{v}$ are keys for binding commitment schemes in A_1 and A_2 respectively, and td_e is a trapdoor that allows the extractor E to compute the maps ρ_1 and ρ_2 for elements in A_1 and A_2.
- **Hiding:** $(\sigma = (B_1, B_2, B_T, F, \vec{u}, \vec{v}), td_s) \leftarrow S_1(gk, sk)$, where $\vec{u}$ and $\vec{v}$ are keys for hiding commitment schemes in A_1 and A_2 respectively, and td_s is a trapdoor that allows the simulator S_2 to open a commitment to g_1 as a commitment to 1.
- **Prover:** On input gk, σ, a commitment c to a value $x \in \mathcal{R}$ and its opening (x, r), do the following:
    1. Form $2k$ commitments (recall that $k = \lfloor \log_2(x) \rfloor$) $c_i = Com(x_i)$ in A_1 and $d_i = Com(x_i)$ in A_2, where the x_i are such that $x = \sum_{j=1}^{k} x_j \cdot 2^j$. Compute also the commitment $c' = \prod_{j=1}^{k} c_j \cdot 2^j$.
    2. Use the techniques of Section 4.2.1 to prove that c_i and d_i are commitments to 0 or 1 for all i. Call this proof π_{or}.

[3]We describe the proof here for $c \in B_1$, but the proof is analogous for a commitment $d \in B_2$.

3. Use the techniques of Section 4.1 to prove that c and c' are commitments to the same value. Call this proof π_s.
4. Send the proofs π_{or} and π_s as well as the commitments $\{c_i\}_{i=1}^k$, and $\{d_i\}_{i=1}^k$.

- **Verifier:** On input gk, σ, a commitment c, and a proof $\{c_i\}_{i=1}^k, \{d_i\}_{i=1}^k, \pi_{or}, \pi_s$, do the following:

  1. Use the techniques of Section 4.2.1 to check the proof π_{or} for showing that each c_i and d_i are commitments to the same bit b.
  2. Compute the value $c' = \prod_{j=1}^k c_j \cdot 2^j$ and invoke the verifier from Section 4.1 on input c, c', π_s to check that c and c' are commitments to the same value.
  3. Accept iff all these checks pass.

- **Simulator:** On input gk, σ, td_s, and a commitment c do the following:

  1. Pick a value $x \in \mathcal{R}$ and complete Steps 1 and 2 as the prover would to form the commitments $\{c_i\}_{i=1}^{k=\lfloor \log_2(x) \rfloor}$ in A_1 and $\{d_i\}_{i=1}^k$ in A_2 to the bits x_i, $c' = \prod_{j=1}^k c_j \cdot 2^j$, and the proof π_{or}.
  2. Use the trapdoor td_s to invoke the simulator in Section 4.1 and simulate the proof that c and c' are commitments to the same value. Denote the simulated proof as π_s.
  3. Send all the same information as the prover; namely the proofs π_{or} and π_s as well as the commitments $\{c_i\}_{i=1}^k$, and $\{d_i\}_{i=1}^k$.

- **Extractor:** On input gk, σ, td_e, a commitment c, and a proof consisting of commitments $\{c_i\}_{i=1}^k, \{d_i\}_{i=1}^k$ and proofs π_{or}, π_s, do the following:

  1. For each commitment c_i, use td_e to invoke the extractor in Section 4.2.1 and recover the i-th bit x_i of x.
  2. Use the individual bits to recover the exponent x by computing $x = \sum_{j=1}^k x_j \cdot 2^j$.

Theorem 4.2.2. *The above protocol constitutes a non-interactive zero-knowledge proof of knowledge of an exponent that satisfies perfect completeness, perfect soundness, perfect zero knowledge, and perfect extractability.*

Proof. The completeness of the protocol follows from the completeness of its individual parts. An honest prover committing to the bits of an exponent $x \in \mathcal{R}$ will be forming commitments to 0 or 1 values, so by Theorem 4.2.1 the verifier will accept π_{or}. Similarly, the homomorphic property of GS commitments tells us that $c(x) = \prod_{i=1}^{\log_2(x)} c_i \cdot 2^i$ (where each c_i represents a commitment to the i-th bit of x), so that c and c' really are commitments to the same value and Theorem 4.1.1 tells us that the proof π_s will verify as well.

