P-signatures and Noninteractive Anonymous Credentials

Mira Belenkiy[1], Melissa Chase[1], Markulf Kohlweiss[2], and Anna Lysyanskaya[1]

[1] Brown University
{mira, melissa, anna}@cs.brown.edu
[2] KU Leuven
mkohlwei@esat.kuleuven.be

Abstract. In this paper, we introduce P-signatures. A P-signature scheme consists of a signature scheme, a commitment scheme, and (1) an interactive protocol for obtaining a signature on a committed value; (2) a *non-interactive* proof system for proving that the contents of a commitment has been signed; (3) a non-interactive proof system for proving that a pair of commitments are commitments to the same value. We give a definition of security for P-signatures and show how they can be realized under appropriate assumptions about groups with a bilinear map. We make extensive use of the powerful suite of non-interactive proof techniques due to Groth and Sahai. Our P-signatures enable, for the first time, the design of a practical non-interactive anonymous credential system whose security does not rely on the random oracle model. In addition, they may serve as a useful building block for other privacy-preserving authentication mechanisms.

1 Introduction

Anonymous credentials [Cha85,Dam90,Bra99,LRSW99,CL01,CL02,CL04] let Alice prove to Bob that Carol has given her a certificate. Anonymity means that Bob and Carol cannot link Alice's request for a certificate to Alice's proof that she possesses a certificate. In addition, if Alice proves possession of a certificate multiple times, these proofs cannot be linked to each other. Anonymous credentials are an example of a privacy-preserving authentication mechanism, which is an important theme in modern cryptographic research. Other examples are electronic cash [CFN90,CP93,Bra93,CHL05] and group signatures [CvH91,CS97,ACJT00,BBS04,BW06,BW07]. In a series of papers, Camenisch and Lysyanskaya [CL01,CL02,CL04] identified a key building block commonly called "a CL-signature" that is frequently used in these constructions. A CL-signature is a signature scheme with a pair of useful protocols.

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

The second protocol, called *Prove*, is a zero-knowledge proof of knowledge of a signature on a committed value. The prover has a message-signature pair $(x, \sigma_{pk}(x))$.

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

It is clear that using general secure two-party computation [Yao86] and zero-knowledge proofs of knowledge of a witness for any NP statement [GMW86], we can construct the Issue and Prove protocols from any signature scheme and commitment scheme. Camenisch and Lysyanskaya's contribution was to construct specially designed signature schemes that, combined with Pedersen [Ped92] and Fujisaki-Okamoto [FO98] commitments, allowed them to construct Issue and Prove protocols that are efficient enough for use in practice. In turn, CL-signatures have been implemented and standardized [CVH02,BCC04]. They have also been used as a building block in many other constructions [JS04,BCL04,CHL05,DDP06,CHK+06,TS06].

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

There are two known techniques for making the CL Prove protocols non-interactive. We can use the Fiat-Shamir heuristic [FS87], which requires the random-oracle model. A series of papers [CGH04,DNRS03,GK03] show that proofs of security in the random-oracle model do not imply security. The other option is to use general techniques: [BFM88,DSMP88,BDMP91] show how any statement in NP can be proven in non-interactive zero-knowledge. This option is prohibitively expensive.

We give the first *practical* non-interactive zero-knowledge proof of knowledge of a signature on a committed message. We have two constructions using two different practical siganture schemes and a special class of commitments due to Groth and Sahai [GS07]. Our constructions are secure in the common reference string model.

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

OUR CONTRIBUTIONS. Our main contribution is the formal definition of a P-signature scheme and two efficient constructions.

Anonymous credentials are an immediate consequence of P-signatures (and of CL-signatures [Lys02]). Let us explain why (see full paper for an in-depth treatment). Suppose there is a public-key infrastructure that lets each user register a public key. Alice registers unlinkable pseudonyms A_B and A_C with Bob and Carol. A_B and A_C are commitments to her secret key, and so they are unlinkable by the security properties of the commitment scheme. Suppose Alice wishes to obtain a certificate from Carol and show it to Bob. Alice goes to Carol and identifies herself as the owner of pseudonym A_C. They run the P-signature Issue protocol as a result of which Alice gets Carol's signature on her secret key. Now Alice uses the P-signature Prove protocol to construct a non-interactive proof that she has Carol's signature on the opening of A_B.

