*An extended abstract of this paper appears in M. Franklin, editor, Advances in Cryptology—CRYPTO 2004, volume 3152 of Lecture Notes in Computer Science, pages 41–55, Berlin: Springer-Verlag, 2004. This version is available from the cryptology eprint archive at* `http://eprint.iacr.org/2004/174/`.

# Short Group Signatures

Dan Boneh *
dabo@cs.stanford.edu

Xavier Boyen †
xb@boyen.org

Hovav Shacham
hovav@cs.stanford.edu

**Abstract**

We construct a short group signature scheme. Signatures in our scheme are approximately the size of a standard RSA signature with the same security. Security of our group signature is based on the Strong Diffie-Hellman assumption and a new assumption in bilinear groups called the Decision Linear assumption. We prove security of our system, in the random oracle model, using a variant of the security definition for group signatures recently given by Bellare, Micciancio, and Warinschi.

## 1 Introduction

Group signatures, introduced by Chaum and van Heyst [14], provide anonymity for signers. Any member of the group can sign messages, but the resulting signature keeps the identity of the signer secret. In some systems there is a third party that can trace the signature, or undo its anonymity, using a special trapdoor. Some systems support revocation [12, 4, 31, 16] where group membership can be selectively disabled without affecting the signing ability of unrevoked members. Currently, the most efficient constructions [2, 12, 4] are based on the Strong-RSA assumption introduced by Baric and Pfitzman [5].

In the last two years a number of projects have emerged that require the properties of group signatures. The first is the Trusted Computing effort [30] 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 [11] [19, Sect. 2.2]. Perhaps an even more relevant project is the Vehicle Safety Communications (VSC) system from the Department of Transportation in the U.S. [20]. The system embeds short-range transmitters in cars; these transmit status information to other cars in close proximity. For example, if a car executes an emergency brake, all cars in its vicinity are alerted. To prevent message spoofing, all messages in the system are signed by a tamper-resistant chip in each car. (MACs were ruled out for this many-to-many broadcast environment.) Since VSC messages reveal the speed and location of the car, there is a strong desire to provide user privacy so that the full identity of the car sending each message is kept private. Using group signatures, where the group is the set of all cars, we can maintain privacy while still being able to revoke a signing key in case the tamper resistant chip in a car is compromised. Due to the number of cars transmitting concurrently there is a hard requirement that the length of each signature be under 250 bytes.

*Supported by NSF and the Packard Foundation.
†Voltage Security, Palo Alto.

The two examples above illustrate the need for efficient group signatures. The second example also shows the need for short group signatures. Currently, group signatures based on Strong-RSA are too long for this application.

We construct short group signatures whose length is under 200 bytes that offer approximately the same level of security as a regular RSA signature of the same length. The security of our scheme is based on the Strong Diffie-Hellman (SDH) assumption [8] in groups with a bilinear map. We also introduce a new assumption in bilinear groups, called the Linear assumption, described in Section 3.2. The SDH assumption was recently used by Boneh and Boyen to construct short signatures without random oracles [8]. A closely related assumption was used by Mitsunari et al. [24] to construct a traitor-tracing system. The SDH assumption has similar properties to the Strong-RSA assumption. We use these properties to construct our short group signature scheme. Our results suggest that systems based on SDH are simpler and shorter than their Strong-RSA counterparts.

Our system is based on a new Zero-Knowledge Proof of Knowledge (ZKPK) of the solution to an SDH problem. We convert this ZKPK to a group signature via the Fiat-Shamir heuristic [17] and prove security in the random oracle model. Our security proofs use a variant of the security model for group signatures proposed by Bellare, Micciancio, and Warinschi [6].

Recently, Camenisch and Lysyanskaya [13] proposed a signature scheme with efficient protocols for obtaining and proving knowledge of signatures on committed values. They then derive a group signature scheme using these protocols as building blocks. Their signature scheme is based on the LRSW assumption [23], which, like SDH, is a discrete-logarithm-type assumption. Their methodology can also be applied to the SDH assumption, yielding a different SDH-based group signature.