Similarly, perfect soundness also follows from Theorems 4.2.1 and 4.1.1.

To show perfect zero-knowledge, we observe that in Step 1, the simulator is forming the bit commitments $\{c_i\}$ and $\{d_i\}$ just as the prover would. This means that the simulator will have legitimate witnesses for the proof π_{or}, which by the perfect witness indistinguishability from Theorem 4.2.1 means the π_{or} output by the prover and the π_{or} output by the simulator will have the same distribution. Finally, Theorem 4.1.1 gives us perfect zero-knowledge on π_s, which means the total outputs of the prover and simulator will be distributed identically and so we get perfect zero-knowledge.

Finally, we need to show perfect extractability. By Theorem 4.2.1, we know that each of the individual bit commitments c_i is perfectly extractable. This means that we can recover each bit of x, which of course means we can recover the whole exponent x. $\square$

4.2.3 The symmetric setting

One important observation about the above protocol is that it becomes twice as efficient if run in the symmetric setting where $A_1 = A_2 = A$. In Step 1, the prover will no longer need to form two sets of commitments to the bits of x and so will instead just compute $k = \lfloor \log_2(x) \rfloor$ commitments $c_i = Com(x_i)$. In Step 2, the proof π_{or} will be half the size. To see this, recall that for each pair c_i and d_i we were proving

$$f(x_1, g_2)f(g_1, 1/x_2) = 1 \quad \text{and} \quad f(x_1, x_2/g_2) = 1.$$

Because it is now the case that $x_1 = x_2 = x$ and $g_1 = g_2 = g$, we need only prove the second equation to show that the discrete logarithm of x is equal to 0 or 1, as the first equation follows trivially. So, while the proof π_s remains the same, there will only be half as many bit commitments and the proof π_{or} will be twice as small as in the asymmetric setting.

4.2.4 Efficiency

To analyze the efficiency of the above protocol for an exponent $x \in \mathcal{R}$, we consider instantiating it under the SXDH and DLIN assumptions. Under SXDH the prover sends commitments $\{c_i\}$ and $\{d_i\}$ and proofs π_{or} and π_s. Each commitment c_i or d_i contains 2 group elements, and there are $2\log_2(x)$ of them, which means that our commitments contribute $4\log_2(x)$ group elements. The proof π_{or} consists of proofs for $2\log_2(x)$ pairing product equations (two for each bit) and the proofs contain 8 group elements each. This means that π_{or} contains $16\log_2(x)$ group elements in total. Finally, π_s consists of one commitment and proofs for 2 pairing product equations. This means that our grand total under the SXDH assumption is a proof consisting of $20\log_2(x) + 18$ group elements.

Because the DLIN assumption operates in the symmetric setting, we can reduce the cost somewhat. This time the prover only has to send $\log_2(x)$ commitments $\{c_i\}$, where each commitment contains 3 group elements, and so the commitments contribute $3\log_2(x)$ group elements. The proof π_{or} now only consists of proofs for $\log_2(x)$ pairing product equations (this time one equation per bit), where each proof contains 9 group elements, so that the size of π_{or} is $9\log_2(x)$ group elements. The number of commitments and equations in π_s remains the same, but it now contains 3 additional group elements (one for each equation and one for the commitment). This means our total number of group elements under the DLIN assumption is $12\log_2(x) + 21$.

It would be possible to reduce this cost even further by formulating the equations in Section 4.2.1 as quadratic equations instead of pairing product equations, as proofs for quadratic equations are half the size under the SXDH assumption (4 elements instead of 8) and two-thirds the size under the DLIN assumption (6 instead of 9). We leave this as an interesting open question. We also mention that because the common reference string σ does not require any additional elements, its size remains constant.