Our techniques may be of independent interest. Typically, a proof of knowledge π of a witness x to a statement s implies that there exists an efficient algorithm that can extract a value x' from π such that x' satisfies the statement s. Our work uses Groth-Sahai non-interactive proofs of knowledge [GS07] from which we can only extract $f(x)$ where f is a one-way function. We formalize the notion of an f-extractable proof of knowledge and develop useful notation for describing f-extractable proofs that committed values have certain properties. Our notation has helped us understand how to work with the GS proof system and it may encourage others to use the wealth of this powerful building block.

TECHNICAL ROADMAP. We use Groth and Sahai's f-extractable non-interactive proofs of knowledge [GS07] to build P-signatures. Groth and Sahai give three instantiations for their proof system, using SXDH, DLIN, and SDA assumptions. We can use either of the first two instantiations. The SDA-based instantiation does not give us the necessary extraction properties.

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

Our first construction uses the Weak Boneh-Boyen (WBB) signature scheme [BB04]. Using a rather strong assumption, we prove that WBB is F-unforgeable and our P-signature construction is secure. Our second construction uses a better assumption (because it is falsfiable [Nao03]) and Our construction is based on the Full Boneh-Boyen signature scheme [BB04]. We had to modify the Boneh-Boyen construction, however, because the GS proof system would not allow the knowledge extraction of the entire signature. Our first construction is much simpler, but, as it's security relies on an interactive and thus much stronger assumption, we have decided to focus here on our second construction. For details on the first construction, see the full version.

ORGANIZATION. Sections 2 and 3 define P-signatures and introduce complexity assumptions. Section 4 explains non-interactive proofs of knowledge, introduces our new notation, and reviews GS proofs. Finally, Section 5 contains our second construction.

2 Definition of a Secure P-Signature Scheme

We say that a function $\nu : \mathbb{Z} \rightarrow \mathbb{R}$ is negligible if for all integers c there exists an integer K such that $\forall k > K$, $|\nu(k)| < 1/k^c$. We use the standard GMR [GMR88] notation to describe probability spaces.

Here we introduce P-signatures a primitive which lets a user (1) obtain a signature on a committed message without revealing the message, (2) construct a non-interactive *zero-knowledge proof of knowledge* of $(F(m), \sigma)$ such that $\mathsf{VerifySig}(pk, m, \sigma) = \mathsf{accept}$ and m is committed to in a commitment $comm$, and (3) a non-interactive method for proving that a pair of commitments are to the same value. In this section, we give the first formal definition of a non-interactive P-signature scheme. We begin by reviewing digital signatures and introducing the concept of F-unforgeability.

2.1 Digital Signatures

A signature scheme consists of four algorithms: SigSetup, Keygen, Sign, and VerifySig. $\mathsf{SigSetup}(1^k)$ generates public parameters $params_{Sig}$. $\mathsf{Keygen}(params_{Sig})$ generates signing keys (pk, sk). $\mathsf{Sign}(params_{Sig}, sk, m)$ computes a signature σ on m. VerifySig $(params_{Sig}, pk, m, \sigma)$ outputs accept if σ is a valid signature on m, reject if not.

The standard definition of a secure signature scheme [GMR88] states that no adversary can output (m, σ), where σ is a signature on m, without first previously obtaining a signature on m . This is insufficient for our purposes. Our P-Signature constructions prove that we know some value $y = F(m)$ (for an efficiently computable bijection F) and a signature σ such that $\mathsf{VerifySig}(params_{Sig}, pk, m, \sigma) = \mathsf{accept}$. However, even if an adversary cannot output (m, σ) without first obtaining a signature on m, he might be able to output $(F(m), \sigma)$. Therefore, we introduce the notion of F-Unforgeability:

Definition 1 (F-Secure Signature Scheme). *We say that a signature scheme is F-secure (against adaptive chosen message attacks) if it is* Correct *and* F-Unforgeable.

Correct. VerifySig always accepts a signature obtained using the Sign algorithm.

F-Unforgeable. Let F be an efficiently computable bijection. No adversary should be able to output $(F(m), \sigma)$ unless he has previously obtained a signature on m. Formally, for every PPTM adversary $\mathcal{A}$, there exists a negligible function ν such that

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

$\mathcal{O}_{\mathsf{Sign}}(params_{Sig}, sk, m)$ records m-queries on Q_{Sign} and returns $\mathsf{Sign}(params_{Sig}, sk, m)$. $F(Q_{\mathsf{Sign}})$ evaluates F on all values on Q_{Sign}.

