Group Signatures with Verifier-Local Revocation

Dan Boneh
Stanford University
dabo@cs.stanford.edu

Hovav Shacham
Stanford University
hovav@cs.stanford.edu

ABSTRACT

Group signatures have recently become important for enabling privacy-preserving attestation in projects such as Microsoft's NGSCB effort (formerly Palladium). Revocation is critical to the security of such systems. We construct a *short* group signature scheme that supports Verifier-Local Revocation (VLR). In this model, revocation messages are only sent to signature verifiers (as opposed to both signers and verifiers). Consequently there is no need to contact individual signers when some user is revoked. This model is appealing for systems providing attestation capabilities. Our signatures are as short as standard RSA signatures with comparable security. Security of our group signature (in the random oracle model) is based on the Strong Diffie-Hellman assumption and the Decision Linear assumption in bilinear groups. We give a precise model for VLR group signatures and discuss its implications.

Categories and Subject Descriptors

E.3 [**Data**]: Data Encryption—*Public key cryptosystems*

General Terms

Design, Theory

Keywords

Group signatures, revocation, trusted computing

1. INTRODUCTION

Group signatures, introduced by Chaum and van Heyst [11], provide anonymity for signers. Each group member has a private key that enables him to sign messages. However, the resulting signature keeps the identity of the signer secret. Often there is a third party that can undo the signature anonymity (trace) using a special trapdoor [11, 1]. Some systems support revocation [10, 2, 21, 12], where group membership can be disabled without affecting the signing ability of unrevoked members. Currently, the most efficient constructions are based on the Strong-RSA assumption introduced by Baric and Pfitzman [3]. These signatures are usually much longer than RSA signatures of comparable security.

A number of recent projects require properties provided by group signatures. One such project is the Trusted Computing effort (TCG) [20] that, among other things, enables a desktop PC to prove to a remote party what software it is running via a process called *attestation*. Group signatures are needed for privacy-preserving attestation [9]. To enable attestation, each computer ships with an embedded TCG tamper-resistant chip that signs certain system components using a secret key embedded in the chip. During attestation to a remote party (e.g., a bank) these signatures are sent to the remote party. To maintain user privacy it is desirable that the signatures not reveal the identity of the chip that issued them. To do so, each tamper resistant chip issues a group signature (rather than a standard signature) on system components that it signs. Here the group is the set of all TCG-enabled machines. The group signature proves that the attestation was issued by a valid tamper-resistant chip, but hides which machine it comes from. We refer to this as privacy-preserving attestation. Revocation is critical in such systems — if the private key in a TCG chip is exposed, all signatures from that chip must be invalidated since otherwise attestation becomes meaningless.

In this paper we focus on a revocation model that is motivated by privacy-preserving attestation. At a high level, one can consider three natural communication models for revoking a user's signing capabilities, without affecting other group members:

1. The simplest method revokes user i by issuing a new signature verification key and giving each signer, except user i, a new signing key. This requires an individual secret message to each signer (e.g., TCG chip) and a public broadcast message to all verifiers.

2. A better revocation mechanism sends a single short public broadcast message to all signers and verifiers. A recent system by Camenisch and Lysyanskaya [10] provides such a mechanism.

3. Brickell [9] proposes a simpler mechanism where revocation messages are only sent to signature verifiers, so that there is no need ever to communicate with an end-user machine. A similar mechanism was considered by Ateniese et al. [2] and Kiayias et al. [14]. We

CCS'04, October 25-29, 2004, Washington, DC, USA.

refer to this as Verifier-Local Revocation (VLR) group signatures.

In this paper we formalize the concept of VLR group signatures. In Section 2 we give a precise definition of security using the framework of Bellare et al. [4]. We discuss the mechanics of working with VLR group signatures and the features they provide. In Section 4 we present our short VLR group signature scheme. Signatures in our system are about the same length as standard RSA signatures of comparable security. Security is based on the Strong Diffie-Hellman (SDH) assumption [6] in groups with a bilinear map. We also need the Decision Linear assumption, an assumption that has proven useful for constructing short group signatures [7].

VLR group signatures are implemented by giving the signature verification algorithm an additional argument called the Revocation List (RL). The RL contains a token for each revoked user. The verification algorithm accepts all signatures issued by unrevoked users and reveals no information about which unrevoked user issued the signature. However, if a user is ever revoked (by having his revocation token added to the RL), signatures from that user are no longer accepted. It follows that signatures from a revoked user become linkable: to test that two signatures were issued by the same revoked user, verify the signatures once using the RL before the user is revoked and once using the RL after. As a result, users who break the tamper resistance of their TCG chip and are revoked deliberately lose their privacy. Our specific VLR group signatures have an additional useful property: given a user's private key it is easy to derive that user's revocation token — the revocation token is the left half of the private key. Hence, any private key that is published on the web can be trivially added to the RL and revoked. This potentially eliminates the need for a trusted revocation authority. Instead, revocation could be done by just scanning the web and newsgroups for exposed private keys and telling all signature verifiers to add these keys to their RL. We discuss this in more detail in the next section.

Notation. Throughout the paper we use boldface to denote a vector of elements such as $\boldsymbol{gsk}$. We use $\boldsymbol{gsk}[i]$ to denote the i'th element of the vector $\boldsymbol{gsk}$.

2. VERIFIER-LOCAL REVOCATION

A verifier-local group signature scheme comprises three algorithms, KeyGen, Sign, and Verify, which behave as follows:

$\mathsf{KeyGen}(n)$. This randomized algorithm takes as input a parameter n, the number of members of the group. It outputs a group public key gpk, an n-element vector of user keys $\boldsymbol{gsk} = (\boldsymbol{gsk}[1], \boldsymbol{gsk}[2], \ldots, \boldsymbol{gsk}[n])$, and an n-element vector of user revocation tokens $\boldsymbol{grt}$, similarly indexed.

$\mathsf{Sign}(gpk, \boldsymbol{gsk}[i], M)$. This randomized algorithm takes as input the group public key gpk, a private key $\boldsymbol{gsk}[i]$, and a message $M \in \{0,1\}^*$, and returns a signature σ.

$\mathsf{Verify}(gpk, RL, \sigma, M)$. The verification algorithm takes as input the group public key gpk, a set of revocation tokens RL (whose elements form a subset of the elements of $\boldsymbol{grt}$), and a purported signature σ on a message M. It returns either `valid` or `invalid`. The latter response can mean either that σ is not a valid signature, or that the user who generated it has been revoked.

Implicit Tracing Algorithm. Any VLR group signature scheme has an associated implicit tracing algorithm that, using a secret tracing key, can trace a signature to at least one group member who generated it. The vector of revocation tokens, $\boldsymbol{grt}$, functions as this secret tracing key. Given a valid message-signature pair (M, σ), a party possessing all the revocation tokens $\boldsymbol{grt}$ can determine which user issued the signature using the following algorithm:

1. For each $i = 1, \ldots, n$ run the verification algorithm on M, σ with revocation list $RL = \{\boldsymbol{grt}[i]\}$.
2. Output the index of the first user for which the verification algorithm says `invalid`. Output `fail` if the signature verifies properly for all n users.

Our security definitions below explain why this is a correct tracing algorithm. The algorithm above demonstrates that the $\boldsymbol{grt}$ vector can function as a secret tracing key, if so desired. Note that $\boldsymbol{grt}$ in our system can be derived from just one value so that there is no need to store a large vector as a tracing key.

In the constructions we have in mind, a user can derive her revocation token from her private key, and can therefore determine whether her key was used to generate a particular signature. We refer to this as *selfless-anonymity*: a group member can tell whether she generated a particular signature σ, but if she didn't she learns nothing else about the origin of σ. We describe a new security model that captures this notion. We use the framework of Bellare et al. [4].

A secure VLR group signature scheme must satisfy three requirements: correctness, traceability, and selfless-anonymity. We describe each in turn.

Correctness. This requires that, for all $(gpk, \boldsymbol{gsk}, \boldsymbol{grt})$ generated by the generation algorithm, every signature generated by a user verify as valid, except when the user is revoked; or, formally, that

$$\mathsf{Verify}(gpk, RL, \mathsf{Sign}(gpk, \boldsymbol{gsk}[i], M), M) = \texttt{valid} \iff \boldsymbol{grt}[i] \notin RL \ . \quad (1)$$

Traceability. We say that a VLR group signature scheme is traceable if no adversary can win the traceability game. In the traceability game, the adversary's goal is to forge a signature that cannot be traced to one of the users in his coalition using the implicit tracing algorithm above. Let n be a given group size. The traceability game, between a challenger and an adversary $\mathcal{A}$, is defined as follows.