The SDH group signature we construct is very flexible and we show how to add a number of features to it. In Section 6 we show how to apply the revocation mechanism of Camenisch and Lysyanskaya [12]. In Section 7 we briefly sketch how to add strong exculpability.

## 2 Bilinear Groups

We first review a few concepts related to bilinear maps. We follow the notation of [9]:

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. $\psi$ is a computable isomorphism from $G_2$ to $G_1$, with $\psi(g_2) = g_1$; and
4. $e$ is a computable map $e : G_1 \times G_2 \to G_T$ with the following properties:
   - Bilinearity: 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-degeneracy: $e(g_1, g_2) \neq 1$.

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. [25]. 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 that discrete log in

$G_1$ is as hard as discrete log in $\mathbb{Z}_q^*$ where $q$ is 1020 bits. We will use these groups to construct short group signatures. We note that the bilinear groups of Rubin and Silverberg [27] can also be used.

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

The isomorphism $\psi$ is only needed for the proofs of security. To keep the discussion general, we simply assume that $\psi$ 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 Complexity Assumptions

## 3.1 The Strong Diffie-Hellman Assumption

Let $G_1, G_2$ be cyclic groups 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 $G_2$. Consider the following problem:

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

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

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

**Definition 3.1.** We say that the $(q, t, \epsilon)$-SDH assumption holds in $(G_1, G_2)$ if no $t$-time algorithm has advantage at least $\epsilon$ in solving the $q$-SDH problem in $(G_1, G_2)$.

Occasionally we drop the $t$ and $\epsilon$ and refer to the $q$-SDH assumption rather than the $(q, t, \epsilon)$-SDH assumption. The $q$-SDH assumption was recently used by Boneh and Boyen [8] to construct a short signature scheme without random oracles. To gain confidence in the assumption they prove that it holds in generic groups in the sense of Shoup [29]. The $q$-SDH assumption has similar properties to the Strong-RSA assumption [5]. We use these properties to construct our short group signature scheme. Mitsunari et al. [24] use a related assumption where $x$ is pre-specified rather than chosen by the adversary.

## 3.2 The Decision Linear Diffie-Hellman 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 Problem in $G_1$.** 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 (e.g., when $G_1 = G_2$). More precisely, we define the advantage of an algorithm $\mathcal{A}$ in deciding the Decision Linear problem in $G_1$ as

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

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 $\mathcal{A}$ $(t, \epsilon)$-decides Decision Linear in $G_1$ if $\mathcal{A}$ runs in time at most $t$, and $\mathsf{Adv\,Linear}_{\mathcal{A}}$ is at least $\epsilon$.

**Definition 3.2.** We say that the $(t, \epsilon)$-Decision Linear Assumption (LA) holds in $G_1$ if no $t$-time algorithm has advantage at least $\epsilon$ in solving the Decision Linear problem in $G_1$.

In Section 8 we show that the Decision Linear Assumption holds in generic bilinear groups [29].

### 3.2.1 Linear Encryption

The Decision Linear problem gives rise to the Linear encryption (LE) scheme, a natural extension of ElGamal encryption. Unlike ElGamal encryption, Linear encryption can be secure even in groups where a DDH-deciding algorithm exists. In this scheme, a user's public key is a triple of generators $u, v, h \in G_1$; her private key is the exponents $x, y \in \mathbb{Z}_p$ such that $u^x = v^y = h$. To encrypt a message $M \in G_1$, choose random values $a, b \in \mathbb{Z}_p$, and output the triple $(u^a, v^b, m \cdot h^{a+b})$. To recover the message from an encryption $(T_1, T_2, T_3)$, the user computes $T_3/(T_1^x \cdot T_2^y)$. By a natural extension of the proof of security of ElGamal, LE is semantically secure against a chosen-plaintext attack, assuming Decision-LA holds.

## 4 A Zero-Knowledge Protocol for SDH

We are now ready to present the underlying building block for our group signature scheme. We present a protocol for proving possession of a solution to an SDH problem. The public values are $g_1, u, v, h \in G_1$ and $g_2, w \in G_2$. Here $u, v, h$ are random in $G_1$, $g_2$ is a random generator of $G_2$, $g_1$ equals $\psi(g_2)$, and $w$ equals $g_2^\gamma$ for some (secret) $\gamma \in \mathbb{Z}_p$. The protocol proves 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 standard generalization of Schnorr's protocol for proving knowledge of discrete logarithm in a group of prime order [28].

