Ad-Hoc-Group Signatures from Hijacked Keypairs

Ben Adida[*] Susan Hohenberger[*,†] Ronald L. Rivest[*]

June 24, 2005

Abstract

Ad-hoc-group signatures enable an individual to sign on behalf of a group without requiring prior group membership setup. Such signatures are used to provide credibility – the signer must be one of the group members – combined with some degree of anonymity – the identity of the signer within the group cannot be determined. Thus, in many instances, other group members might not cooperate in the creation of such a signature. They may even wish to interfere with its creation, refusing to generate keypairs of a form that might facilitate such activity.

We present a combination of techniques for efficiently coercing any user into an ad-hoc signatory group, using only that user's public key. This public key may correspond to almost any signature or encryption scheme, as long as there exists an efficient Special Honest Verifier Zero Knowledge Proof of Knowledge protocol for the secret key, or, alternatively, a hash-and-sign algorithm for that keypair type. Our approach effectively hijacks any public key for the purpose of building an ad-hoc group signature. We also present a new proof protocol that enables, within our framework, the hijacking of Boneh-Franklin and Waters identity-based encryption keys, as well as Camenisch-Lysyanskaya signature keys.

1 Introduction

Ad-hoc-group signatures allow a signer (Alice) to produce a digital signature for a message $\mathcal{M}$ such that a verifier can confirm that $\mathcal{M}$ was signed by *someone* from the list of "ad-hoc group" members included with the signature, but nothing more. Alice specifies the ad-hoc-group (which must include herself) when she signs the message; there is no prior setup or group manager needed.

We note that the other members of the ad-hoc-group don't need to be aware of Alice's signing activities, or even to give Alice permission to include them in the ad-hoc-group. They may even disagree with or resent Alice's activity, and, most likely, they have taken no steps to facilitate Alice's use of ad-hoc-group signatures. The question asked in this paper is: under what conditions can Alice rope other parties into "participating" in her ad-hoc-group signature?

As we shall demonstrate, Alice has rather broad power to induct others into her ad-hoc group. In general, Alice can get Bob into her group if Bob is using more-or-less any kind of public-key encryption or signature scheme. Merely knowing Bob's public key enables Alice to get him to be a "co-signer." Even if Bob's public key is intended only for encryption, Alice can usually hijack his key anyway.

This paper presents a protocol compiler for handling heterogeneous key types within a single ad-hoc-group signature. As an example, we show how to handle public keys for two types of identity-based encryption schemes and one type of clasical signature scheme, all based on bilinear maps, using new proofs of knowledge of the corresponding secret keys.

[*]Computer Science and Artifical Intelligence Laboratory; Massachusetts Institute of Technology; 32 Vassar Street; Cambridge, MA 02139, USA. Email: {ben,srhohen,rivest}@mit.edu.

[†]Supported by an NDSEG Fellowship.

1.1 Motivating Applications

An ad-hoc-group signature allows Alice to gain some credibility from demonstrable group membership, while maintaining anonymity within that group. Two immediate applications arise from this technique: the leaking of secrets from a committee, and one type of construction of designated-verifier signatures.

Leaking Secrets. Imagine that Alice is a cabinet member, and that she wishes to "leak" a document $\mathcal{M}$ to the press, so that it is verifiably authentic but so that she is not obviously the source. She can do this by signing $\mathcal{M}$ with an ad-hoc-group signature, where the ad-hoc group includes all cabinet members and other senior government officials (including herself, of course).

A journalist can verify the signature and determine that $\mathcal{M}$ was, in fact, signed by some senior government official, but he cannot identify the actual signer within that group. The journalist can safely report, "A senior government official has supplied the following document, which says ..." without fear of being forced to reveal his source (because he can't).

(Depending on your political views, you may or may not approve of this application. Nonetheless, it is interesting, and was a motivation for Rivest et al.'s development of ring signatures [21].)

Designated-Verifier Signatures. When Alice sends a message $\mathcal{M}$ to Bob, she may want to prove the message's authenticity in a more private manner than by applying a publicly verifiable signature to $\mathcal{M}$. Specifically, Alice may not want Eve, a third party, to be able to verify Alice's signature on $\mathcal{M}$. This concept, introduced by Jakobsson et al [17], is called *designated-verifier signatures*: Alice effectively designates Bob as the sole verifier of her signature.

Alice can construct a type of designated-verifier signature using a carefully crafted ad-hoc group. Alice will designate a two-person ad-hoc group including herself and Bob, her intended recipient. Upon receipt of such a signature, Bob can be certain that Alice was the signer, as he knows that he himself did not produce this signature. However, Bob is unable to prove this fact to anyone else, since others may simply believe he is trying to frame Alice.

Using such a system, Alice effectively gains "repudiability:" if Bob reveals their conversation to a third party, Alice can plausibly deny any knowledge of such communication. It has been noted in the literature that such a property is quite important for maintaining the privacy of casual online conversations such as instant messaging or email [6].

1.2 Our Results

Our contributions in this paper begin with a protocol compiler for creating ad-hoc-group signatures from any collection of keypairs. We then present a new SHVZK protocol for proving bilinear map pre-images, which we use to enable Boneh-Franklin IBE [5], Waters IBE [27], and Camenisch-Lysyanskaya signature [7] keypairs within our compiler. We stress that these SHVZK protocols are highly efficient, 3-round proofs where each round involves at most one bilinear map computation. The protocol compiler is also quite efficient, as it adds comparatively little overhead to the included proofs of knowledge.

Ad-Hoc-Group Signature Protocol Compiler. Our protocol compiler borrows heavily from Cramer et al.'s proofs of partial knowledge [11] and includes a few tricks from Abe et al.'s ring signature compiler [1]. Though our construction's individual components are well understood, we believe this combination to be both useful and novel. Our compiler takes inputs:

- message $\mathcal{M}$.
- $pk_1, \ldots, pk_{i-1}, pk_{i+1}, \ldots, pk_n$, public keys, each with its own encryption or signature scheme.
- sk_i, a secret key.

- for $0 \leq j \leq n$, either Π_j, a Special Honest Verifier Zero Knowledge Proof of Knowledge protocol for proving the secret key equivalent of pk_j, or H_j, a hash-and-sign algorithm compatible with pk_j.

Proof of Knowledge of Bilinear Map Pre-Images. Consider the typical bilinear map setup, with groups G_1 and G_2 of order q and a non-degenerate, efficient, bilinear mapping function $e(\cdot, \cdot)$ from $G_1 \times G_1$ to G_2. We provide efficient SHVZK protocols for proving knowledge of a bilinear map pre-image. Given $Q \in G_2$ and $x \in G_1$, our protocol proves knowledge of α such that $e(\alpha, x) = Q$. (We also provide similar protocols for proving knowledge of *any* pre-image tuple, or of a symmetric pre-image when one exists.)

Proof of Knowledge of Bilinear-Map-based Secret Keys. Using this proof of knowledge protocol for bilinear map pre-images, we present the natural proofs of knowledge of secret keys for:

- **Boneh-Franklin [5] identity-based keypairs** where $e(pk, MPK) = e(sk, g)$. sk is the pre-image against $g \in G_1$ of the publicly computable $Q = e(pk, MPK)$.
- **Waters [27] identity-based keypairs** where the secret key is $sk = (S_1, S_2)$ and, given X a publicly computable function of the MPK, $e(S_1, g) = Xe(S_2, pk)$. Publish $A = e(S_1, g)$. S_1 is then the pre-image of A against g, and S_2 the pre-image of A/X against pk.
- **Camenisch-Lysyanskaya [7] keypairs** where $pk = (P_1, P_2)$, and $e(P_1, P_2) = e(sk, g)$. [1] sk is the pre-image against g of the publicly computable $Q = e(P_1, P_2)$.

1.3 Related Work

Our construction relies heavily on Cramer et al.'s Proofs of Partial Knowledge Protocol [11], which provides the basis for the one-out-of-many group signature structure. Additional tricks borrowed from Abe et al.'s ring signature compiler [1] provide the ability to integrate hash-and-sign algorithm keypairs as well as full separability, a concept first introduced by Camenisch and Michels [8].

While our construction is not exactly a ring signature, previous ad-hoc-group signatures have been implemented as rings, as first introduced by Rivest et al [21], and later adapted to identity-based keypairs by Zhakim et al [28]. These signatures, including the one we present, provide one implementation of *designated verifier proofs*, as introduced by Jakobsson et al. [17]. A similar cryptographic construction based on Jakobsson's work is Naor's deniable signature scheme [18], including recent extensions [25, 26, 19]. Recently, Bellare et al. [4] created weaker ad-hoc-group signatures from general assumptions, such as families of trapdoor permutations. (By weaker, we mean they do not allow for the broad key-hijacking that we address.)