4.2.5 Proving a value is contained within a given range

One case in which we can use the above technique in existing protocols is proving knowledge of an exponent x that is contained within a range whose size is a power of 2. We can consider only the case when we want to prove that $0 \leq x < \Delta$, since it is always possible to formulate a range proof in this way (as an example, consider a proof of the form $lo \leq x < hi$. Then we can prove $0 \leq x - lo < hi - lo$ and since lo and hi are public parameters this will prove the original statement).

If Δ is a power of 2 (so $\log_2(\Delta) = \delta$), then to prove that $0 \leq x \leq \Delta$ we can prove knowledge of an exponent using the techniques of Section 4.2. In addition to the normal verification, the verifier will also check the number of commitments $\{c_i\}_{i=1}^{k}$ received. If there are at most δ of these, the verifier will know that $k = \lfloor \log_2(x) \rfloor \leq \log_2(\Delta)$ and so x really is in the correct range.

4.3 Anonymous credentials

An important application of our protocol for proving knowledge of an exponent is in anonymous credentials. In particular, the extractability of exponents allows us to relax the assumptions made by Belenkiy et al. [BCKL08] in their construction of anonymous credentials, and rely instead only on the Strong Diffie-Hellman (SDH) assumption used by Boneh and Boyen [BB04]. We first recall a rather strong assumption used by Belenkiy et al. called the Hidden SDH assumption:

Assumption 4.3.1 ([BCKL08])**.** *Assuming a bilinear map $e : G_1 \times G_2 \to G_T$ where g_1 is a generator for G_1 and g_2 is a generator for G_2, on input $g_1, g_1^x, u \in G_1$, $g_2, g_2^x \in G_2$ and $\{g_1^{1/(x+c_i)}, g_2^{c_i}, u^{c_i}\}_{i=1}^{q(k)}$ for a function $q(\cdot)$ polynomially bounded in the size of the security parameter k, it is computationally infeasible to output a new tuple $(g_1^{1/(x+c)}, g_2^c, u^c)$. To put this more formally, there exists a negligible function $\nu(\cdot)$ such that for all PPT $\mathcal{A}$ and all polynomially bounded functions $q : \mathbb{Z} \to \mathbb{Z}$*

$$\begin{aligned}
Pr[(p,&G_1, G_2, G_T, e, g_1, g_2) \leftarrow \mathcal{G}(1^k);\; u \leftarrow G_1;\; x, c_1, \ldots, c_{q(k)} \leftarrow \mathbb{Z}_p;\\
&(A, B, C) \leftarrow \mathcal{A}(p, G_1, G_2, G_T, e, g_1, g_1^x, g_2, g_2^x, u, \{g_1^{1/(x+c_i)}, g_2^{c_i}, u^{c_i}\}_{i=1}^{q(k)}) \;:\\
&(A, B, C) = (g_1^{1/(x+c)}, g_2^c, u^c) \wedge c \neq c_i \;\forall\; 1 \leq i \leq q(k)] < \nu(k).
\end{aligned}$$

This assumption is analogous to the Hidden SDH assumption as put forth by Boyen and Waters [BW07], but it extends the assumption further by defining it over asymmetric maps. In addition to this assumption, Belenkiy et al. rely on another new assumption: the Triple DH assumption.

Assumption 4.3.2 ([BCKL08])**.** *Assuming a bilinear map $e : G_1 \times G_2 \to G_T$ where g_1 is a generator for G_1 and g_2 is a generator for G_2, on input $g_1, g_1^x, g_1^y, g_2, g_2^x, \{c_i, g_1^{1/(x+c_i)}\}_{i=1}^{q(k)}$ for $q(\cdot)$ polynomially bounded in the size of the security parameter k, it is computationally infeasible to output a new tuple $(g_2^{ax}, g_1^{ay}, g_1^{axy})$. More formally, there exists a negligible function $\nu(\cdot)$ such that for all PPT $\mathcal{A}$ and polynomially bounded functions $q : \mathbb{Z} \to \mathbb{Z}$*