Setup. The challenger runs algorithm $\mathsf{KeyGen}(n)$, obtaining group parameters gpk, $\boldsymbol{gsk}$, and $\boldsymbol{grt}$. He provides the adversary $\mathcal{A}$ with gpk and $\boldsymbol{grt}$, and sets $U \leftarrow \emptyset$.

Queries Algorithm $\mathcal{A}$ can make queries of the challenger, as follows.

Signing. Algorithm $\mathcal{A}$ requests a signature on an arbitrary message M for the user at index i, where $1 \leq i \leq n$. The challenger computes $\sigma \leftarrow \mathsf{Sign}(gpk, \boldsymbol{gsk}[i], M)$ and returns the signature σ to $\mathcal{A}$.

Corruption. Algorithm $\mathcal{A}$ requests the private key of the user at index i, $1 \leq i \leq n$. The challenger appends i to U, the adversary's coalition, and responds with $\boldsymbol{gsk}[i]$.

Response. Finally, forger $\mathcal{A}$ outputs a message M^*, a set RL^* of revocation tokens, and a signature σ^*.

The forger wins if: (1) σ^* is accepted by the verification algorithm as a valid signature on M^* with revocation-token set RL^*; (2) σ^* traces (using the implicit tracing algorithm above) to some user outside of the coalition $U \setminus RL^*$, or the tracing algorithm fails; and (3) σ^* is nontrivial, i.e., $\mathcal{A}$ did not obtain σ^* by making a signing query at M^*.

We denote by $\mathsf{Succ\,PT}_{\mathcal{A}}$ the probability that $\mathcal{A}$ wins the game. The probability is taken over the coin tosses of $\mathcal{A}$ and the randomized key generation and signing algorithms.

The security proof for our system is set in the random oracle model [5] and therefore we include in our security definitions an extra parameter q_H denoting the number of random oracle queries that the adversary issues.

Definition 1. An aggregate forger $\mathcal{A}$ $(t, q_H, q_S, n, \epsilon)$-breaks traceability in an n-user VLR group signature scheme if: $\mathcal{A}$ runs in time at most t; $\mathcal{A}$ makes at most q_H hash oracle queries and at most q_S signing queries; and $\mathsf{Succ\,PT}_{\mathcal{A}}$ is at least ϵ.

Selfless-anonymity. In the selfless-anonymity game, the adversary's goal is to determine which of two keys generated a signature. He is not given access to either key. The game is defined as follows.

Setup. The challenger runs the KeyGen algorithm, obtaining group parameters gpk, $\boldsymbol{gsk}$, and $\boldsymbol{grt}$. It provides the adversary $\mathcal{A}$ with gpk.

Queries. Algorithm $\mathcal{A}$ can make queries of the challenger, as follows.

Signing. Algorithm $\mathcal{A}$ requests a signature on an arbitrary message M for the user at index i, where $1 \leq i \leq n$. The challenger computes $\sigma \leftarrow \mathsf{Sign}(gpk, \boldsymbol{gsk}[i], M)$ and returns the signature σ to $\mathcal{A}$.

Corruption. Algorithm $\mathcal{A}$ request the private key of the user at index i, $1 \leq i \leq n$. The challenger responds with $\boldsymbol{gsk}[i]$.

Revocation. Algorithm $\mathcal{A}$ can request the revocation token of the user at index i, $1 \leq i \leq n$. The challenger responds with $\boldsymbol{grt}[i]$.

Challenge. Algorithm $\mathcal{A}$ outputs a message M and two indices i_0 and i_1. It must have made neither a corruption nor a revocation query at either index. The challenger chooses a bit $b \xleftarrow{\mathrm{R}} \{0, 1\}$ uniformly at random, computes a signature on M by user i_b as $\sigma^* \leftarrow \mathsf{Sign}(gpk, \boldsymbol{gsk}[i_b], M)$, and provides σ^* to $\mathcal{A}$.

Restricted Queries. After obtaining the challenge, algorithm $\mathcal{A}$ can make additional queries of the challenger, restricted as follows.

Signing. Algorithm $\mathcal{A}$ can make signing queries as before.

Corruption. As before, but $\mathcal{A}$ cannot make corruption queries at i_0 and i_1.

Revocation. As before, but $\mathcal{A}$ cannot make revocation queries at i_0 and i_1.

Output. Finally, $\mathcal{A}$ outputs a bit b', its guess of b. The adversary wins if $b' = b$.

We define $\mathcal{A}$'s advantage in winning the game as $\mathsf{Succ\,PA}_{\mathcal{A}}$ as $|\Pr[b = b'] - 1/2|$. The probability is taken over the coin tosses of $\mathcal{A}$, of the randomized key generation and signing algorithms, and the choice of b. Note that $\mathcal{A}$ can make no more than $n - 2$ corruption and revocation queries.

Definition 2. An aggregate adversary $\mathcal{A}$ $(t, q_H, q_S, n, \epsilon)$-breaks selfless-anonymity in an n-user VLR group signature scheme if: $\mathcal{A}$ runs in time at most t; $\mathcal{A}$ makes at most q_H queries to the hash function and at most q_S signing queries; and $\mathsf{Adv\,PA}_{\mathcal{A}}$ is at least ϵ.

Definition 3. A group signature scheme with verifier-local revocation is $(t, q_H, q_S, n, \epsilon)$ secure in the VLR security model if: it is correct; no algorithm $(t, q_H, q_S, n, \epsilon)$-breaks its traceability; and no algorithm $(t, q_H, q_S, n, \epsilon)$-breaks its selfless-anonymity.

We note that a signature scheme that satisfies the VLR security model above is existentially unforgeable under a chosen message attack. This follows immediately from the traceability game.

3. BACKGROUND

Our VLR group signature scheme makes use of bilinear groups. The security of the scheme depends on the Strong Diffie-Hellman assumption and the Decision Linear assumption. In this section, we review the definitions of bilinear groups and of the complexity assumptions.

3.1 Bilinear Groups

We first review a few concepts related to bilinear maps. Although many groups with a useful bilinear map are based on elliptic curves, our definitions are abstract and do not require any familiarity with elliptic curves. We follow the notation of Boneh, Lynn, and Shacham [8]:

1. G_1 and G_2 are two (multiplicative) cyclic groups of prime order p;
2. g_1 is a generator of G_1 and g_2 is a generator of G_2;
3. ψ is an efficiently computable isomorphism from G_2 to G_1, with $\psi(g_2) = g_1$; and
4. e is an efficiently computable bilinear map $e : G_1 \times G_2 \to G_T$ with the following properties:
   - Bilinear: for all $u \in G_1, v \in G_2$ and $a, b \in \mathbb{Z}$, $e(u^a, v^b) = e(u, v)^{ab}$.
   - Non-degenerate: $e(g_1, g_2) \neq 1$.

Throughout the paper, we consider bilinear maps $e : G_1 \times G_2 \to G_T$ where all groups G_1, G_2, G_T are multiplicative and of prime order p. One could set $G_1 = G_2$. However, we allow for the more general case where $G_1 \neq G_2$ so that our constructions can make use of certain families of non-supersingular elliptic curves defined by Miyaji et al. [16]. In this paper we only use the fact that G_1 can be of size approximately 2^{170}, elements in G_1 are 171-bit strings, and discrete log in G_1 is as hard as discrete log in $\mathbb{Z}_q^*$ where q is a 1020-bit prime power. We will use these groups to construct short VLR group signatures.

We say that two groups (G_1, G_2) as above are a bilinear group pair if the group action G_1 and G_2, the map ψ, and the bilinear map e are all efficiently computable.

To keep the discussion general, we simply assume that ψ exists and is efficiently computable. When G_1, G_2 are subgroups of the group of points of an elliptic curve $E/\mathbb{F}_q$, the trace map on the curve can be used as this isomorphism. In this case, $G_1 \subseteq E(\mathbb{F}_q)$ and $G_2 \subseteq E(\mathbb{F}_{q^r})$.

3.2 Complexity Assumptions

The security of our VLR group signatures relies on the difficulty of two problems: the Strong Diffie-Hellman problem and the Decision Linear problem. We describe each of these problems in turn.