Note also that our new protocol for proving knowledge of bilinear map pre-images is reminiscent of Schnorr's original discrete-log based protocol [23]. Our SHVZK proof of secret keys for Boneh-Franklin keys effectively formalizes and proves the Hess identity-based signature scheme [16]. Our signature scheme for Waters keys continues the identity-based signature work begun by Shamir [24], first implemented by Guillou and Quisquater [15], and recently summarized by Bellare et al. [2].

1.4 Roadmap

In Section 2, we formally define our goals. In Section 3, we review some technical preliminaries and notation. Our main construction is detailed in Section 4, which combines any key types that support: (1) a SHVZK proof of knowledge protocol or (2) a Hash-and-Sign algorithm, into an ad-hoc-group signature. Then, in Sections 5 and 6, we show how to achieve efficient SHVZK proof of knowledge protocols for Boneh-Franklin [5] IBE keys, Waters [27] IBE keys, and Camenisch-Lysyanskaya [7] signature keys.

[1] The secret key was originally specified as the pair $(\log_g(P_1), \log_g(P_2))$, but we observe that the above-specified secret key is equivalent.

2 Definitions

Intuitively, an ad-hoc-group signature scheme is a method by which any user with a keypair can create a signature on behalf of a group including himself and any other users with public keys. The important point is that the public keys associated with these users may have been generated according to different algorithms; for example, one user may have an RSA encryption [20] keypair while another member may have a Hess identity-based signature [16] keypair.

Notation: By $\mathsf{negl}(\cdot)$, we denote a negligible function such that for all polynomials $p(\cdot)$ and all sufficiently large integers k, $\mathsf{negl}(k) < 1/p(k)$.

Definition 2.1 (Ad-Hoc-Group Signature) *Let n, i be positive integers such that $1 \leq i \leq n$. An* ad-hoc-group signature scheme *is defined as a set of algorithms: $\{\mathsf{KeyGen}_j\}_{j\in[1,n]}$, $\mathsf{GroupSign}$, and $\mathsf{GroupVerify}$:*

- KeyGen_j *produces a public key pair (pk, sk), on input a security parameter 1^{k_j}.*
- $\mathsf{GroupSign}$ *produces a group signature σ, on input a set of public keys $\{pk_1, \ldots, pk_{i-1}, pk_{i+1}, \ldots, pk_n\}$ for $n-1$ of the group members, a secret key sk_i for the remaining group member, and an arbitrary string m.*
- $\mathsf{GroupVerify}$ *produces a verification bit (1 or 0), on input $\sigma, m, \{pk_j\}_{j\in[1,n]}$. The result is 1 if and only if σ is a valid group signature on message m for the group designated by public keys pk_j.*

Correctness. *We require that the usual property holds with probability one:*

$$\mathsf{GroupVerify}(\{pk_j\}_{j\in[1,n]}, m, \mathsf{GroupSign}(pk_1, \ldots, pk_{i-1}, pk_{i+1}, \ldots, pk_n, sk_i, m)) = 1.$$

Security. *We have two guarantees: (1)* unforgeability *and (2)* signer-ambiguity.

1. Unforgeable against adaptive chosen-message attack: *We guarantee that forgeries occur with no more than negligible probability. A forgery is defined as a group signature that passes* $\mathsf{GroupVerify}$*, but was not created by any of the group members. That is, for all positive integers n, i, where $1 \leq i \leq n$, all security parameters $1^{k_1}, \ldots, 1^{k_n}$, and all probabilistic polynomial-time (PPT) adversaries* Adv,

$$\begin{aligned}&\Pr[(pk_1, sk_1) \leftarrow \mathsf{KeyGen}_1(1^{k_1}), \ldots, (pk_n, sk_n) \leftarrow \mathsf{KeyGen}_n(1^{k_n}),\\ &\quad (m, \sigma) \leftarrow \mathsf{Adv}^{\mathsf{OGroupSign}(\cdot,\cdot)}(\{pk_j\}_{j\in[1,n]}) :\\ &\quad \mathsf{GroupVerify}(\{pk_j\}_{j\in[1,n]}, m, \sigma) = 1 \ \wedge \ m \notin Q] \leq \mathsf{negl}(\min(k_1, \ldots, k_n)),\end{aligned}$$

   where $\mathsf{OGroupSign}$ *is an oracle that takes as input a message m and an index i, and returns* $\mathsf{GroupSign}(pk_1, \ldots, pk_{i-1}, pk_{i+1}, \ldots, pk_n, sk_i, m)$*; Q is the set of messages queried to* $\mathsf{OGroupSign}$*; and $\min(k_1, \ldots, k_n)$ represents the smallest security parameter.*

2. Computational signer-ambiguity: *We guarantee that the original signer's identity is concealed even if the secret keys of all group members are revealed. That is, for all positive integers n, i, where $1 \leq i \leq n$, and all PPT adversaries* Adv,

$$\begin{aligned}&\Pr[(pk_1, sk_1) \leftarrow \mathsf{KeyGen}_1(1^{k_1}), \ldots, (pk_n, sk_n) \leftarrow \mathsf{KeyGen}_n(1^{k_n}),\\ &\quad m \leftarrow \mathsf{Adv}(\{(pk_j, sk_j)\}_{j\in[1,n]}),\end{aligned}$$

$$i \overset{R}{\leftarrow} [1,n],$$
$$\sigma \leftarrow \mathsf{GroupSign}(pk_1,\ldots,pk_{i-1},pk_{i+1},\ldots,pk_n,sk_i,m),$$
$$y \leftarrow \mathsf{Adv}(\sigma,m,\{(pk_j,sk_j)\}_{j\in[1,n]}):$$
$$y = i] \le \frac{1}{n} + \mathsf{negl}(\min(k_1,\ldots,k_n)).$$

If the above definition holds with respect to all infinitely powerful adversaries Adv*, we say that the scheme is has* unconditional *signer-ambiguity.*

In the above definition and throughout this paper, we will only be considering ad-hoc-group signatures created by a single group member. One might also imagine ad-hoc-group signatures created by a subset of the group members. Our definition and construction can be extended to those cases as well.

3 Number Theoretic Preliminaries and Notation

We briefly review some technical preliminaries which will be used from this point on in the paper.

3.1 Bilinear Maps

Let Setup be an algorithm that, on input the security parameter 1^k, outputs $\gamma = (q, g, G_1, G_2, e)$, where e is a non-degenerate, efficiently computable bilinear map from $G_1 \times G_1$ to G_2, where both G_1 and G_2 are groups of prime order $q = \Theta(2^k)$, and where g is a generator element of G_1. We assume that each group element has a unique binary representation. More formally, $e : G_1 \times G_1 \to G_2$ is a function that is:

- *Bilinear*: for all $g, h \in G_1$, for all $a, b \in \mathbb{Z}_q$, $e(g^a, h^b) = e(g,h)^{ab}$,
- *Non-degenerate*: if g is a generator of G_1, then $e(g,g)$ generates G_2, and
- *Efficient*: computing $e(g,h)$ is efficient for all $g, h \in G_1$.

By writing, $G_1 = \langle g \rangle$, we mean that g generates G_1. We recognize that, for some instantiations of the mappings, it is more efficient to let $e : G_1 \times \tilde{G} \to G_2$, where G_1 and $\tilde{G}$ are distinct groups of size q. Most of our constructions will work in this setting as well.

3.2 Notation

By writing $x \overset{R}{\leftarrow} S$, we denote that x is chosen uniformly at random from a set S. By writing $P(\mathcal{A}(x), \mathcal{B}(y))$, we denote that P is a protocol between two parties $\mathcal{A}$ and $\mathcal{B}$, where $\mathcal{A}$ takes input x and $\mathcal{B}$ takes input y.

When discussing various proofs of knowledge, we will follow the notation introduced by Camenisch and Stadler [9] for proofs of knowledge of discrete logarithms. For example,

$$PK\left\{(\alpha,\beta) : e(\alpha, g) = X \wedge h^\beta = Y\right\}$$

denotes a "*zero-knowledge proof of knowledge of values α and β such that $e(\alpha,g) = X$ and $h^\beta = Y$*". Greek letters denote values whose knowledge is being proven, while other parameters are public.

4 Our Construction

Our goal is to provide an efficient ad-hoc-group signature algorithm that can "hijack" (i.e., combine) as many different keytypes as possible. We begin this section by describing the broad classes of keytypes that are eligible for use in our algorithm. As we shall see, these keytypes must *efficiently* support at least one of two different types of protocols themselves. Then, we guarantee that the computations required to generate and verify one of our ad-hoc-group signatures, beyond running these individual protocols, is essentially negligible. Thus, the complete signature scheme is also efficient.