$$\begin{aligned}
Pr[(p,&G_1, G_2, G_T, e, g_1, g_2) \leftarrow \mathcal{G}(1^k);\; (x, y, \{c_i\}_{i=1}^{q(k)}) \leftarrow \mathbb{Z}_p;\\
&(A, B, C) \leftarrow \mathcal{A}(p, G_1, G_2, G_T, e, g_1, g_1^x, g_1^y, g_2, g_2^x, \{c_i, g_1^{1/(x+c_i)}\}_{i=1}^{q(k)}) \;:\\
&\exists\; a :\; (A, B, C) = (g_2^{ax}, g_1^{ay}, g_1^{axy})] < \nu(k).
\end{aligned}$$

Finally, we also recall the SDH assumption:

Assumption 4.3.3 ([BB04])**.** *Assuming a bilinear map $e : G_1 \times G_2 \to G_T$ where g_1 is a generator for G_1 and g_2 is a generator for G_2, on input $(g_1, g_1^x, g_1^{(x^2)}, \ldots, g_1^{x^q}, g_2, g_2^x)$ it is computationally infeasible to output a pair $(g_1^{1/(x+c)}, c)$ where $c \neq -x$. More formally, there exists a negligible function $\nu(\cdot)$ such that for all PPT $\mathcal{A}$ and all polynomially bounded functions $q : \mathbb{Z} \to \mathbb{Z}$*

$$\begin{aligned}
Pr[(p,&G_1, G_2, G_T, e, g_1, g_2) \leftarrow \mathcal{G}(1^k);\; x \leftarrow \mathbb{Z}_p;\\
&(A, B) \leftarrow \mathcal{A}(p, G_1, G_2, G_T, e, g_1, \{g_1^{(x^i)}\}_{i=1,\ldots,q(k)}, g_2, g_2^x) \;:\\
&(A, B) = (g^{1/(x+c)}, c) \wedge c \neq -x] < \nu(k).
\end{aligned}$$

SDH is a weaker assumption than Hidden SDH and Triple DH and so it would be nice to rely on this assumption alone, while still gaining the advantages of the constructions put forth by Belenkiy et al. To do this, we use the techniques from Section 4.2 to bypass the need for the Hidden SDH and Triple DH assumptions.

One of the core building blocks for anonymous credentials is the idea of a signature scheme. A signature scheme consists of four PPT algorithms: a setup algorithm Setup that outputs some public parameters, a key generation algorithm KeyGen that uses the public parameters to generate signing public and private keys (pk, sk), a signing algorithm Sign that takes in the secret key and a message m to sign and outputs a signature σ, and finally a signature verification algorithm $\mathsf{VerifySig}$ that takes in (pk, m, σ) and accepts if and only if σ is a valid signature on m. Here we recall these algorithms as defined by Boneh and Boyen:

- **Setup:** $gk = (p, G_1, G_2, G_T, e, g_1, g_2) \leftarrow \mathsf{Setup}(1^k)$, where G_1, G_2, and G_T are all groups of order p, e is a bilinear map from $G_1 \times G_2$ to G_T, and g_1 and g_2 are generators for G_1 and G_2 respectively.

- **Key generation:** $(pk, sk) \leftarrow \mathsf{KeyGen}(gk)$, where pk and sk are chosen as follows: KeyGen picks random values $x, y \leftarrow \mathbb{Z}_p^*$ and computes $u = g_2^x$ and $v = g_2^y$. Then $pk = (u, v)$ and $sk = (x, y)$.

- **Signing:** $(\sigma, r) \leftarrow \mathsf{Sign}(gk, sk, m)$, where r is chosen randomly from $\mathbb{Z}_p^*$ and σ is computed as $\sigma = g_1^{1/(x+m+yr)}$.

- **Verifying a signature:** $b \leftarrow \mathsf{VerifySig}(gk, pk, m, (\sigma, r))$, where $b = 1$ if and only if $e(\sigma, u \cdot g_2^m \cdot v^r) = e(g_1, g_2)$.