3.2.1 The Strong Diffie-Hellman Assumption

Let G_1, G_2 be cyclic group of prime order p, where possibly $G_1 = G_2$. Let g_1 be a generator of G_1 and g_2 a generator of G_2.

q-Strong Diffie-Hellman Problem. The q-SDH problem in (G_1, G_2) is defined as follows: given a $(q+2)$-tuple $(g_1, g_2, g_2^\gamma, g_2^{(\gamma^2)}, \ldots, g_2^{(\gamma^q)})$ as input, output a pair $(g_1^{1/(\gamma+x)}, x)$, where $x \in \mathbb{Z}_p^*$. An algorithm $\mathcal{A}$ has advantage ϵ in solving q-SDH in (G_1, G_2) if

$$\Pr\left[\mathcal{A}(g_1, g_2, g_2^\gamma, \ldots, g_2^{(\gamma^q)}) = (g_1^{\frac{1}{\gamma+x}}, x)\right] \geq \epsilon \ ,$$

where the probability is over the random choice of generator g_2 in G_2 (with $g_1 \leftarrow \psi(g_2)$), of γ in $\mathbb{Z}_p^*$, and of the random bits of $\mathcal{A}$.

Definition 4. We say that the (q, t, ϵ)-SDH assumption holds in (G_1, G_2) if no t-time algorithm has advantage at least ϵ in solving the q-SDH problem in (G_1, G_2).

Occasionally we drop the t and ϵ and refer to the q-SDH assumption rather than the (q, t, ϵ)-SDH assumption. The q-SDH assumption was recently used by Boneh and Boyen [6] to construct a short signature scheme without random oracles. To gain confidence in the assumption they prove that it holds in generic groups [19]. The assumption has properties similar to the Strong-RSA assumption [3] and we exploit these properties to build short VLR group signatures. Mitsunari et al. [15] use a related assumption (where x is part of the input given to the adversary) in a tracing traitors system.

3.2.2 The Decision Linear Assumption

With $g_1 \in G_1$ as above, along with arbitrary generators u, v, and h of G_1, consider the following problem:

Decision Linear on G_1: Given $u, v, h, u^a, v^b, h^c \in G_1$ as input, output `yes` if $a + b = c$ and `no` otherwise.

One can easily show that an algorithm for solving Decision Linear in G_1 gives an algorithm for solving DDH in G_1. The converse is believed to be false. That is, it is believed that Decision Linear is a hard problem even in bilinear groups where DDH is easy. More precisely, we define the advantage of an algorithm $\mathcal{A}$ in deciding the Decision Linear problem on G_1 as

$$\mathsf{Adv\,Linear}_{\mathcal{A}} \stackrel{\text{def}}{=} \left| \begin{array}{l} \Pr\left[\begin{array}{r} \mathcal{A}(u, v, h, u^a, v^b, h^{a+b}) = \texttt{yes} \ : \\ u, v, h \stackrel{\text{R}}{\leftarrow} G_1, a, b \stackrel{\text{R}}{\leftarrow} \mathbb{Z}_p \end{array}\right] \\ - \Pr\left[\begin{array}{r} \mathcal{A}(u, v, h, u^a, v^b, \eta) = \texttt{yes} \ : \\ u, v, h, \eta \stackrel{\text{R}}{\leftarrow} G_1, a, b \stackrel{\text{R}}{\leftarrow} \mathbb{Z}_p \end{array}\right] \end{array} \right| .$$

The probability is over the uniform random choice of the parameters to $\mathcal{A}$, and over the coin tosses of $\mathcal{A}$. We say that an algorithm $\mathcal{A}$ (t, ϵ)-decides Decision Linear on G_1 if $\mathcal{A}$ runs in time at most t, and $\mathsf{Adv\,Linear}_{\mathcal{A}}$ is at least ϵ.

Definition 5. We say that the (t, ϵ)-Decision Linear assumption holds in G_1 if no t-time algorithm has advantage at least ϵ in solving the Decision Linear problem in G_1.

The Decision Linear assumption was recently introduced by Boneh, Boyen, and Shacham [7]. They prove that the problem is intractable in generic bilinear groups.

4. SHORT VLR GROUP SIGNATURES

In this section, we describe in detail our VLR group signature based on SDH. In the next section, we give intuition for how the scheme is derived.

Consider bilinear groups (G_1, G_2) with isomorphism ψ and respective generators g_1 and g_2, as in Section 3.1. The scheme employs hash functions H_0 and H, with respective ranges G_2^2 and $\mathbb{Z}_p$, treated as random oracles.

KeyGen(n)**.** The key generation algorithm takes as input n, the number of user keys to generate. It proceeds as follows:

1. Select a generator g_2 in G_2 uniformly at random, and set $g_1 \leftarrow \psi(g_2)$.
2. Select $\gamma \stackrel{\text{R}}{\leftarrow} \mathbb{Z}_p^*$ and set $w = g_2^\gamma$.
3. Using γ, generate for each user an SDH tuple (A_i, x_i) by selecting $x_i \stackrel{\text{R}}{\leftarrow} \mathbb{Z}_p^*$ such that $\gamma + x_i \neq 0$, and setting $A_i \leftarrow g_1^{1/(\gamma+x_i)}$.

The group public key is $gpk = (g_1, g_2, w)$. Each user's private key is her tuple $\boldsymbol{gsk}[i] = (A_i, x_i)$. The revocation token corresponding to a user's key (A_i, x_i) is $\boldsymbol{grt}[i] = A_i$. The algorithm outputs $(gpk, \boldsymbol{gsk}, \boldsymbol{grt})$. No party is allowed to possess γ; it is only known to the private-key issuer.

Sign$(gpk, \boldsymbol{gsk}[i], M)$**.** The signing algorithm takes as input a group public key $gpk = (g_1, g_2, w)$, a user private key $\boldsymbol{gsk}[i] = (A_i, x_i)$, and a message $M \in \{0,1\}^*$, and proceeds as follows.

1. Pick a random nonce $r \stackrel{\text{R}}{\leftarrow} \mathbb{Z}_p$. Obtain generators $(\hat{u}, \hat{v})$ in G_2 from H_0 as

$$(\hat{u}, \hat{v}) \leftarrow H_0(gpk, M, r) \in G_2^2 \ , \qquad (2)$$

and compute their images in G_1:

$$u \leftarrow \psi(\hat{u}) \ , \qquad v \leftarrow \psi(\hat{v}) \ .$$

2. Select an exponent $\alpha \xleftarrow{R} \mathbb{Z}_p$ and compute:

$$T_1 \leftarrow u^\alpha \quad \text{and} \quad T_2 \leftarrow A_i v^\alpha \ . \tag{3}$$

3. Set $\delta \leftarrow x_i\alpha \in \mathbb{Z}_p$. Pick blinding values r_α, r_x, and $r_\delta \xleftarrow{R} \mathbb{Z}_p$.
4. Compute helper values R_1, R_2, and R_3:

$$\begin{aligned} R_1 &\leftarrow u^{r_\alpha} \qquad R_3 \leftarrow T_1^{r_x} \cdot u^{-r_\delta} \\ R_2 &\leftarrow e(T_2, g_2)^{r_x} \cdot e(v, w)^{-r_\alpha} \cdot e(v, g_2)^{-r_\delta} \ . \end{aligned} \tag{4}$$

5. Compute a challenge value $c \in \mathbb{Z}_p$ using H:

$$c \leftarrow H(gpk, M, r, T_1, T_2, R_1, R_2, R_3) \in \mathbb{Z}_p \ . \tag{5}$$

6. Compute $s_\alpha = r_\alpha + c\alpha$, $s_x = r_x + cx_i$, and $s_\delta = r_\delta + c\delta \in \mathbb{Z}_p$.

Output the signature $\sigma \leftarrow (r, T_1, T_2, c, s_\alpha, s_x, s_\delta)$.

Verify(gpk, R, σ, M)**.** The verification algorithm takes as input a group public key $gpk = (g_1, g_2, w)$, a set RL of revocation tokens (each an element of G_1), a purported signature $\sigma = (r, T_1, T_2, c, s_\alpha, s_x, s_\delta)$, and a message $M \in \{0, 1\}^*$, and proceeds in two phases. First, it ensures that the signature σ is valid; then it ensures that σ was not generated by a revoked user. It accepts only if both conditions hold.

1. Signature Check. Check that σ is a valid signature, as follows.

1. Compute $\hat{u}$ and $\hat{v}$ using equation (2), and their images u and v in G_1:

$$u \leftarrow \psi(\hat{u}) \ , \qquad v \leftarrow \psi(\hat{v}) \ .$$

2. Rederive R_1, R_2, and R_3 as:

$$\begin{aligned} \tilde{R_1} &\leftarrow u^{s_\alpha}/T_1^c \qquad \tilde{R_3} \leftarrow T_1^{s_x} u^{-s_\delta} \\ \tilde{R_2} &\leftarrow e(T_2, g_2)^{s_x} e(v, w)^{-s_\alpha} e(v, g_2)^{-s_\delta} \\ &\quad \cdot \big(e(T_2, w)/e(g_1, g_2)\big)^c \ . \end{aligned} \tag{6}$$

3. Check that the challenge c is correct:

$$c \stackrel{?}{=} H(gpk, M, r, T_1, T_2, \tilde{R_1}, \tilde{R_2}, \tilde{R_3}) \ . \tag{7}$$

If it is, accept. Otherwise, reject.