4.1 Eligible Keytypes

A keytype must *efficiently* support one of the following two protocols to be included in our construction: (1) SHVZK Proof of Knowledge of Secret Key given the Public Key, or (2) a Hash-and-Sign algorithm. By "efficient," we mean that, for all keypairs of that type, the running-times of all corresponding algorithms are polynomial, and, more specifically, that the computations required to either create or verify the proof of knowledge or the hash-and-sign values should be at most some small constant number (e.g., 6) of exponentiations and/or bilnear pairings. The overall litmus test is "is it fast enough to be used in practice?"

Property #1: SHVZK Proofs of Knowledge. For our purposes, a proof of knowledge is a three-round protocol between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$, where, for a given public key pk (and global information), $\mathcal{P}$ convinces $\mathcal{V}$ that he knows the corresponding secret key sk. The three rounds are called the commit t, challenge c, and the response s. The prover begins the protocol by sending t.

Loosely, we call such a scheme zero-knowledge if the verifier learns nothing in addition to being convinced that $\mathcal{P}$ knows sk. For our purposes, we will only need to consider *honest-verifier* zero-knowledge proofs of knowledge, where $\mathcal{V}$ is honest-but-curious: $\mathcal{V}$ always sends back a fresh random value for the challenge c. This makes it possible for a simulator to generate tuples of the form (t, c, s) that are indistinguishable from conversations between an honest $\mathcal{P}$ (with sk) and an honest $\mathcal{V}$.

Finally, we say that a protocol is *special honest-verifier zero-knowledge* (SHVZK) if, given fixed public parameters (including, in our case, pk), the transcript simulation can be performed by taking any challenge c and producing a valid commitment t and response s, all under the same indistinguishability constraint.

Examples: A *subset* of schemes with keytypes that support efficient SHVZK proofs of knowledge follows. (See the specific references for details on the system parameters; this is just a quick review.)

Scheme	**Keytype**	**(One) Eff. SHVZK PoK**
Waters IBE [27]	pub info $= (g, h, g^s, g^x)$ $sk = (h^s g^{xr}, g^r)$ (for a random r)	Section 5
Boneh-Franklin IBE [5] Hess IBS [16] Cha-Cheon IBS [10] Sakai-Ohgishi-Kasahara IBS [22]	pub info $= (g, g^x, g^y), sk = g^{xy}$	Hess [16], Section 5
Camenisch-Lysyanskaya Signatures [7]	$pk = (g, g^x, g^y), sk = g^{xy}$	Hess [16], Section 5
Schnorr Signatures [23] El Gamal Encryption [14] Cramer-Shoup Encryption [12]	$pk = (y, y^x), sk = x$	Schnorr [23]
Guillou-Quisquater IBS [15]	$pk = (N, e, x^e \bmod N), sk = x$	GQ [15]

Property #2: Hash-and-Sign. As described by AOS [1], we say that a keypair (pk, sk) has a hash-and-sign algorithm if, together with an appropriate hash function H and trapdoor one-way function F_{pk} (with inverse F_{sk}^{-1}), the following algorithms are efficient:

Signing: On input (sk, m), compute $c = H(m, \alpha)$, $s = F_{sk}^{-1}(c)$, and output the signature $\sigma = (s, \alpha)$, for an arbitrary string α.

Verification: On input (pk, σ), parse σ as (s, α), compute $c = H(m, \alpha)$ and $d = F_{pk}(s)$, accept if $c = d$ and reject otherwise.

The security of such a scheme depends upon the inability of an adversary to compute F^{-1} without sk.

Examples: One popular example in this category is the Full-domain RSA signature scheme [3].

4.2 The Ad-Hoc-Group Signature Construction

Many of the ingredients needed to hijack a variety of public keypairs for group signatures already exist. Our contribution is two-fold. First, we provide a hybrid of two previous ad-hoc-group signature schemes [11, 1] which covers more *signature* keypairs than either does individually. Along these lines, we also point out that, while different signature schemes may share the same keytype (e.g., several IBS schemes [22, 16, 10] all use the same keypairs), our construction can simply use the most efficient corresponding algorithms known (among those that are SHVZK proofs of knowledge of the secret key).

Second, we broaden the scope of ad-hoc-group signature schemes by including *encryption* keypairs as potential targets. We address certain technical difficulties that result: for example, in previous ad-hoc-group schemes [11, 1] each keypair was required to be associated with a hash function that was necessary for the ad-hoc-group construction. For some signature [7] and encryption [14, 27] keys, such a hash function is not inherently defined. Thus, we exert a little creativity to include these types of keypairs as well.

As we said, our ad-hoc-group signature scheme is a hybrid of two existing schemes: one due to Cramer, Damgard, and Schoenmakers [11] (CDS) and a subsequent one proposed by Abe, Ohkubo, and Suzuki [1] (AOS). Let us describe their relation to each other and our new scheme.

Combining CDS and AOS. In 1994, CDS [11] showed how to efficiently execute a proof of partial knowledge; that is, for example, a protocol for proving, in special honest-verifier zero-knowledge, that you know either Alice's secret key or Bob's secret key. Such a protocol has many uses, and the authors briefly mention ad-hoc-group signatures as one possible application. They do not, however, deal explicitly with the difficulties of constructing such a protocol when the keypairs are heterogeneous, nor do they consider the integration of SHVZK protocols with other mechanisms.

More recently, in 2002, AOS [1] provided an ad-hoc-group signature scheme that combines keypairs which efficiently support either a hash-and-sign algorithm or a certain type of SHVZK proof of knowledge (specifically ones where the zero-knowledge simulator works by generating the response s after seeing the challenge c and *then* computes the commitment t). The AOS scheme focused on *signature* keypairs only.

In this work, we include all keypairs efficiently supporting either *any* SHVZK proof of knowledge protocol or a hash-and-sign algorithm. Thus, we provide functionality which includes the union of that of CDS [11] and AOS [1]. Furthermore, we go beyond their scope and include *encryption* keypairs and signature keypairs which are secure without random oracles.

Details of Signing Algorithm (GroupSign**):** The input is $(pk_1, \ldots, pk_{i-1}, pk_{i+1}, \ldots, pk_n, sk_i, m, \mathit{info})$. The index of the actual signer is i, n is the total number of group members, m is an arbitrary bit string to be signed, and info is some string containing information for the verifier (e.g., a description of a hash function). Furthermore, 1^{k_j} is the security parameter for member j. (These details are illustrated in Figure 1.)

1. For all group members $j = 1$ to n, except signer i, select a random string c_j from $\{0,1\}^{\max(k_1,\ldots,k_n)}$. This will serve as the *raw challenge* for member j.

2. Next, for each member j (except i), map the *raw challenge* value c_j into the *scheme-specific* challenge value c'_j using the hash function H_j such that $H_j(c_j) = c'_j$. We assume that, for every type of keypair, there is an efficient hash function mapping arbitrary strings to *scheme-specific* challenge values; for example, such a hash function for Schnorr signatures would map arbitrary strings to elements in integers modulo a large prime q. The hash function H_j is specified concretely for the eventual verifier of the signature by one of two methods: (1) it is specified as part of the public key pk_j, or (2) its description is appended to a string info. Of course, the verifier's confidence in the final signature will

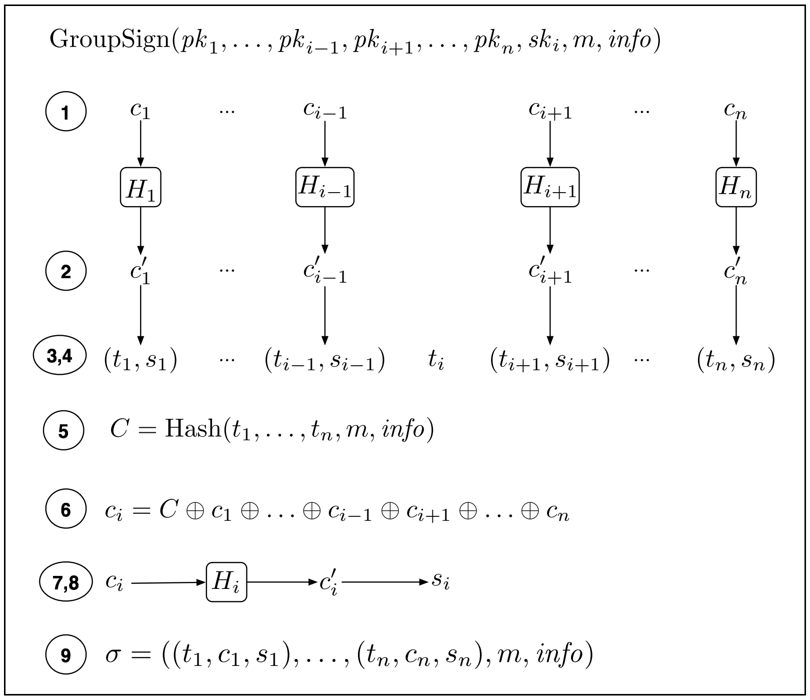
GroupSign$(pk_1, \ldots, pk_{i-1}, pk_{i+1}, \ldots, pk_n, sk_i, m, info)$
1
c_1 ··· c_{i-1} c_{i+1} ··· c_n
H_1 H_{i-1} H_{i+1} H_n
2
c'_1 ··· c'_{i-1} c'_{i+1} ··· c'_n
3,4
(t_1, s_1) ··· (t_{i-1}, s_{i-1}) t_i (t_{i+1}, s_{i+1}) ··· (t_n, s_n)
5
$C = \text{Hash}(t_1, \ldots, t_n, m, info)$
6
$c_i = C \oplus c_1 \oplus \ldots \oplus c_{i-1} \oplus c_{i+1} \oplus \ldots \oplus c_n$
7,8
c_i H_i c'_i s_i
9
$\sigma = ((t_1, c_1, s_1), \ldots, (t_n, c_n, s_n), m, info)$

Figure 1: The GroupSign Algorithm: a temporal view for a group of size n with signer i.

be affected by the choice of this hash function; therefore, a prudent signer will choose well-known (and standardized) hash functions for this purpose.