Boneh and Boyen showed that, under the SDH assumption, this signature scheme achieves existential unforgeability against adaptive chosen-message attacks, so that it is secure in the strongest sense possible. While this signature scheme is initially only well-defined if we consider $m \in \mathbb{Z}_p^*$, it is possible to use a collision-resistant hash function that takes a value m from some larger domain and maps it into $\mathbb{Z}_p^*$ to achieve a larger message space.

In order to use this signature scheme for anonymous credentials, we will need to define three more algorithms to extend our signature scheme into a *P-signature scheme* (more specifically, a non-interactive P-signature scheme), as defined by Belenkiy et al. We now require interactive $\mathsf{ObtainSig}$ and $\mathsf{IssueSig}$ algorithms for issuing credentials, as well as non-interactive Prove and $\mathsf{VerifyProof}$ algorithms for proving/verifying possession of a credential. We must also extend our Setup algorithm to output parameters for the GS commitment scheme in addition to the parameters for the Boneh-Boyen signature scheme. These new algorithms run as follows:

- **Obtaining/Issuing signatures:** Here, the user would like to obtain a signature from the issuer on a message m without revealing m to the issuer. To do this, the user forms a GS commitment $c = Com(m; r)$ and gives c to the issuer. The user then picks r_1 and r_2 randomly from $\mathbb{Z}_p$ and the user and the issuer engage in a secure two-party computation protocol[4]. The user's private inputs are r_1, r_2, and the opening (m, r) for c; the issuer's private inputs are the secret key for the Boneh-Boyen signature scheme, namely $sk = (x, y)$, and a randomly picked $r' \leftarrow \mathbb{Z}_p$. As private output, the issuer obtains $\alpha = (x + m + yr_1r') \cdot r_2$ if (m, r) was a valid opening for c and $\perp$ otherwise. If $\alpha = \perp$, the issuer terminates. If $\alpha \neq \perp$, the issuer computes $\sigma' = g_1^{\alpha}$ and sends σ' and r' to the user. The user then computes $\sigma = (\sigma')^{r_2}$ and $r = r_1 \cdot r'$. The user must also check that (σ, r) is a valid signature (using $\mathsf{VerifySig}$ from above).

- **Proving a valid signature:** To prove knowledge of a valid signature (σ, r) on a message m, first form Groth-Sahai commitments $c_\sigma = Com(\sigma)$, $c_m = Com(g_2^m)$, and $c_r = Com(v^r)$[5]. For the latter two commitments, also form the commitments to the bits of m and r as outlined in Section 4.2. Form a proof π_s for the pairing product equation $e(\sigma, u \cdot g_2^m \cdot v^r) = e(g_1, g_2)$ to prove the validity of the signature, and then also form proofs (again, as outlined in Section 4.2) π_m and π_r to prove knowledge of the values m and r contained in c_m and c_r respectively.

- **Verifying a proof:** To verify that a proof formed as above is correct, perform all the checks from Section 4.2 to check the proofs of knowledge π_m and π_r, as well as a check that the proof π_s has been formed correctly for the pairing product equation $e(\sigma, u \cdot g_2^m \cdot v^r) = e(g_1, g_2)$. Output accept if and only if all these checks pass.

In addition, we also need to define what it means for a P-signature scheme to be secure. For this, we require the underlying signature scheme be secure, the underlying commitment scheme to be perfectly binding and strongly computationally hiding, and the underlying non-interactive proof system to satisfy the properties from Section 2.6.1. We also require the following five properties[6]:

1. **Correctness:** If all parties behave honestly, a user who obtains a P-signature from an issuer will always be able to prove that he has a valid signature.

2. **Signer privacy:** No PPT adversary $\mathcal{A}$ can distinguish between an $\mathsf{IssueSig}$ interaction with the honest issuer and with a simulator who has access to a signing oracle.

[4]We follow Belenkiy et al. in saying that, although it is slightly expensive, the two-party protocol of Jarecki and Shmatikov [JS07] can be adapted to work here.