Lemma 1. *F-unforgeable signatures are secure in the standard [GMR88] sense.*

Proof sketch. Suppose an adversary can compute a forgery (m, σ). Now the reduction can use it to compute $(F(m), \sigma)$.

2.2 Commitment Schemes

Recall the standard definition of a non-interactive commitment scheme. It consists of algorithms ComSetup, Commit. $\mathsf{ComSetup}(1^k)$ outputs public parameters $params_{Com}$

for the commitment scheme. $\mathsf{Commit}(params_{Com}, x, open)$ is a deterministic function that outputs $comm$, a commitment to x using auxiliary information $open$. We need commitment schemes that are *perfectly binding* and *strongly computationally hiding*:

Perfectly Binding. For every bitstring $comm$, there exists at most one value x such that there exists opening information $open$ so that $comm = \mathsf{Commit}(params, x, open)$. We also require that it be easy to identify the bitstrings $comm$ for which there exists such an x.

Strongly Computationally Hiding. There exists an alternate setup $\mathsf{HidingSetup}(1^k)$ that outputs parameters (computationally indistinguishable from the output of $\mathsf{ComSetup}(1^k)$) so that the commitments become information-theoretically hiding.

2.3 Non-Interactive P-Signatures

A non-interactive P-signature scheme extends a signature scheme (Setup, Keygen, Sign, VerifySig) and a non-interactive commitment scheme (Setup, Commit). It consists of the following algorithms (Setup, Keygen, Sign, VerifySig, Commit, ObtainSig, IssueSig, Prove, VerifyProof, EqCommProve, VerEqComm).

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

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

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

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

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

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

Definition 2 (Secure P-Signature Scheme). *Let F be a efficiently computable bijection (possibly parameterized by public parameters). A P-signature scheme is secure if* (Setup, Keygen, Sign, VerifySig) *form an F-unforgeable signature scheme, if* (Setup, Commit) *is a perfectly binding, strongly computationally hiding commitment scheme, if* (Setup, EqCommProve, VerEqComm) *is a non-interactive proof system, and if the* Signer privacy*,* User privacy*,* Correctness*,* Unforgeability*, and* Zero-knowledge *properties hold:*

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

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

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

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

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

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

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

Here again $\mathsf{SimObtain}$ is allowed to rewind the adversary.

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

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

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

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

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

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

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

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

3 Preliminaries

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

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

We introduce a new assumption which we call TDH and review the HSDH assumption introduced by Boyen and Waters [BW07]. Groth-Sahai proofs use either the DLIN [BBS04] or SXDH [Sco02] assumption. For formal definitions, see the full version.

Definition 3 (Triple DH (TDH)). *On input $g, g^x, g^y, h, h^x, \{c_i, g^{1/(x+c_i)}\}_{i=1\ldots q}$, it is computationally infeasible to output a tuple $(h^{\mu x}, g^{\mu y}, g^{\mu x y})$ for $\mu \neq 0$.*

Definition 4 (Hidden SDH [BW07]). *On input $g, g^x, u \in G_1$, $h, h^x \in G_2$ and $\{g^{1/(x+c_\ell)}, h^{c_\ell}, u^{c_\ell}\}_{\ell=1\ldots q}$, it is computationally infeasible to output a new tuple $(g^{1/(x+c)}, h^c, u^c)$.*

Definition 5 (Decisional Linear Assumption (DLIN)). *On input $u, v, w, u^r, v^s \leftarrow G_1$ it is computationally infeasible to distinguish $z_0 \leftarrow w^{r+s}$ from $z_1 \leftarrow G_1$. The assumption is analogously defined for G_2.*

Definition 6 (Symmetric External Diffie-Hellman Assumption (SXDH)). *SXDH states that the Decisional Diffie Hellman problem is hard in both G_1 and G_2. This precludes efficient isomorphisms between these two groups.*

4 Non-Interactive Proofs of Knowledge

Our P-signature constructions use the Groth and Sahai [GS07] non-interactive proof of knowledge (NIPK) system. De Santis et al. [DDP00] give the standard definition of NIPK systems. Their definition does not fully cover the Groth and Sahai proof system. In this section, we review the standard notion of NIPK. Then we give a useful generalization, which we call an f-extractable NIPK, where the extractor only extracts a function of the witness. We develop useful notation for expressing f-extractable NIPK systems, and explain how this notation applies to the Groth-Sahai construction. We then review Groth-Sahai commitments and pairing product equation proofs. Finally, we show how they can be used to prove statments about committed exponents, as this will be necessary later for our constructions.