3. For each member j (except i), compute a simulated proof of knowledge transcript: (t_j, c'_j, s_j) given c'_j as input. This transcript may be completed by one of two methods depending on the properties supported by its keypair. The method of its completion can be indicated in $info$.

   - *SHVZK Proof of Knowledge:* By definition, a SHVZK proof of knowledge has an efficient algorithm for generating valid proof of knowledge transcripts of the form (t_j, c'_j, s_j), for a given c'_j. This is often (but not exclusively done) by selecting a random s_j and then computing the corresponding t_j.
   - *Hash-and-Sign Algorithm:* The observation that any hash-and-sign algorithm can be written as a SHVZK transcript is due to Abe et al. [1]. This is done as follows: choose a random value s_j from the appropriate range and set $t_j = F_{pk}(s_j) + c'_j$.

4. Choose an appropriate t_i value for the actual signer i.

5. Compute $C = \text{Hash}(t_1, \ldots, t_n, m, info)$, where $\text{Hash} : \{0,1\}^* \to \{0,1\}^{\max(k_1, \ldots, k_n)}$ is a hash function whose description is appended to the end of $info$.

6. Compute the raw challenge for the actual signer, member i, as $c_i = C \oplus c_1 \oplus \ldots \oplus c_n$.

7. Compute the scheme-specific challenge for member i as $c'_i = H_i(c_i)$, where H_i is the appropriate hash function, whose description is appended to $info$, as discussed in step 2.

8. With knowledge of sk_i, compute the response s_i to complete the transcript tuple (t_i, c'_i, s_i). In the case that sk_i is the trapdoor in a hash-and-sign algorithm, the tuple (t_i, c'_i, s_i) is formed as described in step 3, with t_i random, and $s_i = F_{pk}^{-1}(t_i - c'_i)$.

9. Output the group signature $\sigma = (\{(t_j, c_j, s_j)\}_{j\in[1,n]}, m, \mathit{info})$. Note that c_j's are the raw challenges (*not* the scheme-specific ones, which will need to be recomputed during verification). Furthermore, the values m and info may optionally be omitted from σ.

Details of Verification Algorithm (GroupVerify): The input is $(pk_1, \ldots, pk_n, m, \mathit{info}, \sigma)$. Parse the first portion of σ as $\{(t_j, c_j, s_j)\}_{j\in[1,n]}$.

1. Extract the description of the function Hash from info and compute $C = \mathrm{Hash}(t_1, \ldots, t_n, m, \mathit{info})$.

2. Check that the length of C and each $c_1, \ldots, c_n$ is $max(k_1, \ldots, k_n)$ bits, and that $C = c_1 \oplus \ldots \oplus c_n$.

3. For $j = 1$ to n, extract the description of function H_j from either pk_j or info. Compute $c'_j = H_j(c_j)$. According to the preference in info and the individual verification algorithm specified by info or pk_j, verify that the triple (t_j, c'_j, s_j) is either a valid SHVZK proof of knowledge tuple or a valid hash-and-sign tuple.

   There are many such possible algorithms for the SHVZK proof verification (we will see some examples in Section 6). For the case of hash-and-sign, this simply involves checking that $t_j = F_{pk}(s_j) + c'_j$.

4. If all checks pass, accept; otherwise, reject. We note that a verifier must further base his trust in this signature on the quality of the hash functions, proofs of knowledge, and hash-and-sign schemes included. Thus, a prudent verifier would only accept when all subfunctions included are standardized (e.g., SHA-256) and/or well-known (e.g., RSA hash-and-sign).

Theorem 4.1 *The above ad-hoc-group signature scheme is correct, existentially unforgeable under adaptive chosen message attack, and unconditionally signer-ambiguous.*

Correctness is by observation. The unconditional signer-ambiguity follows from the perfectly indistiguishable transcripts that can be produced for either the SHVZK proof-of-knowledge or the hash-and-sign protocols. The unforgeability is more involved; there we show that an adversary who can forge with probability ε can be used to invert a trapdoor permutation or fake an SHVZK prover with probability $\frac{\varepsilon}{nq_{H_j}q_{H_{wide}}} - \mathsf{negl}(k)$, where q_{H_j} is the number of queries the adversary makes to the hash function associated with the protocol we wish to break, and $q_{H_{wide}}$ is the number of queries to the ad-hoc-wide hash function. We provide more details in Appendix A.

5 Proofs of Knowledge of Bilinear Pre-Images

To hijack certain desirable bilinear-based keypairs, for which no previous efficient SHVZK or hash-and-sign algorithms were known, we must first develop some tools: proofs of knowledge of bilinear pre-images.

We work in the common parameters model, where the global parameters $\mathit{params} = (q, g, G_1, G_2, e)$ are obtained by running the Setup algorithm on input the security parameter 1^k. Let $x \in G_1$ and $Q, Q^* \in G_2$ be publicly known values, where Q is any random value in G_2 and $Q^* = e(y, y)$ for some $y \in G_1$. We provide the following novel proof-of-knowledge protocols, which are all special honest-verifier zero-knowledge:

1. PoK of canonical bilinear pre-image of Q with respect to x: $PK\{(\alpha) : e(\alpha, x) = Q\}$.
2. PoK of any bilinear pre-image of Q: $PK\{(\alpha, \beta) : e(\alpha, \beta) = Q\}$.
3. PoK of the symmetric bilinear pre-image of Q^*: $PK\{(\alpha) : e(\alpha, \alpha) = Q^*\}$.

Each of these protocols naturally extends the Schnorr protocol for proving knowledge of a discrete logarithm [23] into protocols for proving knowledge of certain pre-images of bilinear mappings. The first

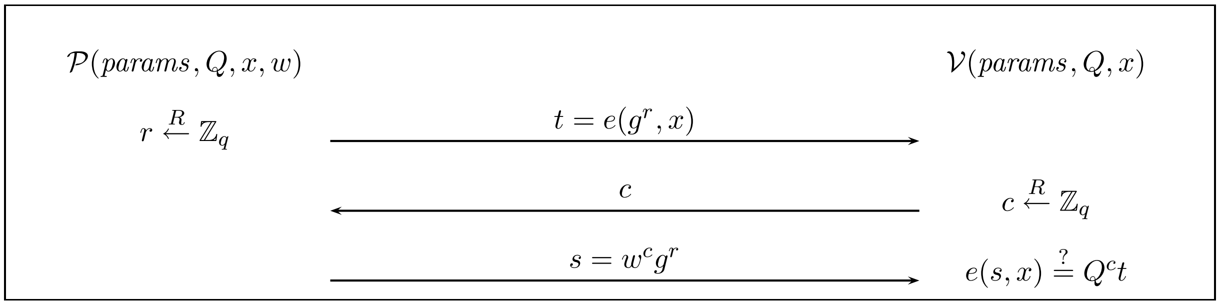
$\mathcal{P}(params, Q, x, w)$
$\mathcal{V}(params, Q, x)$
$r \overset{R}{\leftarrow} \mathbb{Z}_q$
$t = e(g^r, x)$
c
$c \overset{R}{\leftarrow} \mathbb{Z}_q$
$s = w^c g^r$
$e(s, x) \overset{?}{=} Q^c t$

Figure 2: Description of the HVZK proof of knowledge protocol $\mathcal{T}_{canonical}$ for showing knowledge of the canonical pre-image of Q with respect to element x (i.e., $w \in G_1$ such that $Q = e(w, x)$). First, the prover sends $t = e(g^r, x)$ as a commitment to a random value $g^r \in G_1$. Next, the verifier issues a random challenge $c \in \mathbb{Z}_q$. Finally, the prover responds with $s = w^c g^r$. The verifier accepts if and only if $e(s, x) = Q^c t$. Note that the prover need not know the value of r during this protocol; it is enough to know g^r, which can be selected at random from G_1.

protocol is the only tool we will use to in Section 6 to efficiently prove knowledge of several different keypair types. Thus, for space reasons, we include only the first protocol here and include the second two in Appendix B.