[5]This operation and the extraction properties required for c_r are not quite well-defined, as we have not considered commitments with a base other than a group generator. For a full treatment of this question, see [BCKL08].

[6]For more rigorous definitions, see Definition 3 from [BCKL08].

3. **User privacy:** No PPT adversary $\mathcal{A}_1, \mathcal{A}_2$ can distinguish between an ObtainSig interaction with the honest user who wants a signature on a message m and with a simulator who does not have access to m.

4. **Zero knowledge:** No PPT adversary $\mathcal{A}$ can distinguish between the output of the Prove algorithm and a simulator.

5. **Unforgeability:** No PPT adversary $\mathcal{A}$ can create a proof for a message m for which he has not previously obtained either a signature or a proof.

Theorem 4.3.4. *The above P-signature construction is secure given the security of the Boneh-Boyen signature scheme and the Groth-Sahai commitments and proofs.*

Proof. The security of the signature scheme follows directly from Boneh and Boyen [BB04], and the binding of the commitment scheme as well as the completeness, soundness, and zero-knowledge properties of the proof system follow directly from Groth and Sahai [GS08]. Similarly, correctness and zero-knowledge follow from the perfect completeness and perfect zero-knowledge in Theorems 3.3.2 and 4.2.2.

Signer privacy and user privacy both follow from the security of the two-party computation; we'll start with signer privacy. To show this property, we need to construct an algorithm SimIssue with access to a signing oracle that can successfully simulate the behavior of the issuer, even when dealing with an adversarial user. First, upon input the commitment c, SimIssue will invoke the simulator S_I for the two-party computation. This simulator has the power that it can extract the user's private input, so from S_I we obtain r_1, r_2, and (m, r). SimIssue must then check that (m, r) is a valid opening for the commitment c; if not, it will terminate (as the issuer would). Otherwise, it queries the signing oracle on message m to get back a valid signature (σ, r) for m. It then sends $\sigma' = \sigma^{1/r_2}$ and $r' = r/r_1$ to the user. Since we know that (σ, r) is a valid signature for m, the only way for the user to distinguish between talking to the simulator and the issuer would be if the user's input to the two-party computation was incorrect. This breaks the security of the two-party computation.

To show user privacy we construct a simulator SimObtain who has as input a commitment c to the user's message m but does not know the opening of c. SimObtain forms a commitment c' to a random value x (it is enough to always use $x = 0$), using some randomness r'. SimObtain then sends this value c' to the adversarial issuer and then invokes the simulator S_U for the two-party computation (unlike S_I, S_U acts as the user). This simulator can extract the issuer's private input, which in this case is some $sk' = (x', y')$. The simulator S_U will pick a random value α and continue to act as the user in the protocol such that if the adversarial issuer outputs a value $\alpha' \neq \perp$, it will be the case that $\alpha' = \alpha$. Upon receiving the outcome of the two-party computation, SimObtain acts just as the user would (so checks the signature obtained for the message x). Any issuer who could tell that he was talking with a simulator would break the security of the two-party computation.

Finally, we must show unforgeability. By our definition of unforgeability, we know that for an adversary $\mathcal{A}$ to succeed in outputting a successful forgery it must be that $\mathsf{VerifyProof}(c_\sigma, c_m, c_r, \pi_s, \pi_m, \pi_r) = \mathsf{accept}$ and we extract m, σ, r, but one of three cases holds: either (1) $\mathsf{VerifySig}(m, \sigma, r) = \mathsf{reject}$, (2) c_m is not a commitment to m, or (3) $\mathcal{A}$ never queried the signing oracle on message m. We know that (1) can never happen, as the soundness properties from Theorem 3.3.2 ensure that π_s proves the validity of the signature. Similarly, we know from Theorem 4.2.2 that π_m proves that c_m must be a commitment to m, so (2) can never happen. Finally, by the unforgeability of the Boneh-Boyen signature scheme, we know that (3) can never happen and so we are done. $\square$