2. Revocation Check. For each element $A \in RL$, check whether A is encoded in (T_1, T_2) by checking if:

$$e(T_2/A, \hat{u}) \stackrel{?}{=} e(T_1, \hat{v}) \ .$$

If no element of RL is encoded in (T_1, T_2), the signer of σ has not been revoked.

The algorithm outputs `valid` if both phases accept, `invalid` otherwise.

Signature Length. A group signature in the system above comprises two elements of G_1 and five elements of $\mathbb{Z}_p$. When using the curves described in [8] one can take p to be a 170-bit prime and use a group G_1 where each element is 171 bits. Thus, the total group signature length is 1192 bits or 149 bytes. With these parameters, security is approximately the same as a standard 1024-bit RSA signature, which is 128 bytes.

Performance. Signature generation requires two applications of the isomorphism ψ. Computing the isomorphism takes roughly the same time as an exponentiation in G_1 (using fast computations of the trace map). Thus, signature generation requires about 8 exponentiations (or multi-exponentiations) and 2 bilinear map computations. Signature verification takes 6 exponentiations and $3+2|RL|$ computations of the bilinear map. A far more efficient revocation check algorithm, whose running time is independent of $|RL|$, is described in Section 7.

We now prove the correctness of the VLR group signature scheme. The proofs of the selfless-anonymity and traceability of the scheme are given in Section 6.

THEOREM 1. *The VLR group signature scheme is correct, as defined in equation (1).*

PROOF. Consider public parameters $gpk = (g_1, g_2, w)$; secret-key vector $\boldsymbol{gsk}$ where, for each i, $\boldsymbol{gsk}[i] = (A_i, x_i)$, an SDH tuple, i.e., a tuple satisfying $e(A_i, wg_2^{x_i}) = e(g_1, g_2)$; and revocation-token list $\boldsymbol{grt}$ where $\boldsymbol{grt}[i] = A_i$, as output by the key generation algorithm.

An honest signer with private key (A_i, x_i) generates a signature $\sigma = (r, T_1, T_2, c, s_\alpha, s_x, s_\delta)$ by following the signing algorithm described above. In particular, the signer computes the generators $\hat{u}$ and $\hat{v}$ according to equation (2), so the verifier uses the same generators. Now, the first phase of the signature verification algorithm accepts a signature if the output of H equals the challenge c. This will only be true (except with negligible probability) when all inputs to H are exactly the same for the verifier as for the signer. An honest signer's signature includes all these inputs except R_1, R_2, and R_3, which are rederived by the verifier. We must therefore show that the values rederived by the verifier using equations (6) equal those derived by the signer using equations (4). First,

$$\tilde{R_1} = u^{s_\alpha}/T_1^c = u^{r_\alpha + c\alpha}/(u^\alpha)^c = u^{r_\alpha} = R_1 \ ,$$

so $\tilde{R_1} = R_1$. Further,

$$\begin{aligned} \tilde{R_3} = T_1^{s_x} u^{-s_\delta} &= (u^\alpha)^{r_x + cx_i} \cdot u^{-r_\delta - cx_i\alpha} \\ &= (u^\alpha)^{r_x} \cdot u^{-r_\delta} = T_1^{r_x} \cdot u^{-r_\delta} = R_3 \ , \end{aligned}$$

so $\tilde{R_3} = R_3$. Finally,

$$\begin{aligned} \tilde{R_2} &= e(T_2, g_2)^{s_x} \cdot e(v, w)^{-s_\alpha} \cdot e(v, g_2)^{-s_\delta} \cdot \left(\frac{e(T_2, w)}{e(g_1, g_2)}\right)^c \\ &= \Big(e(T_2, g_2)^{r_x} \cdot e(v, w)^{-r_\alpha} \cdot e(v, g_2)^{-r_\delta}\Big) \\ &\quad \cdot \left(e(T_2, g_2)^{x_i} \cdot e(v, w)^{-\alpha} \cdot e(v, g_2)^{-x_i\alpha} \cdot \frac{e(T_2, w)}{e(g_1, g_2)}\right)^c \\ &= R_2 \cdot \left(\frac{e(T_2 v^{-\alpha}, wg_2^{x_i})}{e(g_1, g_2)}\right)^c = R_2 \cdot \left(\frac{e(A_i, wg_2^{x_i})}{e(g_1, g_2)}\right)^c = R_2 \ , \end{aligned}$$

so $\tilde{R_2} = R_2$. The last equality follows from the SDH equation. Thus (7) will be satisfied.

In a signature generated by the signing algorithm, we have $T_1 = \psi(\hat{u})^\alpha$ and $T_2 = A_i\psi(\hat{v})^\alpha$ for some α. The revocation check algorithm will reject a signature as originating from a revoked user with token A exactly when $(\hat{u}, \hat{v}, T_1, T_2/A)$ is a co-Diffie-Hellman tuple, i.e., when A equals A_i. Thus the group signature verification algorithm will accept a signature as valid exactly when A_i is not included in its input RL, as required. $\square$

5. INTUITION

The VLR group signature scheme presented in Section 4 above is derived, via a variant of the Fiat-Shamir heuristic [13], from a new protocol for proving possession of an SDH tuple. We present this protocol below to give intuition into the construction of our VLR group signature scheme.

The protocol is a proof of knowledge, which means that by rewinding a prover it is possible to extract an SDH pair. The protocol is intentionally *not* zero-knowledge; a verifier in possession of a revocation token can determine whether he is interacting with a revoked prover.

The public values are $g_1 \in G_1$ and $g_2, w \in G_2$. Here g_2 is a random generator of G_2, g_1 equals $\psi(g_2)$, and w equals g_2^γ for some (secret) $\gamma \in \mathbb{Z}_p$. The prover wishes to demonstrate possession of a pair (A, x), where $A \in G_1$ and $x \in \mathbb{Z}_p$, such that $A^{x+\gamma} = g_1$. Such a pair satisfies $e(A, wg_2^x) = e(g_1, g_2)$. We use a generalization of Schnorr's protocol for proving knowledge of discrete logarithm [18] in a group of prime order.

Protocol 1. Bob, the verifier, selects elements $\hat{u}$ and $\hat{v}$ uniformly at random from G_2 and sends them to Alice, the prover. Alice sets $u \leftarrow \psi(\hat{u})$ and $v \leftarrow \psi(\hat{v})$. She selects exponent $\alpha \xleftarrow{R} \mathbb{Z}_p$, and computes

$$T_1 \leftarrow u^\alpha \qquad \text{and} \qquad T_2 \leftarrow Av^\alpha \ .$$

Alice and Bob then undertake a proof of knowledge of values (α, x, δ) satisfying the following three relations:

$$u^\alpha = T_1 \ , \qquad T_1^x = u^\delta \ , \qquad e(T_2 v^{-\alpha}, wg_2^x) = e(g_1, g_2) \ .$$

This proof of knowledge proceeds as follows. Alice computes a helper value $\delta = x\alpha$. She then picks blinding values r_α, r_x, and r_δ at random from $\mathbb{Z}_p$. She computes three values based on all these:

$$R_1 \leftarrow u^{r_\alpha} \qquad R_3 \leftarrow T_1^{r_x} \cdot u^{-r_\delta}$$
$$R_2 \leftarrow e(T_2, g_2)^{r_x} \cdot e(v, w)^{-r_\alpha} \cdot e(v, g_2)^{-r_\delta} \ .$$

She then sends $(T_1, T_2, R_1, R_2, R_3)$ to Bob. Bob sends a challenge value c chosen uniformly at random from $\mathbb{Z}_p$. Alice computes and sends back $s_\alpha = r_\alpha + c\alpha$, $s_x = r_x + cx$, and $s_\delta = r_\delta + c\delta$. Finally, Bob verifies the following three equations:

$$u^{s_\alpha} \stackrel{?}{=} T_1^c \cdot R_1 \tag{8}$$

$$e(T_2, g_2)^{s_x} \cdot e(v, w)^{-s_\alpha} \cdot e(v, g_2)^{-s_\delta} \stackrel{?}{=} \big(e(g_1, g_2)/e(T_2, w)\big)^c \cdot R_2 \tag{9}$$

$$T_1^{s_x} u^{-s_\delta} \stackrel{?}{=} R_3 \ . \tag{10}$$

Bob accepts if all three hold. Applying a standard variant of the Fiat-Shamir heuristic to this protocol produces the signature scheme of the previous section.