5.1 Proving Knowledge of the Canonical Pre-image of a Bilinear Mapping

For global parameters $params = (q, g, G_1, G_2, e)$ and any values $x \in G_1$ and $Q \in G_2$, the *canonical* pre-image of Q with respect to the bilinear mapping e and element x is the value $\alpha \in G_1$ such that $e(\alpha, x) = Q$. We provide a protocol $\mathcal{T}_{canonical}$ in Figure 2 for proving knowledge of the *canonical* pre-image of Q with respect to the bilinear mapping e and element x; that is:

$$PK\{(\alpha) : e(\alpha, x) = Q\}.$$

Observe that protocol $\mathcal{T}_{canonical}$ also works smoothly for mappings of the form $e : G_1 \times \tilde{G} \to G_2$. This protocol generalizes a protocol proposed by Hess [16] (as an identity-based signature scheme), which was proven secure by Dodis et al. [13].

Lemma 5.1 *Protocol $\mathcal{T}_{canonical}$ is a special honest-verifier, zero-knowledge proof of knowledge of the canonical pre-image of Q with respect to the global parameters $params = (q, g, G_1, G_2, e)$ and element x.* (Proof in Appendix B.1.)

6 Bilinear-Map Keypair Types

Recall that our goal is to construct an ad-hoc-group signature from as many key types as possible. In particular, we are interested in including identity-based key types, since we conjecture that they have the best chance of seeing wide-spread use. We now use the proof of knowledge from Section 5.1 to form efficient SHVZK proofs of knowledge of the (user) secret key for the two dominant IBE schemes, Boneh-Franklin [5] and Waters [27], and for the Camenisch-Lysyanskaya [7] signatures.

6.1 Boneh-Franklin IBE Keypairs

The keypairs for the master authority and the users in the Boneh-Franklin [5] identity-based encryption scheme are also used by a variety of identity-based signature schemes [22, 16, 10]. Thus, by providing a SHVZK proof of knowledge protocol for these user secret keys, we include keypairs published for several distinct schemes in our ad-hoc-group signature construction. This protocol is actually identical to one proposed by Hess [16], although the proof that it is a full SHVZK proof of knowledge is new.

Setup: On input the security parameter 1^k, run the bilinear map generation algorithm $\mathsf{Setup}(1^k) \to (q, g, G_1, G_2, e) = \mathit{params}$. Select a hash function $H : \{0,1\}^* \to G_1$. Select a random element $s \in \mathbb{Z}_q$. Output the master public key $MPK = (q, g, G_1, G_2, e, H, g^s)$ and store the master secret key $MSK = (MPK, s)$.

User Keypair: On input an identity $ID \in \{0,1\}^*$, the master computes the secret key for identity ID as $SK_{ID} = H(ID)^s$, where anyone can compute the corresponding public key $PK_{ID} = H(ID)$.

SHVZK Proof of Knowledge Protocol of SK_{ID}: On common input MPK and ID, where the prover $\mathcal{P}$ has additional input SK_{ID}, the prover $\mathcal{P}$ and verifier $\mathcal{V}$:

1. both locally compute the value $Q = e(g^s, PK_{ID})$.
2. execute the $\mathcal{T}_{canonical}(\mathcal{P}(\mathit{params}, Q, g, SK_{ID}), \mathcal{V}(\mathit{params}, Q, g))$ protocol from Section 5.1.

Let us explain why the above protocol works. The prover is being asked to prove knowledge of the canonical pre-image of $Q = e(g^s, PK_{ID})$ with respect to element g. There is only one solution for this and it is $SK_{ID} = H(ID)^s = (PK_{ID})^s$. Interestingly, when one views the value $H(ID)$ as g^b for some $b \in \mathbb{Z}_q$, it becomes apparent that the above identification protocol is actually proving knowledge of the *completion* of a DDH tuple in G_1. To see this, note that we have public values $g, g^s, H(ID) = g^b$, and the prover must show that he knows the value $g^{sb} = H(ID)^s$. Thus, for any $g, X, Y \in G_1$, we have the following proof protocol: $PK\{(\alpha) : X = g^x \wedge Y = g^y \wedge \alpha = g^{xy}\}$.

Theorem 6.1 *The above scheme is a SHVZK proof of knowledge protocol in the random oracle model.*

The above result follows from Lemma 5.1. We also note in passing that the above protocol can be turned into a *regular* signature scheme by applying the extended Fiat-Shamir heuristic due to Bellare et al. [2].

6.2 Waters IBE Keypairs

We now provide a SHVZK proof of knowledge protocol for user secret keys in the Waters [27] identity-based encryption scheme. This is the first such scheme for these keypairs. The technical difficulty is that there is an exponentially-large set of secret keys corresponding to any specific public key: as we shall see shortly the secret keys are a pair of the form (AB^r, g^r) for an arbitrary value of r. Thus, the goal is to prove knowledge of a pair of values satisfying a certain relationship to each other and the public values, rather than proving knowledge of a specific value.

We note that this technical difficulty seems to prevent the integration of Waters keys into the AOS protocol compiler. Thus, one notable advantage of our scheme over AOS is its ability to include this new keypair type.

Setup: On input the security parameter 1^k, run $\mathsf{Setup}(1^k) \to (q, g, G_1, G_2, e)$. Select random elements $h, u', u_1, \ldots, u_n \leftarrow G_1$. Select a random element $s \in \mathbb{Z}_q$. Output the master public key $MPK = (q, g, G_1, G_2, e, h, u', u_1, \ldots, u_n, g^s)$ and store the master secret key $MSK = (MPK, h^s)$. (Note that this scheme supports IDs of length $n \in \mathrm{poly}(k)$.)

User Keypair: On input an identity $ID \in \{0,1\}^*$, the master computes the secret key for identity ID by selecting a random $r \in \mathbb{Z}_q$ and outputting $SK_{ID} = (h^s F(ID)^r, g^r)$, where the function $F(ID) = u' \prod_{ID_i=1} u_i$. Anyone can compute the corresponding public key $PK_{ID} = F(ID)$.

SHVZK Proof of Knowledge Protocol of SK_{ID}: On common input MPK and ID, where the prover $\mathcal{P}$ has additional input $SK_{ID} = (S_1, S_2)$, the prover $\mathcal{P}$ and the verifier $\mathcal{V}$:

1. both locally compute $X = e(g^s, h)$, where $g^s, h \in MPK$.

2. $\mathcal{P}$ simultaneously sends $\mathcal{V}$ the value $A = e(S_1, g)$ with the first messages of the procotols:
$$P_1 = \mathcal{T}_{canonical}(\mathcal{P}(params, A, g, S_1), \mathcal{V}(params, A, g))$$
$$P_2 = \mathcal{T}_{canonical}(\mathcal{P}(params, A/X, PK_{ID}, S_2), \mathcal{V}(params, A/X, PK_{ID}))$$

   Intuitively, the prover is simultaneously showing:
$$PK\{(\alpha, \beta) : e(\alpha, g) = Xe(\beta, PK_{ID})\}.$$
3. $\mathcal{V}$ issues a challenge that can be parsed as (c_1, c_2).
4. $\mathcal{P}$ jointly sends the response of protocol P_1 on challenge c_1 with that of protocol P_2 on challenge c_2.

This protocol requires more creativity than the previous one for Boneh-Franklin keys, because Waters' user secret keys can be re-randomized. Thus, the prover must convince the verifier that he knows a pair of values *of a certain form,* rather than a fixed value. Proof of the following theorem appears in Appendix C.

Theorem 6.2 *The above scheme is a SHVZK proof of knowledge protocol.*

6.3 Camenisch-Lysyanskaya Signature Keypairs

The Camenisch-Lysyanskaya (CL) [7] signatures support a variety of efficient protocols, which have allowed construction of anonymous credentials, electronic cash, and other interesting schemes. If any user publishes a CL keypair, then that keypair can be hijacked for our ad-hoc-group signature purposes. We describe these key types followed by a SHVZK proof of knowledge protocol for these secret keys.

Setup: On input the security parameter 1^k, run the bilinear map generation algorithm $\mathsf{Setup}(1^k) \rightarrow (q, g, G_1, G_2, e) = params$.

User Keypair: On input the system parameters $params$, select random elements $x, y \in \mathbb{Z}_q$ and output the keypair $PK = (g^x, g^y)$ and $SK = g^{xy}$.