The construction of anonymous credentials on top of this P-signature scheme is straightforward and given by Belenkiy et al., we therefore do not need to describe it in depth here. Briefly, an anonymous credentials scheme consists of various users and a credential authority, where each user has a secret key sk_U. Suppose a user Alice wishes to obtain a credential from Carol, and then prove (anonymously) to Bob and Dave that she does in fact have this credential. First, Alice must obtain a credential from the credential authority. Then, she must register two pseudonyms with Carol – she does this by forming commitments C_B and C_D to her secret key sk_A and using these commitments as her pseudonyms. By the hiding/binding properties of the GS commitment scheme the identity associated with both C_B and C_D is unique, but at the same time it is hard for users Bob and Dave to link these two pseudonyms. Now, under either pseudonym, Alice and Carol run the $\mathsf{ObtainSig}$ and $\mathsf{IssueSig}$ protocols. The resulting output for Alice will be a signature (σ, r) on her secret key sk_A; this is her credential. To prove possession of this credential to Bob, Alice can first run the Prove protocol to obtain a proof $c_\sigma, c_m, c_r, \pi_s, \pi_m, \pi_r$, where in this case $m = sk_A$. She also must link this proof to her pseudonym, which means proving, using the techniques of Section 4.1, that c_m and C_B are commitments to the same value (to prove possession to Dave Alice would instead prove that c_m and C_D are commitments to the same value). Upon receiving this proof, Bob or Dave need only run the $\mathsf{VerifyProof}$ algorithm and the verification from Section 4.1 to be satisfied with Alice's possession of the credential.

In addition to being useful for anonymous credentials, P-signature schemes are a valuable building block on their own. Although the P-signature scheme created here is less efficient than the one created by Belenkiy et al., it has the advantage that it is a much simpler scheme with a proof of security that it only requires the SDH assumption, as opposed to the Hidden SDH and Triple DH assumptions required by Belenkiy et al. in their proof of security.

Chapter 5

Conclusions

In this thesis, I have demonstrated the usefulness of the Groth-Sahai proof system by extending it to form a non-interactive zero-knowledge proof of knowledge of an exponent. This allows GS proofs to be used more fully in a variety of applications; to demonstrate one of these applications I construct a simple and unforgeable anonymous credentials scheme. I have also shown the usefulness of the quadratic equations setting and its improved efficiency over the pairing product equations setting, an observation which will hopefully encourage others in the future to formulate problems in terms of quadratic equations.

Although I have demonstrated a way to extend the GS proof system to obtain NIZKPoK proofs for exponents, using such proofs in practice would require thousands of group elements if working over prime-order elliptic curve groups with a 384-bit modulus (which is the current security standard). Therefore, one immediate open question is whether or not such proofs can be made more efficient.

Another important open area is to find new instantiations of the Groth-Sahai proof system. There are many examples of groups which induce a bilinear map beyond those used in the three instantiations given by Groth and Sahai, and it would be interesting to see if any improvements in efficiency could be made by using such groups. It would also be interesting to see what sorts of cryptographic assumptions are necessary to use different instantiations.

Acknowledgments

Primary thanks go to Anna Lysyanskaya, who suggested this really fascinating topic and was extremely helpful and patient over the course of the year. Thanks also go to Joe Silverman and Jeff Hoffstein for helping to work out some problems and for bringing a mathematical perspective to this work. Finally, continuing thanks to all my family and friends for their invaluable moral support.

Bibliography

[BB04] D. Boneh and X. Boyen. Short signatures without random oracles. In *EUROCRYPT '04*, volume 3027 of *Lecture Notes in Computer Science*, pages 54–73. Springer-Verlag, 2004.

[BBS04] D. Boneh, X. Boyen, and H. Shacham. Short group signatures. In *CRYPTO '04*, volume 3152 of *Lecture Notes in Computer Science*, pages 41–55. Springer-Verlag, 2004.

[BCKL08] M. Belenkiy, M. Chase, M. Kohlweiss, and A. Lysyanskaya. Non-interactive anonymous credentials. In *Proc. 5th Theory of Cryptography Conference (TCC)*, pages 356–374, 2008.