The protocol above is (by design) not a zero-knowledge protocol. Given (T_1, T_2) and a candidate A, anyone can check whether A is ElGamal-encrypted in (T_1, T_2) by checking whether $e(T_2/A, \hat{u}) \stackrel{?}{=} e(T_1, \hat{v})$. Below, however, we show that the protocol has an extractor and, given a (T_1, T_2) pair, can be simulated. The correctness of the protocol follows from Theorem 1.

LEMMA 1. *For any $(\hat{u}, \hat{v}, T_1, T_2)$, Transcripts of Protocol 1 can be simulated.*

PROOF. Choose challenge $c \xleftarrow{R} \mathbb{Z}_p$. Select $s_\alpha \xleftarrow{R} \mathbb{Z}_p$, and set $R_1 \leftarrow u^{s_\alpha}/T_1^c$. Then equation (8) is satisfied. With α and c fixed, a choice for either of r_α or s_α determines the other, and a uniform random choice of one gives a uniform random choice of the other. Therefore s_α and R_1 are distributed as in a real transcript.

Select $s_x \xleftarrow{R} \mathbb{Z}_p$. Now, A and α are fixed by T_1 and T_2, x is implicitly fixed by the SDH equation for A, r_x is fixed by x and s_x, and δ is fixed as $x\alpha$. Select $s_\delta \xleftarrow{R} \mathbb{Z}_p$; a uniform distribution on this gives a uniform distribution on r_δ. Set $R_3 \leftarrow T_1^{s_x} u^{-s_\delta}$. Again, all the computed values are distributed as in a real transcript. Finally, set

$$R_2 \leftarrow e(T_2, g_2)^{s_x} \cdot e(v, w)^{-s_\alpha} \cdot e(v, g_2)^{-s_\delta} \cdot \left(\frac{e(T_2, w)}{e(g_1, g_2)}\right)^c \ .$$

This R_2 satisfies (9), so it, too, is properly distributed.

Finally, the simulator outputs the transcript $(\hat{u}, \hat{v}, T_1, T_2, R_1, R_2, R_3, c, s_\alpha, s_\beta, s_x, s_\delta)$. As argued above, this transcript is distributed identically to transcripts of Protocol 1 for the given $(\hat{u}, \hat{v}, T_1, T_2)$. $\square$

LEMMA 2. *There exists an extractor for Protocol 1 that extracts an SDH pair from a convincing prover.*

PROOF. Suppose that an extractor can rewind a prover in the protocol above. The verifier sends $\hat{u}, \hat{v}$ to the prover. Let $u = \psi(\hat{u})$ and $v = \psi(\hat{v})$. The prover then sends T_1, T_2 and R_1, R_2, R_3. To challenge value c, the prover responds with s_α, s_x, and s_δ. To challenge value $c' \neq c$, the prover responds with s'_α, s'_x, and s'_δ. If the prover is convincing, all three verification equations hold for each set of values.

For brevity, let $\Delta c = c - c'$, $\Delta s_\alpha = s_\alpha - s'_\alpha$, and similarly for Δs_x, and Δs_δ.

Consider (8) above. Dividing the two instances of this equation, we obtain $u^{\Delta s_\alpha} = T_1^{\Delta c}$. The exponents are in a group of known prime order, so we can take roots; let $\tilde{\alpha} = \Delta s_\alpha / \Delta c$. Then $u^{\tilde{\alpha}} = T_1$.

Now consider (10) above. Dividing the two instances gives $T_1^{\Delta s_x} = u^{\Delta s_\delta}$. Substituting $T_1 = u^{\tilde{\alpha}}$ gives $u^{\tilde{\alpha}\Delta s_x} = u^{\Delta s_\delta}$, or $\Delta s_\delta = \tilde{\alpha}\Delta s_x$.

Finally, dividing the two instances of (9), we obtain

$$\big(e(g_1, g_2)/e(T_2, w)\big)^{\Delta c} = e(T_2, g_2)^{\Delta s_x} \cdot e(v, w)^{-\Delta s_\alpha} \cdot e(v, g_2)^{-\tilde{\alpha}\Delta s_x} \ .$$

Taking Δc-th roots, and letting $\tilde{x} = \Delta s_x / \Delta c$, we obtain

$$e(g_1, g_2)/e(T_2, w) = e(T_2, g_2)^{\tilde{x}} \cdot e(v, w)^{-\tilde{\alpha}} \cdot e(v, g_2)^{-\tilde{x}\tilde{\alpha}} \ .$$

This can be rearranged as

$$e(g_1, g_2) = e(T_2 v^{-\tilde{\alpha}}, wg_2^{\tilde{x}}) \ ,$$

or, letting $\tilde{A} = T_2 v^{-\tilde{\alpha}}$,

$$e(\tilde{A}, wg_2^{\tilde{x}}) = e(g_1, g_2) \ .$$

Thus the extractor obtains an SDH tuple $(\tilde{A}, \tilde{x})$. Moreover, the $\tilde{A}$ in this SDH tuple is, perforce, the same as that in the ElGamal encryption (T_1, T_2). In other words, the extractor recovers the same A that a revocation-checker matches. $\square$

6. PROOF OF SECURITY

We show that the scheme described in Section 4 is a VLR group signature scheme. The correctness of the scheme was demonstrated in Theorem 1, above. Below we prove the selfless-anonymity and traceability of the scheme, as defined in Section 2.

6.1 Selfless-Anonymity

LEMMA 3. *The VLR group signature scheme in (G_1, G_2) has $(t, q_H, q_S, n, \epsilon)$ selfless anonymity in the random oracle model assuming the (t, ϵ') Decision Linear assumption holds in the group G_2 for $\epsilon' = \frac{\epsilon}{2}\left(\frac{1}{n^2} - \frac{q_S q_H}{p}\right) \approx \epsilon/2n^2$.*

PROOF. Suppose algorithm $\mathcal{A}$ $(t, q_H, q_S, n, \epsilon)$-breaks the selfless anonymity of the VLR group signature scheme. We build an algorithm $\mathcal{B}$ that breaks the Decision Linear assumption in G_2. Algorithm $\mathcal{B}$ is given as input a 6-tuple $(u_0, u_1, v, h_0 = u_0^a, h_1 = u_1^b, Z) \in G_2^6$ where $u_0, u_1, v \xleftarrow{R} G_2$, $a, b \xleftarrow{R} \mathbb{Z}_p^*$ and either $Z = v^{a+b} \in G_2$ or Z is random in G_2. Algorithm $\mathcal{B}$ decides which Z it was given by interacting with $\mathcal{A}$ as follows:

Setup. Recall that g_1, g_2 are the fixed generators of G_1, G_2 respectively. $\mathcal{B}$ does the following:

1. $\mathcal{B}$ picks a random $\gamma \xleftarrow{R} \mathbb{Z}_p$ and sets $w = g_2^\gamma$. It gives $\mathcal{A}$ the $gpk = (g_1, g_2, w)$.
2. $\mathcal{B}$ picks two random users $i_0, i_1 \xleftarrow{R} \{1, \ldots, n\}$ and keeps i_0, i_1 secret. For all users $j \neq i_0, i_1$ it generates private keys $GSK[j] = (A_j, x_j)$ using γ and the standard key generation algorithm.
3. $\mathcal{B}$ picks a random $W \xleftarrow{R} G_2$.

To give some intuition for the rest of the simulation we define $A_{i_0} = ZW/v^a$ and $A_{i_1} = Wv^b$. In what follows, $\mathcal{B}$ will behave as if user i_0's private key is (A_{i_0}, x_{i_0}) and user i_1's private key is (A_{i_1}, x_{i_1}) for some $x_{i_0}, x_{i_1} \in \mathbb{Z}_p$. We emphasize that $\mathcal{B}$ does not know either A_{i_0} or A_{i_1} since it doesn't know a or b. Observe that if $Z = v^{a+b}$ then

$$A_{i_0} = ZW/v^a = v^{a+b}W/v^a = Wv^b = A_{i_1}$$

Hence, if $Z = v^{a+b}$ users i_0 and i_1 have the same private key. But if Z is random in G_2 then i_0, i_1 have independent private keys.

Hash queries. At any time, $\mathcal{A}$ can query the hash functions H and H_0. Algorithm $\mathcal{B}$ responds with random values while ensuring consistency.