SHVZK Proof of Knowledge Protocol of SK: On common input $params$ and PK, where the prover $\mathcal{P}$ has additional input SK, the prover $\mathcal{P}$ and verifier $\mathcal{V}$:

1. both locally compute the value $Q = e(g^x, g^y)$.
2. execute the $\mathcal{T}_{canonical}(\mathcal{P}(params, Q, g, SK_{ID}), \mathcal{V}(params, Q, g))$ protocol from Section 5.1.

Theorem 6.3 *The above scheme is a SHVZK proof of knowledge protocol.*

7 Conclusion

The purpose of an ad-hoc-group signature is to help a signer gain credibility while maintaining a certain degree of anonymity. The signer may be trying to leak a secret from an in-the-know committee, or to achieve designated-verifier signatures with a sender-verifier signatory group. An ideal ad-hoc-group signature scheme will allow such a signer to proceed in the face of minimal or often inexistent cooperation from other group members.

In this paper, we presented a scheme that allows a signer to use almost any pre-existing cryptographic setup to serve in the creation of such a signature. This existing setup need not be group-signature specific: if a user publishes just about any kind of public key, it can be hijacked for the purposes of constructing a group signature that includes that user. We illustrated the strength of this approach by presenting specific methods to integrate bilinear-map-based keypairs in this scheme.

Our scheme makes it easy to establish an anonymity-preserving authentication mechanism in the face of adversity. We have yet to identify a practical keypair that cannot be integrated in our scheme.

Acknowledgements. We thank Brent Waters for useful discussions about his IBE scheme. We also thank Steve Weis for comments on an earlier version of this paper.

References

[1] Masayuki Abe, Miyako Ohkubo, and Koutarou Suzuki. 1-out-of-n signatures from a variety of keys. In Yuliang Zheng, editor, *Advances in Cryptology — ASIACRYPT '02*, volume 2501 of *LNCS*, pages 415–432. Springer Verlag, 2002.

[2] Mihir Bellare, Chanathip Namprempre, and Gregory Neven. Security proofs for identity-based identification and signature schemes. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology — EUROCRYPT '04*, volume 3027 of *LNCS*, pages 268–286. Springer Verlag, 2004.

[3] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security (CCS)*, pages 62–73, 1993.

[4] Mihir Bellare, Haixia Shi, and Chong Zhang. Foundations of group signatures: The case of dynamic groups. In *CT-RSA*, volume 3376 of LNCS, pages 136–153, 2005.

[5] Dan Boneh and Matt Franklin. Identity-based Encryption from the Weil Pairing. *SIAM Journal of Computing*, 32(3):586–615, 2003.

[6] N. Borisov, I. Goldberg, and E. Brewer. Off-the-record communication, or, why not to use PGP. In *WPES '04: the 2004 ACM workshop on Privacy in the electronic society*, pages 77–84. ACM Press, 2004.

[7] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *Advances in Cryptology — CRYPTO 2004*, volume 3152 of *LNCS*, pages 56–72. Springer Verlag, 2004.

[8] Jan Camenisch and Markus Michels. Separability and efficiency for generic group signature schemes. In *Advances in Cryptology – CRYPTO '99*, volume 1666 of LNCS, pages 413–430, 1999.

[9] Jan Camenisch and M. Stadler. Efficient group signature schemes for large groups. In *Advances in Cryptology – CRYPTO '97*, volume 1296 of LNCS, pages 410–424, 1997.

[10] Jae Choon Cha and Jung Hee Cheon. An Identity-Based Signature from Gap Diffie-Hellman Groups. In Y.G. Desmedt, editor, *PKC 2003*, volume 2567 of *LNCS*, pages 18–30. Springer-Verlag, 2003.

[11] Ronald Cramer, Ivan Damgard, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Advances in Cryptology – CRYPTO '94*, volume 839 of LNCS, pages 174–187, 1994.

[12] Ronald Cramer and Victor Shoup. A pratical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *Advances in Cryptology – CRYPTO '98*, volume 1642 of LNCS, pages 13–25, 1998.

[13] Yevgeniy Dodis, Jonathan Katz, Shouhuai Xu, and Moti Yung. Strong key-insulated signature schemes. In Y. Desmedt, editor, *PKC 2003*, volume 2567 of *LNCS*, pages 130–144. Springer Verlag, 2003.

[14] Taher El Gamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *Advances in Cryptology – CRYPTO '84*, pages 10–18, 1984.

[15] Louis C. Guillou and Jean-Jacques Quisquater. A "paradoxical" identity-based signature scheme resulting from zero-knowledge. In Shafi Goldwasser, editor, *Advances in Cryptology — CRYPTO '88*, volume 403 of *LNCS*, pages 216–231. Springer Verlag, 1988.

[16] Florian Hess. Efficient identity based signature schemes on pairings. In K. Nyberg and H. Heys, editors, *Selected Areas in Cryptography — SAC '02*, volume 2595 of *LNCS*, pages 310–324. Springer Verlag, 2002.

[17] Markus Jakobsson, Kazue Sako, and Russell Impagliazzo. Designated verifier proofs and their applications. In Ueli Maurer, editor, *Advances in Cryptology — EUROCRYPT '96*, volume 1233 of *LNCS*. Springer, 1996.

[18] Moni Naor. Deniable ring authentication. In *Proceedings of Advances in Cryptology – CRYPTO '02*, volume 2442 of *LNCS*, pages 481–498. Springer, 2002.

[19] Mario Di Raimondo and Rosario Gennaro. New approaches for deniable authentication. In *Workshop on Provable Security*, 2004.

[20] Ronald L. Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[21] Ronald L. Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. In Colin Boyd, editor, *Advances in Cryptology — ASIACRYPT '01*, volume 2248 of *LNCS*, pages 552–565. Springer Verlag, 2001.

[22] R. Sakai, K. Ohgishi, and M. Kasahara. Cryptosystems based on pairing. In *Proceedings of the Symposium on Cryptography and Information Security — SCIS 2000*, 2000.

[23] Claus-Peter Schnorr. Efficient Signature Generation by Smart Cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[24] Adi Shamir. Identity-based cryptosystems and signature schemes. In George Robert Blakley and David Chaum, editors, *Advances in Cryptology — CRYPTO '84*, volume 196 of *LNCS*, pages 47–53. Springer Verlag, 1985.

[25] Willy Susilo and Yi Mu. Non-interactive deniable ring signatures. In *the 6th International Conference on Information Security and Cryptology (ICISC) '03*, pages 397–412, 2003.

[26] Willy Susilo and Yi Mu. Deniable ring authentication revisited. In *Applied Cryptography and Network Security (ANCS) '04*, volume 3089 of LNCS, pages 149–163, 2004.

[27] Brent Waters. Efficient Identity-Based Encryption Without Random Oracles. In *Advances in Cryptology – EUROCRYPT '05*, volume 3494 of LNCS, pages 114–127, 2005.

[28] Fangguo Zhang and Kwangjo Kim. ID-Based Blind Signature and Ring Signature from Pairings. In Yuliang Zheng, editor, *Advances in Cryptology — ASIACRYPT '02*, volume 2501 of *LNCS*, pages 533–547. Springer Verlag, 2002.

A Proof Sketch of Main Construction

Due to space limitations, we will only sketch the proof of Theorem 4.1.

Proof sketch. The correctness property is fairly straight-forward.

For unconditional signer-ambiguity, observe that, in the random oracle model, all raw, and thus all scheme-specific challenge values c'_j are distributed uniformly at random. Given this fact, we know by the definition of special honest-verifier zero-knowledge that the transcripts (t_j, c'_j, s_j) produced as part of the signature are perfectly indistinguishable from those an actual signer would produce. For the case of hash-and-sign algorithms, note that s_j was chosen uniformly at random and by virtue of being a permutation $F_{pk}(s_j)$ is also distributed uniformly at random. Since both c'_j and $F_{pk}(s_j)$ are independently random, so is the value $t_j = F_{pk}(s_j) + c'_j$, exactly as a honest signer would create it. Finally, we observe that the XOR secret sharing scheme we employ obviously reveals nothing about the actual signer, since C is also random.

The unforgeability of the scheme under adaptive chosen message attack has two distinct parts: either the adversary Adv can be used to invert the trapdoor one-way function associated with a hash-and-sign algorithm or Adv can be used to fake interactive SHVZK proof of knowledge of the secret key of some scheme. Suppose the total success probability for Adv is ε.

The security game requires that Adv be provided with a list of public keys $pk_1, \ldots, pk_n$ as well as access to a signing oracle for messages of the adversary's choice. We thus concoct Adv', an adversary that will use Adv as a black box to either (1) invert a trapdoor permutation, or (2) successfully act as a prover in an SHVZK protocol for a secret key. Adv' generates $pk_1, \ldots, pk_{n-1}$ with corresponding secret keys. The last public key pk_n is set to be pk', the public key Adv' is trying to "break." Note that Adv' can easily simulate the $\mathsf{OGroupSign}$ oracle for Adv, since it has at least one of the secret keys and can thus perform real ad-hoc-group signatures on demand. The signer-ambiguity property ensures that Adv cannot determine which secret key Adv' is using. Note also that, given the Random Oracle model, Adv' can set the output value of any call from Adv to the random oracles.