[BdSMP91] M. Blum, A. de Santis, S. Micali, and G. Persiano. Non-interactive zero-knowledge. *SIAM Journal of Computing*, 20(6):1084–1118, 1991.

[BFM88] M. Blum, P. Feldman, and S. Micali. Non-interactive zero-knowledge and its applications. In *Proc. 20th Symposium on Theory of Computing (STOC)*, pages 103–112, 1988.

[BGN05] D. Boneh, E. Goh, and K. Nissim. Evaluating 2-dnf formulas on ciphertexts. In *Proc. 2nd Theory of Cryptography Conference (TCC)*, volume 3378 of *Lecture Notes in Computer Science*, pages 325–341. Springer-Verlag, 2005.

[BW07] X. Boyen and B. Waters. Full-domain subgroup hiding and constant-size group signatures. In *Public Key Cryptography*, pages 1–15, 2007.

[Dam92] I. Damgård. Non-interactive circuit based proofs and non-interactive perfect zero-knowledge with preprocessing. In *EUROCRYPT '92*, volume 658 of *Lecture Notes in Computer Science*, page 341355. Springer-Verlag, 1992.

[DBS04] R. Dutta, R. Barua, and P. Sarkar. Pairing-based cryptographic protocols: a survey. `http://eprint.iacr.org/2004/064`, 2004.

[ElG84] T. ElGamal. A public-key cryptosystem and a signature scheme based on discrete logarithms. In *CRYPTO '84*, pages 10–18, 1984.

[FLS90] U. Feige, D. Lapidot, and A. Shamir. Multiple non-interactive zero-knowledge proofs based on a single random string. In *Proc. 31st Symposium on Theory of Computing (STOC)*, pages 308–317, 1990.

[GL07] J. Groth and S. Lu. A non-interactive shuffle with pairing-based verifiability. In *ASIACRYPT '07*, volume 4833 of *Lecture Notes in Computer Science*, pages 51–67. Springer-Verlag, 2007.

[GMR85] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. In *17th STOC*, pages 186–208, 1985.

[Gol01] O. Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, Cambridge, 2001.

[GOS06] J. Groth, R. Ostrovsky, and A. Sahai. Perfect non-interactive zero-knowledge for np. In *EUROCRYPT '06*, volume 4004 of *Lecture Notes in Computer Science*, pages 339–358. Springer-Verlag, 2006.

[Gro07] J. Groth. Fully anonymous group signatures without random oracles. In *ASIACRYPT '07*, volume 4833 of *Lecture Notes in Computer Science*, pages 164–180. Springer-Verlag, 2007.

[GS08] J. Groth and A. Sahai. Efficient non-interactive proof systems for bilinear groups. In *EUROCRYPT '08*, volume 4965 of *Lecture Notes in Computer Science*, pages 415–432. Springer-Verlag, 2008.

[JS07] S. Jarecki and V. Shmatikov. Efficient two-party secure computation on committed inputs. In *EUROCRYPT '07*, volume 4515 of *Lecture Notes in Computer Science*, pages 143–154. Springer-Verlag, 2007.

[KP98] J. Kilian and E. Petrank. An efficient non-interactive zero-knowledge proof system for np with general assumptions. *Journal of Cryptology*, pages 1–27, 1998.

[Mei08] S. Meiklejohn. Concurrent zero-knowledge proofs. Sc. B. thesis, Brown University, Providence, Rhode Island, 2008.

[NR97] M. Naor and O. Reingold. On the construction of pseudo-random permuations: Luby-rackoff revisited. In *Proc. 29th Symposium on Theory of Computing (STOC)*, pages 189–199, 1997.

[Sco02] M. Scott. Authenticated id-based key exchange and remote log-in with simple token and pin number. `http://eprint.iacr.org/2002/164`, 2002.

[Ver04] E. Verheul. Evidence that xtr is more secure than supersingular elliptic curve cryptosystems. *Journal of Cryptology*, 17(4):277–296, 2004.