4.1 Proofs of Knowledge: Notation and Definitions

In this subsection, we review the definition of NIPK, introduce the notion of f-extractability, and develop some useful notation. We review the De Santis et al. [DDP00] definition of NIPK. Let $L = \{s : \exists x \text{ s.t. } M_L(s, x) = \mathsf{accept}\}$ be a language in NP and M_L a polynomial-time Turing Machine that verifies that x is a valid witness for the statement $s \in L$. A NIPK system consists of three algorithms: (1) $\mathsf{PKSetup}(1^k)$ sets up the common parameters $params_{PK}$; (2) $\mathsf{PKProve}(params_{PK}, s, x)$ computes a proof π of the statement $s \in L$ using witness x; (3) $\mathsf{PKVerify}(params_{PK}, s, \pi)$ verifies correctness of π. The system must be *complete* and *extractable*. Completeness means that for all values of $params_{PK}$ and for all s, x such that $M_L(s, x) = \mathsf{accept}$, a proof π generated by $\mathsf{PKProve}(params_{PK}, s, x)$ must be accepted by $\mathsf{PKVerify}(params_{PK}, s, \pi)$. Extractability means that there exists a polynomial-time extractor $(\mathsf{PKExtractSetup}, \mathsf{PKExtract})$. $\mathsf{PKExtractSetup}(1^k)$ outputs $(td, params_{PK})$ where $params_{PK}$ is distributed identically to the output of $\mathsf{PKSetup}(1^k)$. For all PPT adversaries $\mathcal{A}$, the probability that $\mathcal{A}(1^k, params_{PK})$ outputs (s, π) such that $\mathsf{PKVerify}(params_{PK}, s, \pi) = \mathsf{accept}$ and $\mathsf{PKExtract}(td, s, \pi)$ fails to extract a witness x such that $M_L(s, x) = \mathsf{accept}$ is negligible in k. We have *perfect* extractability if this probability is 0.

We first generalize the notion of NIPK for a language L to languages parameterized by $params_{PK}$ – we allow the Turing machine M_L to receive $params_{PK}$ as a separate input. Next, we generalize extractability to f-extractability. We say that a NIPK system is f-extractable if $\mathsf{PKExtract}$ outputs y, such that there $\exists x : M_L(params_{PK}, s, x) = \mathsf{accept} \wedge y = f(params_{PK}, x)$. If $f(params_{PK}, \cdot)$ is the identity function, we get the usual notion of extractability. We denote an f-extractable proof π obtained by running $\mathsf{PKProve}(params_{PK}, s, x)$ as

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

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

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

We introduce shorthand notation for the above expression: $\pi \leftarrow \mathsf{NIPK}\{((c_1 : x_1), \ldots, (c_n : x_n)) : \mathsf{Condition}(x)\}$. For simplicity, we assume the proof π includes s.

4.2 Groth-Sahai Commitments [GS07]

We review the Groth-Sahai [GS07] commitment scheme. We use their scheme to commit to elements of a group G of prime order p. Technically, their constructions commit to elements of certain modules, but we can apply them to certain bilinear groups elements. Groth and Sahai also have a construction for composite order groups using the Subgroup Decision assumption; however it lacks the necessary extraction properties.

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

$\mathsf{GSCommit}(params_{Com}, x, open)$. Takes as input $x \in G$ and some value $open$ and outputs a commitment $comm$. The extension $\mathsf{GSExpCommit}(params_{Com}, b, \theta, open)$ takes as input $\theta \in Z_p$ and a base $b \in G$ and outputs $(b, comm)$, where $comm = \mathsf{GSCommit}(params_{Com}, b^{\theta}, open)$. (Groth and Sahai compute commitments to elements in Z_p slightly differently;

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

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

GS commitments are *perfectly binding*, *strongly computationally hiding*, and *extractable*. Groth and Sahai [GS07] show how to instantiate commitments that meet these requirements using either the SXDH or DLIN assumptions. Commitments based on SXDH consist of 2 elements in G, while those based on DLIN setting require 3 elements in G. Note that in the Groth-Sahai proof system below, $G = G_1$ or $G = G_2$ for SXDH and $G = G_1 = G_2$ for DLIN.