Phase 1. Algorithm $\mathcal{A}$ can issue signing queries, corruption queries, and revocation queries. If a query is for user $i \neq i_0, i_1$ then $\mathcal{B}$ uses the private key $\boldsymbol{gsk}[i]$ to respond to the query. For queries for users i_0 or i_1 algorithm $\mathcal{B}$ responds as follows:

- Signing queries: given a message $M \in \{0,1\}^*$ and a user $i \in \{i_0, i_1\}$ algorithm $\mathcal{B}$ must generate a signature for M using user i's private key.
  - To generate a signature for user $i = i_0$, $\mathcal{B}$ picks random $s, t, \ell \xleftarrow{R} \mathbb{Z}_p^*$ and sets:

    $$T_1 \leftarrow h_0 u_0^s \ , \qquad T_2 \leftarrow ZWv^s h_0^t u_0^{st} \ ,$$
    $$\hat{u} \leftarrow u_0^\ell \ , \qquad \hat{v} \leftarrow (vu_0^t)^\ell \ .$$

    Let $\alpha = (a+s)/\ell \in \mathbb{Z}_p$. Then $T_1 = \hat{u}^\alpha$ and $T_2 = A_{i_0} \cdot \hat{v}^\alpha$.
  - To generate a signature for user $i = i_1$, $\mathcal{B}$ picks random $s, t, \ell \xleftarrow{R} \mathbb{Z}_p^*$ and sets

    $$T_1 \leftarrow h_1 u_1^s \ , \qquad T_2 \leftarrow Wh_1^t u_1^{st}/v^s \ ,$$
    $$\hat{u} \leftarrow u_1^\ell \ , \qquad \hat{v} \leftarrow (u_1^t/v)^\ell \ .$$

    Let $\alpha = (b+s)/\ell \in \mathbb{Z}_p$. Then $T_1 = \hat{u}^\alpha$ and $T_2 = A_{i_1} \cdot \hat{v}^\alpha$.

  Either way, $T_1 = \hat{u}^\alpha$ and $T_2 = A_i\hat{v}^\alpha$ for some random $\alpha \in \mathbb{Z}_p$ and random and independent $\hat{u}, \hat{v} \in G_2$. Algorithm $\mathcal{B}$ next picks random $r, c, s_\alpha, s_x, s_\delta \xleftarrow{R} \mathbb{Z}_p$ and computes the corresponding R_1, R_2, R_3 using equations (6). In the unlikely event that $\mathcal{A}$ has already issued a hash query either for $H(gpk, M, r, \psi(T_1), \psi(T_2), R_1, R_2, R_3)$ or for $H_0(gpk, M, r)$, $\mathcal{B}$ reports failure and terminates. Since r is random in $\mathbb{Z}_p$ this happens with probability at most q_H/p. Otherwise, $\mathcal{B}$ defines

  $$\begin{aligned} H(gpk, M, r, \psi(T_1), \psi(T_2), R_1, R_2, R_3) &= c \\ H_0(gpk, M, r) &= (\hat{u}, \hat{v}). \end{aligned}$$

  Algorithm $\mathcal{B}$ then computes the signature σ as $\sigma \leftarrow (r, \psi(T_1), \psi(T_2), c, s_\alpha, s_x, s_\delta)$, and gives σ to $\mathcal{A}$. Note that by Lemma 1, σ is a properly distributed signature under user i's private key.
- Corruption queries and revocation queries: if $\mathcal{A}$ ever issues a corruption of revocation query for users i_0 or i_1 then $\mathcal{B}$ reports failure and aborts.

Challenge Algorithm $\mathcal{A}$ outputs a message M and two users i_0^* and i_1^* where it wishes to be challenged. if $\{i_0^*, i_1^*\} \neq \{i_0, i_1\}$ then $\mathcal{B}$ reports failure and aborts. Otherwise, we assume $i_0^* = i_0$ and $i_1^* = i_1$. Algorithm $\mathcal{B}$ picks a random $b \xleftarrow{R} \{0,1\}$ and generates a signature σ^* under user i_b's key for M using the same method used to respond to signing queries in Phase 1. It gives σ^* as the challenge to $\mathcal{A}$.

Phase 2. Algorithm $\mathcal{A}$ issues restricted queries. $\mathcal{B}$ responds as in Phase 1.

Output. Eventually, $\mathcal{A}$ outputs its guess $b' \in \{0,1\}$ for b. If $b = b'$ then $\mathcal{B}$ outputs 0 (indicating that Z is random in G_2); otherwise $\mathcal{B}$ outputs 1 (indicating that $Z = v^{a+b}$).

Suppose $\mathcal{B}$ does not abort during the simulation. Then, when Z is random in G_2 algorithm $\mathcal{B}$ emulates the selfless-anonymity game perfectly. Hence, $\Pr[b = b'] > \frac{1}{2} + \epsilon$. When $Z = v^{a+b}$ then the private keys for users i_0 and i_1 are identical and therefore the challenge signature σ^* is independent of b. It follows that $\Pr[b = b'] = 1/2$. Therefore, assuming $\mathcal{B}$ does not abort, it has advantage at least $\epsilon/2$ in solving the given linear challenge $(u_0, u_1, v, h_0, h_1, Z) \in G_2^6$.

Algorithm $\mathcal{B}$ does not abort if it correctly guesses the values i_0^* and i_1^* during the setup phase and none of the signature queries cause it to abort. The probability that a given signature query causes $\mathcal{B}$ to abort is at most q_H/p and therefore the probability that $\mathcal{B}$ aborts as a result of $\mathcal{A}$'s signature queries is at most $q_S q_H/p$. As long as $\mathcal{B}$ does not abort during phase 1, algorithm $\mathcal{A}$ gets no information about the choice of i_0, i_1. Therefore, the probability that the query pattern during phase 1 and the choice of challenge do not cause $\mathcal{B}$ to abort is at least $1/n^2$. It now follows that $\mathcal{B}$ solves the given linear challenge with advantage at least $\frac{\epsilon}{2}\left(\frac{1}{n^2} - \frac{q_S q_H}{p}\right)$, as required. $\square$

6.2 Traceability

THEOREM 2. *If SDH is (q, t', ϵ')-hard on (G_1, G_2), then the VLR group signature scheme is $(t, q_H, q_S, n, \epsilon)$-traceable, where $n = q - 1$, $\epsilon = 4n\sqrt{2\epsilon' q_H} + n/p$, and $t = \Theta(1) \cdot t'$.*

PROOF. Our proof proceeds in three parts. First, we describe a framework for interacting with an algorithm that wins a traceability game. Second, we show how to instantiate this framework appropriately for different types of such breaker algorithms. Third, we show how to apply the forking lemma [17] to the framework instances, obtaining SDH solutions.

Interaction Framework. Suppose we are given an algorithm $\mathcal{A}$ that breaks the traceability of the VLR group signature scheme. We describe a framework for interacting with $\mathcal{A}$.

Setup. We are given groups (G_1, G_2) as above, with respective generators g_1 and g_2. We are also given $w = g_2^\gamma \in G_2$, and a list of pairs (A_i, x_i) for $i = 1, \ldots, n$. For each i, either $x_i = \star$, indicating that the x_i corresponding to A_i is not known, or else (A_i, x_i) is an SDH pair, and $e(A_i, wg_2^{x_i}) = e(g_1, g_2)$. We then run $\mathcal{A}$, giving it the group public key (g_1, g_2, w) and the users' revocation tokens $\{A_i\}$. We answer its oracle queries as follows.

Hash Queries. At any time, $\mathcal{A}$ can query the hash functions to obtain generators $(\hat{u}, \hat{v})$ or challenge c. We respond with random values while maintaining consistency. made again.

Signature Queries. Algorithm $\mathcal{A}$ asks for a signature on message M by a key at index i. If $s_i \neq \star$, we follow the group signing procedure with key (A_i, x_i) to obtain a signature σ on M, and return σ to $\mathcal{A}$.

Otherwise $s_i = \star$. We pick a nonce $r \xleftarrow{R} \mathbb{Z}_p$, obtain generators $(\hat{u}, \hat{v}) \leftarrow H_0(gpk, M, r)$, and set $u \leftarrow \psi(\hat{u})$ and $v \leftarrow \psi(\hat{v})$. We then pick $\alpha \xleftarrow{R} \mathbb{Z}_p$, set $T_1 \leftarrow u^\alpha$, and $T_2 \leftarrow Ag_1^\alpha$ and run the Protocol 1 simulator with values $(\hat{u}, \hat{v}, T_1, T_2)$. The simulator returns a transcript $(\hat{u}, \hat{v}, T_1, T_2, R_1, R_2, R_3, c, s_\alpha, s_x, s_\delta)$, from which we derive a VLR group signature $\sigma = (r, T_1, T_2, c, s_\alpha, s_x, s_\delta)$. In addition, we must patch the hash oracle at $(M, r, T_1, T_2, R_1, R_2, R_3)$ to equal c. If this causes a collision, i.e., if we previously set the oracle at this point to some other c', we declare failure and exit. Otherwise, we return σ to $\mathcal{A}$. A signature query can trigger a hash query, which we charge against $\mathcal{A}$'s hash query limit to simplify the accounting.