We consider that, whether pk' is a hash-and-sign or a SHVZK key, Adv' will break them in the same way: by acting as the prover in a (t, c, s) protocol. In the SHVZK case, this is obvious. In the hash-and-sign case, this is done via the usual AOS approach: t is random, and $s = F^{-1}(t - c')$, where $c' = H(c)$ where H is the scheme-specific hash function modeled as a random oracle. Being able to act as the prover in such a protocol trivially implies breaking the one-wayness of the trapdoor permutation.

Now, let us describe the way in which Adv' can perform a (t, c, s) proof by using Adv as a black box.

1. Adv' sends Adv the values $pk_1, \ldots, pk_n$. (in a detailed proof, the index position of the algorithm to break should be randomized so it isn't always n).

2. Adv makes requests Q for signatures, which Adv' can easily answer, since it has at least one of the secret keys.

3. For some time, Adv' responds to all Random Oracle queries (either scheme-specific or the ad-hoc-wide random oracle) with random values.

4. At a certain point, Adv' decides that a query it receives destined for the ad-hoc-wide random oracle is "the real one," from which it extracts t_n and sends it to the external verifier.

5. Adv' receives c_n from the external verifier.

6. From that point on, Adv' picks one of the queries destined for scheme n's random oracle as "the real challenge query," and returns c_n to Adv on that one. All other queries are answered with random values.

7. Eventually Adv outputs a forged signature, from which Adv' extracts s_n, which it sends out to the external verifier.

Note that, in the final forged signature, there must be, with overwhelming probability, at least one (t, c, s) triple for which the first call to the appropriate random oracle with value c came *after* the call the to ad-hoc-wide random oracle with values $(t_1, \ldots, t_n)$. Otherwise, Adv is able to predict the output of one of those two random oracles. We call these triples the *late-queried triples.*

Thus, Adv' succeeds if it picks the right random oracle query from which to extract t_n and the right random oracle query into which to inject c_n. Also, for Adv' to succeed, the (t, c, s) triple that corresponds to the key it is trying to break must be one of the late-queried triples (possibly the only one). Finally, it is conceivable, though highly unlikely and, in fact, negligibly probable, that Adv could forge a query without ever querying the ad-hoc-wide random oracle on the set of $t_1, \ldots, t_n$, or one of the scheme-specific random oracles on that scheme's corresponding c_j.

Thus, if Adv succeeds at forgery with probability ε, then Adv' succeeds at faking an SHVZK proof or inverting a trapdoor permutation with probability $\frac{\varepsilon}{n q_{H_n} q_{H_{wide}}} - \mathsf{negl}(k)$. n is the size of the group, q_{H_n} is the number of scheme-specific random oracle queries for the random oracle of the attacked scheme, and $q_{H_{wide}}$ is the number of random oracle queries to the ad-hoc-wide random oracle.

□

B Proofs of Knowledge of Bilinear Pre-images and Their Security

B.1 Proving Knowledge of the Canonical Pre-image of a Bilinear Mapping

We now prove Lemma 5.1.

Proof. We first show that $\mathcal{T}_{canonical}$ is a proof of knowledge, and then show that it also has the special honest-verifier zero-knowledge property.

For $\mathcal{T}_{canonical}$ to be a proof of knowledge, there must exist a PPT extractor $\mathcal{E}$ that, after interacting with any prover $\mathcal{P}$ which can convince an honest $\mathcal{V}$ to accept with probability $> 1/\text{poly}(k)$, can produce the witness w with probability $> 1/\text{poly}(k)$. $\mathcal{E}$ works as follows:

1. execute $\mathcal{T}_{canonical}$ with $\mathcal{P}$ exactly as an honest $\mathcal{V}$ would to obtain the transcript (t_1, c_1, s_1);

2. rewind $\mathcal{P}$ until just after it sends t_1 and reply with a new random challenge c_2, and receive $\mathcal{P}$'s response s_2 to obtain the transcript (t_1, c_2, s_2). If an honest $\mathcal{V}$ would have accepted these transcripts, then $\mathcal{E}$ now holds the values $s_1 = w^{c_1} t_1$ and $s_2 = w^{c_2} t_1$, where $c_1 \neq c_2$, and thus $\mathcal{E}$ can compute the witness as

   $$(s_1/s_2)^{1/(c_1-c_2)} = (w^{c_1} t_1 / w^{c_2} t_1)^{1/(c_1-c_2)} = (w^{c_1-c_2})^{1/(c_1-c_2)} = w.$$

   Thus, $\mathcal{P}$ must know the witness w.

(To see this, consider that on input (g, g^a, g^b), one sets $x = g^r$ for a random $r \in \mathbb{Z}_q$ and $Q = e(g^a, g^b)$, and then carries out the proof with a cheating prover. When the witness w is extracted, as above, it must be the case that $w^r = g^{ab}$ since $e(w, x) = Q$.)

For $\mathcal{T}_{canonical}$ to be a special honest-verifier zero-knowledge proof, there must exist a PPT simulator $\mathcal{S}$ that can simulate a prover for any honest $\mathcal{V}$ without knowing the witness w. The "special" aspect of the simulation implies that, given a random c message, $\mathcal{S}$ can produce a triple (t, c, s) with the proper distribution. By saying $\mathcal{V}$ is honest, we mean that $\mathcal{V}$ has a fixed random tape from which he selects his challenges. $\mathcal{S}$ works as follows:

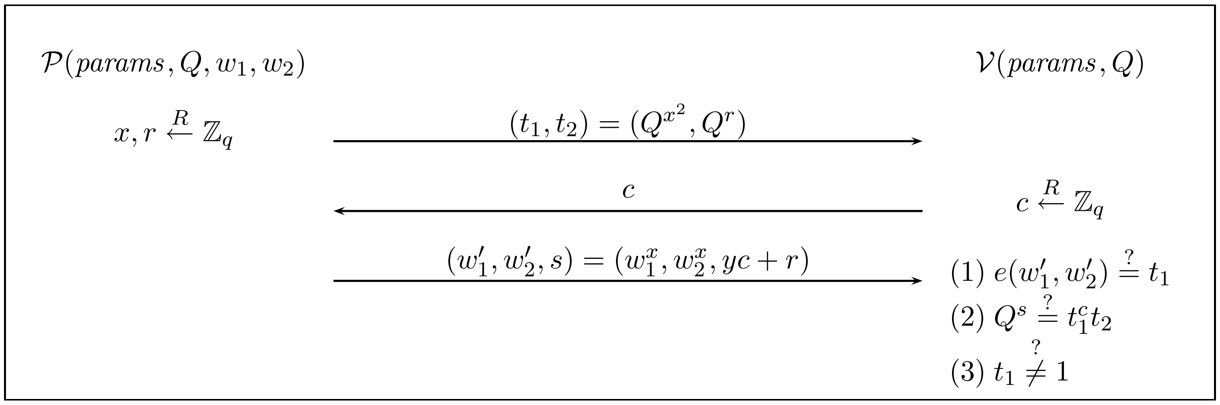
$\mathcal{P}(params, Q, w_1, w_2)$
$\mathcal{V}(params, Q)$
$x, r \xleftarrow{R} \mathbb{Z}_q$
$(t_1, t_2) = (Q^{x^2}, Q^r)$
c
$c \xleftarrow{R} \mathbb{Z}_q$
$(w_1', w_2', s) = (w_1^x, w_2^x, yc + r)$
(1) $e(w_1', w_2') \overset{?}{=} t_1$
(2) $Q^s \overset{?}{=} t_1^c t_2$
(3) $t_1 \overset{?}{\neq} 1$

Figure 3: Description of the HVZK proof of knowledge protocol $\mathcal{T}_{any}$ for showing knowledge of any pre-image of Q (i.e., $w_1, w_2 \in G_1$ such that $Q = e(w_1, w_2)$). First, the prover sends $t_1 = Q^y$ and $t_2 = Q^r$, where x, r are random values in $\mathbb{Z}_q$ and $y = x^2 \pmod q$. Next, the verifier sends a random challenge $c \in \mathbb{Z}_q$. Finally, the prover responds with $w_1' = w_1^x$, $w_2' = w_2^x$, and $s = yc + r$. The verifier accepts if and only if the following relations hold: (1) $e(w_1', w_2') = t_1$, (2) $Q^s = t_1^c t_2$, and (3) $t_1 \neq 1$.