4.3 Groth-Sahai Pairing Product Equation Proofs [GS07]

Groth and Sahai [GS07] construct an f-extractable NIPK system that lets us prove statements in the context of groups with bilinear maps.

$\mathsf{GSSetup}(1^k)$ outputs $(p, G_1, G_2, G_T, e, g, h)$, where G_1, G_2, G_T are groups of prime order p, with g a generator of G_1, h a generator of G_2, and $e : G_1 \times G_2 \to G_T$ a cryptographic bilinear map. $\mathsf{GSSetup}(1^k)$ also outputs $params_1$ and $params_2$ for constructing GS commitments in G_1 and G_2, respectively. (If the pairing is symmetric, $G_1 = G_2$ and $params_1 = params_2$.) The statement s to be proven consists of the following list of values: $\{a_q\}_{q=1\ldots Q} \in G_1$, $\{b_q\}_{q=1\ldots Q} \in G_2$, $t \in G_T$, and $\{\alpha_{q,m}\}_{m=1\ldots M, q=1\ldots Q}, \{\beta_{q,n}\}_{n=1\ldots N, q=1\ldots Q} \in Z_p$, as well as a list of commitments $\{c_m\}_{m=1\ldots M}$ to values in G_1 and $\{d_n\}_{n=1\ldots N}$ to values in G_2. Groth and Sahai show how to construct the following proof:

$$\mathsf{NIPK}\{((c_1 : x_1), \ldots, (c_M : x_M), (d_1 : y_1), \ldots, (d_N : y_N)) : \\ \prod_{q=1}^{Q} e(a_q \prod_{m=1}^{M} x_m^{\alpha_{q,m}}, b_q \prod_{n=1}^{N} y_n^{\beta_{q,n}}) = t\}$$

The proof π includes the statement being proven; this includes the commitments $c_1, \ldots, c_M$ and $d_1, \ldots, d_N$. Groth and Sahai provide an efficient extractor that opens these commitments to values $x_1, \ldots, x_M, y_1, \ldots, y_N$ that satisfy the pairing product equation.

Recall the function $\mathsf{GSExpCommit}(params_1, b, \theta, open) = (b, \mathsf{GSCommit}(params_1, b^\theta, open))$. We can replace any of the clauses $(c_m : x_m)$ with the clause $(c_m : b^\theta)$, and add b to the list of values included in the statement s (and therefore in the proof π). The same holds for commitments d_n. Groth-Sahai proofs also allow us to prove that the openings of $(c_1, \ldots, c_n, d_1, \ldots, d_n)$ satisfy several equations *simultaneously*.

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

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

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

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

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

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

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

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

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

Strong Witness Indistinguishability. A simulator $\mathsf{Sim} = (\mathsf{SimSetup}, \mathsf{SimProve})$ with the following two properties exists: (1) $\mathsf{SimSetup}(params_{BM})$ outputs $params_{GS}'$ such that they are computationally indistinguishable from the output of $\mathsf{GSSetup}(params_{BM})$. Let $params_1' \in params_{GS}'$ be the parameters for the commitment scheme in G_1. Using $params_1'$, commitments are perfectly hiding – this means that for all commitments $comm, \forall x \in G_1, \exists open : \mathsf{VerifyOpening}(params_1', comm, x, open) = \mathsf{accept}$ (analogous for G_2). (2) Using the $params_{GS}'$ generated by the challenger, GS proofs become perfectly witness indistinguishable. Suppose an unbounded adversary $\mathcal{A}$ generates a statement s consisting of the pairing product equations and a set of commitments $(c_1, \ldots, c_M, d_1, \ldots, d_N)$. The adversary opens the commitments in two different ways $W_0 = (x_1^{(0)}, \ldots, x_M^{(0)}, y_1^{(0)}, \ldots, y_N^{(0)}, \mathit{openings}_0)$ and $W_1 = (x_1^{(1)}, \ldots, x_M^{(1)}, y_1^{(1)}, \ldots, y_N^{(1)}, \mathit{openings}_1)$ (under the requirement that these witnesses must both satisfy s). The values $\mathit{openings}_b$ show how to open the commitments to $\{x_m^{(b)}, y_n^{(b)}\}$. (The adversary can do this because it is unbounded.) The challenger gets the statement s and the two witnesses W_0 and W_1. He chooses a bit $b \leftarrow \{0,1\}$ and computes $\pi = \mathsf{GSProve}(\mathit{params}_{GS}', s, W_b)$. Strong witness indistinguishability means that π is distributed independently of b.