Private Key Queries. Algorithm $\mathcal{A}$ asks for the private key of the user at some index i. If $x_i \neq \star$, we return (A_i, x_i) to $\mathcal{A}$. Otherwise, we declare failure and exit.

Output. Finally, if algorithm $\mathcal{A}$ is successful, it outputs a forged VLR group signature $\sigma = (r, T_1, T_2, c, s_\alpha, s_x, s_\delta)$ on a message M with nonce r, along with a revocation list RL^*. We apply the implicit revocation algorithm, with revocation tokens $\{A_i\}$ to determine which A^* is encoded in (T_1, T_2). This A^* cannot be on RL^*; if it were, the signature would have been rejected as invalid. Thus for the forgery to be nontrivial, A^* must also be outside the adversary's coalition U. If A^* does not equal A_i for any i, we output σ. Otherwise, $A^* = A_{i^*}$ for some i^*. If $s_{i^*} = \star$, we output σ. If, however, $s_{i^*} \neq \star$, we declare failure and exit.

As implied by the output phase of the framework above, there are two types of forger algorithm. Type I forgers output a forgery σ on a message M that encodes some identity $A^* \notin \{A_1, \ldots, A_n\}$. Type II forgers output a forgery that encodes an identity A^* such that $A^* = A_{i^*}$ for some i^*, and the forger did not make a private-key oracle query at i^*. We treat these two types of forger differently.

Given a q-SDH instance $(g_1', g_2', (g_2')^\gamma, (g_2')^{\gamma^2}, \ldots, (g_2')^{\gamma^q})$, we apply the technique of Boneh and Boyen's Lemma 3.2 [6], obtaining generators $g_1 \in G_1$, $g_2 \in G_2$, $w = g_2^\gamma$, along with $q-1$ SDH pairs (A_i, x_i) such that $e(A_i, wg_2^{x_i}) = e(g_1, g_2)$ for each i. Any SDH pair (A, x) besides these $q-1$ pairs can be transformed into a solution to the original q-SDH instance, again using Boneh and Boyen's Lemma 3.2.

Type I Forger. Against a $(t, q_H, q_S, n, \epsilon)$-Type I forger $\mathcal{A}$, we turn an instance of $(n+1)$-SDH into values (g_1, g_2, w), and n SDH pairs (A_i, x_i). We then apply the framework to $\mathcal{A}$ with these values. Algorithm $\mathcal{A}$'s environment is perfectly simulated, and the framework succeeds whenever $\mathcal{A}$ succeeds, so we obtain a Type I forgery with probability ϵ.

Type II Forger. Against a $(t, q_H, q_S, n, \epsilon)$-Type II forger $\mathcal{A}$, we turn an instance of n-SDH into values (g_1, g_2, w), and $n-1$ SDH pairs. These pairs we distribute amongst n pairs (A_i, x_i). The unfilled entry at random index i^* we fill as follows. Pick $A_{i^*} \xleftarrow{R} G_1$, and set $x_{i^*} \leftarrow \star$, a placeholder value. Now we run $\mathcal{A}$ under the framework. The framework declares success only if $\mathcal{A}$ never queries the private key oracle at i^*, but forges a group signature that traces to A_{i^*}. It is easy to see that the framework simulation is perfect unless $\mathcal{A}$ queries the private key oracle at i^*. Because the protocol simulator invoked by the signing oracle produces group signatures that are indistinguishable from those of a user whose SDH tuple includes A_{i^*}, the value of i^* is independent of $\mathcal{A}$'s view unless and until it queries the private key oracle at i^*. (Since the hash oracle takes as input five elements of G_1 or G_2 besides the message M, the probability of collision in simulated signing queries is bounded above by $(q_H q_S + q_S^2)/p^9$. Assuming $q_S \ll q_H \ll p = |G_1|$, this probability is negligible, and we ignore it in the analysis.) Finally, when $\mathcal{A}$ outputs its forgery σ, implicating some user i whose private key $\mathcal{A}$ has not requested, the value of i^* (amongst the users whose keys it has not requested) remains independent of $\mathcal{A}$'s view. It is easy to see, then, that $\mathcal{A}$ outputs a forged group signature that traces to user i^* with probability at least ϵ/n.

Application of Forger. Now we show how to use the application of our framework to a Type I or Type II adversary $\mathcal{A}$ to obtain another SDH pair, contradicting the SDH assumption. The remainder of this proof follows closely the methodology and notation of the forking lemma [17].

Let $\mathcal{A}$ be a forger (of either type) for which the framework succeeds with probability ϵ'. From here on, we abbreviate signatures as $(M, \sigma_0, c, \sigma_1)$, where $\sigma_0 = (r, \hat{u}, \hat{v}, T_1, T_2, R_1, R_2, R_3)$, the values given, along with M, to the random oracle H, and from which c is derived, and where $\sigma_1 = (s_\alpha, s_x, s_\delta)$. (Those values normally omitted from the signature can be recovered as the verification algorithm in Section 4 does.)

We require that $\mathcal{A}$ always query H_0 at (M, r) before querying H at $(M, r, \ldots)$. Any adversary can be modified mechanically into satisfying this condition. This technical requirement means that, even if in rewinding we change the value of $H(M, r, \ldots)$, the value of $H_0(M, r)$, and therefore of the u and v used implicitly in the arguments of the H call, remains unchanged.

For any fixed f_0 vector of H_0 responses, a run of the framework on $\mathcal{A}$ is completely described by the randomness string ω used by the framework and $\mathcal{A}$, by the vector f_0 of responses made by the H_0 hash oracle, and by the vector f of responses made by the H hash oracle. Let S be the set of tuples (ω, f_0, f) such that the framework, invoked on $\mathcal{A}$, completes successfully with forgery $(M, \sigma_0, c, \sigma_1)$, and $\mathcal{A}$ queried the hash oracle H on (M, σ_0). In this case, let $\mathrm{Ind}(\omega, f_0, f)$ be the index of f at which $\mathcal{A}$ queried (M, σ_0). We define $\nu = \Pr[S] = \epsilon' - 1/p$, where the $1/p$ term accounts for the possibility that $\mathcal{A}$ guessed the hash of (M, σ_0) without the hash oracle's help. For each j, $1 \le j \le q_H$, let S_j be the set of tuples (ω, f_0, f) as above and such that $\mathrm{Ind}(\omega, f_0, f) = j$. Let J be the set of auspicious indices j such that $\Pr[S_j \mid S] \ge 1/(2q_H)$. Then $\Pr[\mathrm{Ind}(\omega, f) \in J \mid S] \ge 1/2$.

Let $f|_a^b$ be the restriction of f to its elements at indices $a, a+1, \ldots, b$. For each $j \in J$, we consider the heavy-rows lemma [17, Lemma 1] with rows $X = (\omega, f_0, f|_1^{j-1})$ and columns $Y = (f|_j^{q_H})$. Clearly $\Pr_{(x,y)}[(x, y) \in S_j] \ge \nu/(2q_H)$. Let the heavy rows Ω_j be those rows such that, $\forall (x, y) \in \Omega_j : \Pr_{y'}[(x, y') \in S_j] \ge \nu/(4q_H)$. Then, by the heavy-rows lemma, $\Pr[\Omega_j \mid S_j] \ge 1/2$. A simple argument then shows that $\Pr[\exists j \in J : \Omega_j \cap S_j \mid S] \ge 1/4$.

Thus, with probability $\nu/4$, the framework, invoked on $\mathcal{A}$, succeeds and obtains a forgery $(M, \sigma_0, c, \sigma_1)$ that derives from a heavy row $(x, y) \in \Omega_j$ for some $j \in J$, i.e., an execution (ω, f_0, f) such that

$$\Pr_{f'}\left[(\omega, f_0, f') \in S_j \mid f'|_1^{j-1} = f|_1^{j-1}\right] \ge \nu/(4q_H) \ .$$

If we now rewind the framework and $\mathcal{A}$ to the jth query and proceed with an oracle vector f' that differs from f from the jth entry on, we obtain, with probability at least $\nu/(4q_H)$, a successful framework completion and a second forgery $(M, \sigma_0, c', \sigma_1')$, with (M, σ_0) still queried at $\mathcal{A}$'s jth hash query. Since the adversary queried H_0 at (M, r) (where r is the first element of σ_0) before he made his jth H oracle query, the values of $\hat{u}$ and $\hat{v}$ in these two forgeries will be the same.