1. send an arbitrary value in G_2, wait for $\mathcal{V}$ to send a challenge c
2. pick a random $s \in G_1$, compute $t = e(s, x)/Q^c$
3. rewind $\mathcal{V}$, send t as the first message, and s as the response to $\mathcal{V}$'s challenge. Since $\mathcal{V}$ is honest and its random-tape is fixed, he will send the same challenge c and thus accept the transcript (t, c, s). One can observe that this simulation produces perfectly distributed transcripts.

□

B.2 Proving Knowledge of *any* Pre-image of a Bilinear Mapping

For global parameters $params = (q, g, G_1, G_2, e)$ and any value $Q \in G_2$, we provide a protocol $\mathcal{T}_{any}$ in Figure 3 for proving knowledge of *any* pre-image of Q with respect to the bilinear mapping e; that is:

$$PK\{(\alpha, \beta) : e(\alpha, \beta) = Q\}.$$

This is a generalization of the canonical pre-image protocol. Though slightly less efficient, this protocol is interesting because, unlike the previous one, there are many possible witnesses.

Observe that protocol $\mathcal{T}_{any}$ also works smoothly for mappings of the form $e : G_1 \times \tilde{G} \to G_2$. For a witness indistinguishable proof of knowledge, the prover may simply choose a random $c \in \mathbb{Z}_q^*$ and send the verifier $(w_1^c, w_2^{1/c})$. However, this has the potentially negative side-effect of providing the verifier with a valid pre-image of Q. Thus, we achieve something stronger in $\mathcal{T}_{any}$.

Lemma B.1 *Protocol $\mathcal{T}_{any}$ is an honest-verifier, zero-knowledge proof of knowledge of a pre-image of Q with respect to the global parameters $params = (q, g, G_1, G_2, e)$ under the Bilinear One-Way Assumption; that is, given $(params, Q)$ for a random $Q \in G_2$, it is hard to compute any $x, y \in G_1$ such that $e(x, y) = Q$.*

Proof. We first show that $\mathcal{T}_{any}$ is a proof of knowledge, and then show that it also has the honest-verifier zero-knowledge property.

For $\mathcal{T}_{any}$, the proof of knowledge extractor $\mathcal{E}$ works as follows: (Step 1) execute $\mathcal{T}_{any}$ with $\mathcal{P}$ exactly as an honest $\mathcal{V}$ would to obtain the transcript $((t_1, t_2), c, (w_1, w_2, s))$; (Step 2) rewind $\mathcal{P}$ until just after it sends (t_1, t_2) and reply with a new random challenge c', and receive $\mathcal{P}$'s response to obtain the transcript

$$\begin{array}{lcl}
\mathcal{P}(params, Q, w) & & \mathcal{V}(params, Q) \\
x, r \xleftarrow{R} \mathbb{Z}_q & \xrightarrow{\quad (t_1, t_2) = (Q^{x^2}, Q^r) \quad} & \\
& \xleftarrow{\quad c \quad} & c \xleftarrow{R} \mathbb{Z}_q \\
& \xrightarrow{\quad (w', s) = (w^x, yc + r) \quad} & (1)\ e(w', w') \stackrel{?}{=} t_1 \\
& & (2)\ Q^s \stackrel{?}{=} t_1^c t_2 \\
& & (3)\ t_1 \stackrel{?}{\neq} 1
\end{array}$$

Figure 4: Description of the HVZK proof of knowledge protocol $\mathcal{T}_{sym}$ for showing knowledge of a symmetric pre-image of Q (i.e., $w \in G_1$ such that $Q = e(w, w)$). $\mathcal{P}$ and $\mathcal{V}$ follow the $\mathcal{T}_{any}$ protocol; except in round three, $\mathcal{P}$ only sends two values $w' = w^x$ and $s = yc+r$, and $\mathcal{V}$ accepts if and only if the following relations hold: (1) $e(w', w') = t_1$, (2) $Q^s = t_1^c t_2$, and (3) $t_1 \neq 1$.

$((t_1, t_2), c', (w'_1, w'_2, s'))$. If an honest $\mathcal{V}$ would have accepted these transcripts, then $\mathcal{E}$ now holds the values $s = yc + r$ and $s' = yc' + r$, where $c \neq c'$, and thus $\mathcal{E}$ can recover the value y as $(s - s')/(c - c') = y$. Next, $\mathcal{E}$ computes x as the square root of y modulo q. Finally, $\mathcal{E}$ can output the witness $(w_1^{1/x}, w_2^{1/x})$. Thus, $\mathcal{P}$ must know a witness, or she breaks the Bilinear One-Way Assumption.

For $\mathcal{T}_{any}$, the honest-verifier zero-knowledge simulator $\mathcal{S}$ works as follows: (Step 1) select random values $a, b \in G_1$ and $d \in G_2$, send to $\mathcal{V}$ the pair of values $(t_1 = e(a, b), d)$, and wait for $\mathcal{V}$ to send a challenge c; (Step 2) pick a random $s \in G_1$, compute $t_2 = Q^s / t_1^c$; (Step 3) rewind $\mathcal{V}$ to the beginning, send (t_1, t_2) as the first message, and s as the response to $\mathcal{V}$'s challenge. Since $\mathcal{V}$ is honest, he will send the same challenge c and thus accept the transcript $((t_1, t_2), c, s)$. One can observe that this simulation produces perfectly distributed transcripts. $\square$

B.3 Proving Knowledge of the Symmetric Pre-image of a Bilinear Mapping

For global parameters $params = (q, g, G_1, G_2, e)$ and special values of $W \in G_2$, where W is of the form $e(g, g)^{a^2}$ for some $a \in \mathbb{Z}_q$, the above protocol can be adjusted to become a protocol $\mathcal{T}_{sym}$ in Figure 4 for proving knowledge of the *symmetric* pre-image of Q with respect to the bilinear mapping e; that is:

$$PK\{(\alpha) : e(\alpha, \alpha) = Q\}.$$

Lemma B.2 *Protocol $\mathcal{T}_{sym}$ is an honest-verifier, zero-knowledge proof of knowledge of the symmetric pre-image of Q with respect to the global parameters $params = (q, g, G_1, G_2, e)$ under the Bilinear One-Way Assumption.*

C Proof for Waters IBE Keypairs

We now provide proof of Theorem 6.2.

Proof. We first show that the protocol is a proof of knowledge; we then argue that it is also special honest-verifier zero knowledge.

The proof of knowledge extractor $\mathcal{E}$ works as follows: (Step 1) execute the protocol with $\mathcal{P}$ exactly as an honest $\mathcal{V}$ would to obtain the transcript $((A, t_1, t_2), (c_1, c_2), (j_1, j_2))$; (Step 2) rewind $\mathcal{P}$ until just after it sends (A, t_1, t_2) and reply with a new random challenge (c'_1, c'_2), and receive $\mathcal{P}$'s response to obtain the transcript $((A, t_1, t_2), (c'_1, c'_2), (j'_1, j'_2))$. If an honest $\mathcal{V}$ would have accepted these transcripts, then $\mathcal{E}$ can extract the witness (i.e., user's secret key) as follows.

Since $c_1 \neq c'_1$ and $c_2 \neq c'_2$, $\mathcal{E}$ can recover secret key (S_1, S_2) as $(j_2/j'_2)^{1/(c_2-c'_2)} = S_1$ and $(j_1/j'_1)^{1/(c_1-c'_1)}$ $= S_2$. Thus, $\mathcal{P}$ must know the secret key.

The special honest-verifier zero-knowledge simulator $\mathcal{S}$ works as follows: (Step 1) select a random value $r \in \mathbb{Z}_q$ and $t'_1, t'_2 \in G_2$, compute $A = e(g^s, h)e(g, f(ID))^r$ (where g^s and h are part of MPK), and send to $\mathcal{V}$ the tuple (A, t'_1, t'_2), and wait for $\mathcal{V}$ to send a challenge (c_1, c_2); (Step 2) pick two random values $j_1, j_2 \in G_1$, compute $t_1 = e(j_1, f(ID))/(A/X)^{c_1}$ and $t_2 = e(j_2, g)/A^{c_2}$; (Step 3) rewind $\mathcal{V}$ to the beginning, send (A, t_1, t_2) as the first message, and (j_1, j_2) as the response to $\mathcal{V}$'s challenge. Since $\mathcal{V}$ is honest, he will send the same challenge (c_1, c_2) and thus accept this conversation. Again, we observe that the transcripts produced by $\mathcal{S}$ are perfectly distributed. This follows in part from the fact that a real prover can randomize his witness $(h^s f(ID)^r, g^r)$ as $(h^s f(ID)^r f(ID)^{r'}, g^r g^{r'}) = (h^s f(ID)^{r+r'}, g^{r+r'})$ for any value of $(r + r') \in \mathbb{Z}_q$. Thus, *any* random value from $\mathbb{Z}_q$ used by $\mathcal{S}$ to create A in step one is valid and from the correct distribution. $\square$