Composable Zero-Knowledge. Note that Groth and Sahai show that if in a given pairing product equation the constant t can be written as $t = e(t_1, t_2)$ for known t_1, t_2, then these proofs can be done in zero knowledge. However, their zero knowldge proof construction is significantly less efficient than the WI proofs. Thus, we choose to use only the WI construction as a building block. Then we can take advantage of special features of our P-signature construction to create much more efficient proofs that still have the desired zero knowledge properties. The only exception is our construction for EqCommProve, which does use the zero knowledge technique suggested by Groth and Sahai.

4.4 Proofs about Committed Exponents

We use the Groth-Sahai proof system to prove equality of committed exponents.

Equality of Committed Exponents in Different Groups. We want to prove the statement $\mathsf{NIPK}\{((c : g^\alpha), (d : h^\beta)) : \alpha = \beta\}$. We perform a Groth-Sahai pairing product equation proof $\mathsf{NIPK}\{((c : x), (d : y)) : e(x, h)e(1/g, y) = 1\}$. Security is straightforward due to the f-extractability property of the GS proof system.

Equality of Committed Exponents in the Same Group. We want to prove the statement $\mathsf{NIPK}\{((c_1 : g^\alpha), (c_2 : u^\beta)) : \alpha = \beta\}$, where $g, u \in G_1$. This is equivalent to proving $\mathsf{NIPK}\{((c_1 : g^\alpha), (c_2 : u^\beta), (d : h^\gamma) : \alpha = \gamma \wedge \beta = \gamma\}$.

Zero-Knowledge Proof of Equality of Committed Exponents. We want to prove the statement $\mathsf{NIZKPK}\{((c_1 : g^\alpha), (c_2 : g^\beta) : \alpha = \beta\}$ in zero-knowledge. We perform the Groth-Sahai *zero-knowledge* pairing product equation proof $\mathsf{NIPK}\{((c_1 : g^\alpha), (c_2 : g^\beta), (d : h^\theta) : e(a/b, h^\theta) = 1 \wedge e(g, h^\theta)e(1/g, h) = 1\}$. Proof of equality of committed exponents in group G_2 is done analogously. See full version for details.

Remark 1. We cannot directly use Groth-Sahai general arithmetic gates [GS07] to construct the above proofs because they assume that the commitments use the same base.

5 Efficient Construction of P-Signature Scheme

In this section, we present a new signature scheme and then build a P-signature scheme from it. The new signature scheme is inspired by the full Boneh-Boyen signature scheme, and is as follows:

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

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

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

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

Theorem 1. *Let $F(x) = (h^x, u^x)$, where $u \in G_1$ and $h \in G_2$ as in the HSDH and TDH assumptions. Our new signature scheme is F-secure given HSDH and TDH. (*See full version for proof.*)*

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

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

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

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

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

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

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

1. The user chooses $\rho_1, \rho_2 \leftarrow Z_p$.
2. The issuer chooses $r' \leftarrow Z_p$.
3. The user and the issuer run a secure two-party computation protocol where the user's private inputs are $(\rho_1, \rho_2, m, open)$, and the issuer's private inputs are $sk = (\alpha, \beta)$ and r'.
   The issuer's private output is $x = (\alpha + m + \beta\rho_1 r')\rho_2$ if $comm = \mathsf{Commit}(params, m, open)$, and $x = \perp$ otherwise.
4. If $x \neq \perp$, the issuer calculates $C_1' = g^{1/x}$, $C_2' = w^{r'}$ and $C_3' = u^{r'}$, and sends (C_1', C_2', C_3') to the user.
5. The user computes $C_1 = C_1'^{\rho_2}$, $C_2 = C_2'^{\rho_1}$, and $C_3 = C_3'^{\rho_1}$ and then verifies that the signature (C_1, C_2, C_3) is valid.