By using the extractor of Lemma 2, we obtain from the forgeries (σ_0, c, σ_1) and $(\sigma_0, c', \sigma_1')$ an SDH tuple (A, x). The extracted A is the same as the A encoded in (T_1, T_2) in σ_0. The framework declares success only when the A encoded in (T_1, T_2) is not amongst those whose x it knows. Therefore, the extracted SDH tuple (A, x) is not amongst those that we ourselves created, and can be transformed, again following the technique of Boneh and Boyen's Lemma 3.2 [6], to an answer to the posed q-SDH problem.

Putting everything together, we have proved the following claims.

Claim 1. Using a $(t, q_H, q_S, n, \epsilon)$ Type I forger, we solve an instance of $(n+1)$-SDH with probability $(\epsilon - 1/p)^2/(16q_H)$ in time $\Theta(1) \cdot t$.

Claim 2. Using a $(t, q_H, q_S, n, \epsilon)$ Type II forger, we solve an instance of n-SDH with probability $(\epsilon/n - 1/p)^2/(16q_H)$ in time $\Theta(1) \cdot t$.

We can guess which of the two forger types a particular forger is with probability 1/2; then assuming the more pessimistic scenario of Claim 2 proves the theorem. $\square$

7. EFFICIENT REVOCATION

In our VLR group signature scheme (Section 4), signature verification time grows linearly in the number of revoked users. It is desirable to have a Verifier-Local Revocation system where verification time is constant. In this section we describe a simple modification to the signing and verification algorithms that achieves this at the cost of slightly reduced anonymity.

Consider how our system is used for privacy-preserving attestation: Users connect to various web sites and at each site they perform a private attestation using the group signature issued by the tamper-resistant chip in their machine. For an efficient revocation check, when the chip issues a signature for attesting to a site S it uses the signing algorithm from Section 4 with the small modification that the parameters u and v are generated as:

$$(u, v) \overset{\mathrm{R}}{\leftarrow} H_0(gpk, S, r)$$

where r is random in the range $\{1, \ldots, k\}$ and k is a security parameter (e.g., $k = 128$). Note that, unlike Section 4, here (u, v) do not depend on the message being signed. Hence, at a given site S there are only k possible values for the pair (u, v).

Now, suppose site S has been supplied with a revocation list $RL = \{A_1, \ldots, A_b\}$. To verify that a signature $\sigma = (r, T_1, T_2, c, s_\alpha, s_x, s_\delta)$ was not issued by a revoked user the site uses the same procedure as in Section 4:

1. Compute $(u, v) \overset{\mathrm{R}}{\leftarrow} H_0(gpk, S, r)$, and
2. For $i = 1, \ldots, b$ check that $e(T_1, v)e(A_i, u) \ne e(T_2, u)$.

Since at site S there are only k possible values for u, the value $e(A_i, u)$ can be pre-computed for the entire RL for all possible u's. Thus, site S stores a $|RL| \times k$ table of values, $e(A_i, u_j)$. To check revocation, it simply does a table-lookup to see if the value $e(T_2, u)/e(T_1, v)$ is in the r'th row of the table. If not, then the signature was not issued by a revoked user. Hence, the revocation check takes time independent of the size of RL.

The downside is that the scheme is now only partially anonymous. If the user issues two signatures at site S using the same random value $r \in \{1, \ldots, k\}$ then the site can test that these two signatures came from the same user. However, signatures issued at different sites are still completely unlinkable. Similarly, signatures issued at the same site using different r's are unlinkable (e.g., with $k = 100$ only 1% of signatures at S are linkable). For some applications, this trade-off between partial linkability and efficient revocation might be acceptable.

8. CONCLUSIONS AND OPEN PROBLEMS

We have described a short group signature scheme where user revocation only requires sending revocation information to signature verifiers. Our signatures are short: only 149 bytes for a standard security level. They are shorter than group signatures built from the Strong-RSA assumption and are shorter even than BBS short group signatures [7], which do not support verifier-local revocation.

There are still a number of open problems related to VLR signatures. Most importantly, is there an efficient VLR group signature scheme where signature verification time is sub-linear in the number of revoked users, without compromising user privacy?

9. ACKNOWLEDGMENTS

We thank Ernie Brickell for helpful discussions about this work.

10. REFERENCES

[1] G. Ateniese, J. Camenisch, M. Joye, and G. Tsudik. A practical and provably secure coalition-resistant group signature scheme. In M. Bellare, editor, *Proceedings of Crypto 2000*, volume 1880 of *LNCS*, pages 255–70. Springer-Verlag, Aug. 2000.

[2] G. Ateniese, G. Tsudik, and D. Song. Quasi-efficient revocation of group signatures. In M. Blaze, editor, *Proceedings of Financial Cryptography 2002*, Mar. 2002.

[3] N. Baric and B. Pfitzman. Collision-free accumulators and fail-stop signature schemes without trees. In *Proceedings of Eurocrypt 1997*, pages 480–494. Springer-Verlag, May 1997.

[4] M. Bellare, D. Micciancio, and B. Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In E. Biham, editor, *Proceedings of Eurocrypt 2003*, volume 2656 of *LNCS*, pages 614–29. Springer-Verlag, May 2003.

[5] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of CCS 1993*, pages 62–73. ACM Press, Nov. 1993.

[6] D. Boneh and X. Boyen. Short signatures without random oracles. In C. Cachin and J. Camenisch, editors, *Proceedings of Eurocrypt 2004*, LNCS, pages 56–73. Springer-Verlag, May 2004.

[7] D. Boneh, X. Boyen, and H. Shacham. Short group signatures. In M. Franklin, editor, *Proceedings of Crypto 2004*, volume 3152 of *LNCS*, pages 41–55. Springer-Verlag, Aug. 2004.

[8] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. In *Proceedings of Asiacrypt 2001*, volume 2248 of *LNCS*, pages 514–32. Springer-Verlag, Dec. 2001. Full paper: `http://crypto.stanford.edu/~dabo/pubs.html`.

[9] E. Brickell. An efficient protocol for anonymously providing assurance of the container of a private key, Apr. 2003. Submitted to the Trusted Computing Group.

[10] J. Camenisch and A. Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In M. Yung, editor, *Proceedings of Crypto 2002*, volume 2442 of *LNCS*, pages 61–76. Springer-Verlag, Aug. 2002.

[11] D. Chaum and E. van Heyst. Group signatures. In D. W. Davies, editor, *Proceedings of Eurocrypt 1991*, volume 547 of *LNCS*, pages 257–65. Springer-Verlag, Apr. 1991.

[12] X. Ding, G. Tsudik, and S. Xu. Leak-free group signatures with immediate revocation. In T. Lai and K. Okada, editors, *Proceedings of ICDCS 2004*, Mar. 2004.

[13] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In A. M. Odlyzko, editor, *Proceedings of Crypto 1986*, volume 263 of *LNCS*, pages 186–194. Springer-Verlag, Aug. 1986.

[14] A. Kiayias, Y. Tsiounis, and M. Yung. Traceable signatures. In C. Cachin and J. Camenisch, editors, *Proceedings of Eurocrypt 2004*, volume 3027 of *LNCS*, pages 571–89. Springer-Verlag, May 2004.

[15] S. Mitsunari, R. Sakai, and M. Kasahara. A new traitor tracing. *IEICE Trans. Fundamentals*, E85-A(2):481–4, Feb. 2002.

[16] A. Miyaji, M. Nakabayashi, and S. Takano. New explicit conditions of elliptic curve traces for FR-reduction. *IEICE Trans. Fundamentals*, E84-A(5):1234–43, May 2001.

[17] D. Pointcheval and J. Stern. Security arguments for digital signatures and blind signatures. *J. Cryptology*, 13(3):361–96, 2000.

[18] C. Schnorr. Efficient signature generation by smart cards. *J. Cryptology*, 4(3):161–174, 1991.

[19] V. Shoup. Lower bounds for discrete logarithms and related problems. In W. Fumy, editor, *Proceedings of Eurocrypt 1997*, volume 1233 of *LNCS*, pages 256–66. Springer-Verlag, May 1997.

[20] Trusted Computing Group. Trusted Computing Platform Alliance (TCPA) Main Specification, 2003. Online: `www.trustedcomputinggroup.org`.

[21] G. Tsudik and S. Xu. Accumulating composites and improved group signing. In C. S. Laih, editor, *Proceedings of Asiacrypt 2003*, volume 2894 of *LNCS*, pages 269–86. Springer-Verlag, Dec. 2003.