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

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

$$\pi = \mathsf{NIPK}\{((\Sigma : C_1), (R_w : C_2), (R_u : C_3)(M_h : h^\alpha), (M_u : u^\beta)) : \\ e(C_1, vh^\alpha C_2) = z \wedge e(u, C_2) = e(C_3, w) \wedge \alpha = \beta\}.$$

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

$\mathsf{EqCommProve}(params, m, open, open')$ Let commitment $comm = \mathsf{Commit}(params, m, open) = \mathsf{GSCommit}(params_2, h^m, open)$ and $comm' = \mathsf{Commit}(params, m, open') = \mathsf{GSCommit}(params_2, h^m, open')$. Use the GS proof system as described in Section 4.4 to compute $\pi \leftarrow \mathsf{NIZKPK}\{((comm : h^\alpha), (comm' : h^\beta) : \alpha = \beta\}$.

$\mathsf{VerEqComm}(params, comm, comm', \pi)$ Verify the proof π using the GS proof system as described in Section 4.4.

Theorem 2 (Efficiency). *Using SXDH GS proofs, each P-signature proof for our new signature scheme consists of 18 elements in G_1 and 16 elements in G_2. The prover performs 34 multi-exponentiation and the verifier 68 pairings. Using DLIN, each P-signature proof consists of* 42 *elements in $G_1 = G_2$. The prover has to do* 42 *multi-exponentiations and the verifier* 84 *pairings.*

Theorem 3 (Security). *Our second P-signature construction is secure given HSDH and TDH and the security of the GS commitments and proofs.*

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

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

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

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

Finally, we need to show that we can simulate proofs of $\mathsf{EqCommProve}$ given the trapdoor sim_{GS}. This follows directly from composable zero knowledge of $\mathsf{EqCommProve}$. See full version for details.

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

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

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

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

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

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

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

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

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

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

Acknowledgments: Mira Belenkiy, Melissa Chase and Anna Lysyanskaya are supported by NSF grants CNS-0374661 CNS-0627553. Markulf Kohlweiss is supported by the European Commission's IST Program under Contracts IST-2002-507591 PRIME and IST-2002-507932 ECRYPT.

References

[ACJT00] G. Ateniese, J. Camenisch, M. Joye, and G. Tsudik. A practical and provably secure coalition-resistant group signature scheme. In *CRYPTO '00*, p. 255–270.

[BB04] D. Boneh and X. Boyen. Short signatures without random oracles. In *Eurocrypt '04*, p. 54–73.

[BBS04] D. Boneh, X. Boyen, and H. Shacham. Short group signatures using strong Diffie-Hellman. In *CRYPTO '04*, p. 41–55.

[BCC04] E. Brickell, J. Camenisch, and L. Chen. Direct anonymous attestation. Technical Report Research Report RZ 3450, IBM Research Division, March 2004.

[BCL04] E. Bangerter, J. Camenisch, A. Lysyanskaya. A cryptographic framework for the controlled release of certified data. In *Cambridge Security Protocols Workshop '04*.

[BDMP91] M. Blum, A. De Santis, S. Micali, and G. Persiano. Non-interactive zero-knowledge. *SIAM J. of Computing*, 20(6):1084–1118, 1991.

[BFM88] M. Blum, P. Feldman, and S. Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *STOC '88*, p. 103–112.

[Bra93] S. Brands. An efficient off-line electronic cash system based on the representation problem. Technical Report CS-R9323, CWI, April 1993.

[Bra99] S. Brands. *Rethinking Public Key Infrastructure and Digital Certificates— Building in Privacy*. PhD thesis, Eindhoven Inst. of Tech. The Netherlands, 1999.

[BW06] X. Boyen and B. Waters. Compact group signatures without random oracles. In *Eurocrypt '06*, p. 427–444.

[BW07] X. Boyen and B. Waters. Full-domain subgroup hiding and constant-size group signatures. In *PKC '07*, p. 1–15.

[CFN90] D. Chaum, A. Fiat, and M. Naor. Untraceable electronic cash. In *CRYPTO '90*, p. 319–327.

[CGH04] R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004.

[Cha85] D. Chaum. Security without identification: Transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10):1030–1044, October 1985.

[CHK$^+$06] J. Camenisch, S. Hohenberger, M. Kohlweiss, A. Lysyanskaya, and M. Meyerovich. How to win the clonewars: efficient periodic n-times anonymous authentication. In *CCS '06*, p. 201–210.

[CHL05] J. Camenisch, S. Hohenberger, and A. Lysyanskaya. Compact E-Cash. In *Eurocrypt '05*, p. 302–321.

[CL01] J. Camenisch and A. Lysyanskaya. Efficient non-transferable anonymous multi-show credential system with optional anonymity revocation. In *Eurocrypt '01*, p. 93–118.

[CL02] J. Camenisch and A. Lysyanskaya. A signature scheme with efficient protocols. In *SCN '02*, p. 268–289.

[CL04] J. Camenisch and A. Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *CRYPTO '04*, p. 56–72.

[CLM07] J. Camenisch, A. Lysyanskaya, and M. Meyerovich. Endorsed e-cash. In *IEEE Symposium on Security and Privacy '07*, p. 101–115.

[CP93] D. Chaum and T. Pedersen. Transferred cash grows in size. In *Eurocrypt '92*, p. 390–407.

[CS97] J. Camenisch and M. Stadler. Efficient group signature schemes for large groups. In *CRYPTO '97*, p. 410–424.

[CvH91] D. Chaum and E. van Heyst. Group signatures. In *Eurocrypt '91*, p. 257–265.

[CVH02] J. Camenisch and E. Van Herreweghen. Design and implementation of the *idemix* anonymous credential system. In *Proc. 9th ACM CCS '02*, p. 21–30..

[Dam90] I. Damgård. Payment systems and credential mechanism with provable security against abuse by individuals. In *CRYPTO '88*, p. 328–335.

[DDP00] A. De Santis, G. Di Crescenzo, and G. Persiano. Necessary and sufficient assumptions for non-interactive zero-knowledge proofs of knowledge for all NP relations. In *ICALP '00*, p. 451–462.

[DDP06] I. Damgård, K. Dupont, and M. Pedersen. Unclonable group identification. In *Eurocrypt '06*, p. 555–572.

[DNRS03] C. Dwork, M. Naor, O. Reingold, and L. J. Stockmeyer. Magic functions. *J. ACM*, 50(6):852–921, 2003.

[DSMP88] A. De Santis, S. Micali, and G. Persiano. Non-interactive zero-knowledge proof systems. In *CRYPTO '87*, p. 52–72.

[FO98] E. Fujisaki and T. Okamoto. A practical and provably secure scheme for publicly verifiable secret sharing and its applications. In *Eurocrypt '98*, p. 32–46.

[FS87] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO '86*, p. 186–194.

[GK03] S. Goldwasser and Y. Kalai. On the (in)security of the Fiat-Shamir paradigm. In *FOCS '03*, p. 102–115.

[GMR88] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. on Computing*, 17(2):281–308, April 1988.

[GMW86] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity and a method of cryptographic protocol design. In *FOCS '86*, p. 174–187.

[GS07] J. Groth and A. Sahai. Efficient non-interactive proof systems for bilinear groups. `http://eprint.iacr.org/2007/155`.

[JS04] S. Jarecki and V. Shmatikov. Handcuffing big brother: an abuse-resilient transaction escrow scheme. In *Eurocrypt '04*, p. 590–608.

[LRSW99] A. Lysyanskaya, R. Rivest, A. Sahai, and S. Wolf. Pseudonym systems. In H. Heys and C. Adams, eds., *Selected Areas in Cryptography*, volume 1758 of *LNCS*, 1999.

[Lys02] A. Lysyanskaya. *Signature Schemes and Applications to Cryptographic Protocol Design*. PhD thesis, MIT, Cambridge, Massachusetts, September 2002.

[Nao03] M. Naor. On cryptographic assumptions and challenges. In *CRYPTO '03*, p. 96–109.

[Ped92] T. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO '92*, p. 129–140.

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

[TFS04] I. Teranishi, J. Furukawa, and K. Sako. k-times anonymous authentication (extended abstract). In *ASIACRYPT '04*, p. 308–322.

[TS06] I. Teranishi and K. Sako. k-times anonymous authentication with a constant proving cost. In *PKC '06*, p. 525–542.

[Yao86] A. Yao. How to generate and exchange secrets. In *FOCS '86*, p. 162–167.