**Protocol 1.** Alice, the prover, selects exponents $\alpha, \beta \stackrel{\text{R}}{\leftarrow} \mathbb{Z}_p$, and computes a Linear encryption of $A$:

$$T_1 \leftarrow u^\alpha \qquad T_2 \leftarrow v^\beta \qquad T_3 \leftarrow Ah^{\alpha+\beta} \ . \tag{1}$$

She also computes two helper values $\delta_1 \leftarrow x\alpha$ and $\delta_2 \leftarrow x\beta \in \mathbb{Z}_p$.

Alice and Bob then undertake a proof of knowledge of values $(\alpha, \beta, x, \delta_1, \delta_2)$ satisfying the following five relations:

$$\begin{gathered} u^\alpha = T_1 \qquad\qquad v^\beta = T_2 \\ e(T_3, g_2)^x \cdot e(h, w)^{-\alpha-\beta} \cdot e(h, g_2)^{-\delta_1-\delta_2} = e(g_1, g_2)/e(T_3, w) \\ T_1^x u^{-\delta_1} = 1 \qquad\qquad T_2^x v^{-\delta_2} = 1 \ . \end{gathered}$$

This proof of knowledge of $(\alpha, \beta, x, \delta_1, \delta_2)$ proceeds as follows. Alice picks blinding values $r_\alpha$, $r_\beta$, $r_x$, $r_{\delta_1}$, and $r_{\delta_2}$ at random from $\mathbb{Z}_p$. She computes five values based on all these:

$$\begin{gathered} R_1 \leftarrow u^{r_\alpha} \qquad\qquad R_2 \leftarrow v^{r_\beta} \\ R_3 \leftarrow e(T_3, g_2)^{r_x} \cdot e(h, w)^{-r_\alpha - r_\beta} \cdot e(h, g_2)^{-r_{\delta_1} - r_{\delta_2}} \\ R_4 \leftarrow T_1^{r_x} \cdot u^{-r_{\delta_1}} \qquad\qquad R_5 \leftarrow T_2^{r_x} \cdot v^{-r_{\delta_2}} \quad . \end{gathered} \tag{2}$$

She then sends $(T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5)$ to the verifier. Bob, the verifier, sends a challenge value $c$ chosen uniformly at random from $\mathbb{Z}_p$. Alice computes and sends back the values

$$s_\alpha \leftarrow r_\alpha + c\alpha \qquad s_\beta \leftarrow r_\beta + c\beta \qquad s_x \leftarrow r_x + cx \qquad s_{\delta_1} \leftarrow r_{\delta_1} + c\delta_1 \qquad s_{\delta_2} \leftarrow r_{\delta_2} + c\delta_2 \quad . \tag{3}$$

Finally, Bob verifies the following five equations:

$$u^{s_\alpha} \stackrel{?}{=} T_1^c \cdot R_1 \tag{4}$$

$$v^{s_\beta} \stackrel{?}{=} T_2^c \cdot R_2 \tag{5}$$

$$e(T_3, g_2)^{s_x} \cdot e(h, w)^{-s_\alpha - s_\beta} \cdot e(h, g_2)^{-s_{\delta_1} - s_{\delta_2}} \stackrel{?}{=} \big(e(g_1, g_2)/e(T_3, w)\big)^c \cdot R_3 \tag{6}$$

$$T_1^{s_x} \cdot u^{-s_{\delta_1}} \stackrel{?}{=} R_4 \tag{7}$$

$$T_2^{s_x} \cdot v^{-s_{\delta_2}} \stackrel{?}{=} R_5 \quad . \tag{8}$$

Bob accepts if all five hold.

**Theorem 4.1.** *Protocol 1 is an honest-verifier zero-knowledge proof of knowledge of an SDH pair under the Decision Linear assumption.*

The proof of the theorem follows from the following lemmas that show that the protocol is (1) complete (the verifier always accepts an interaction with an honest prover), (2) zero-knowledge (can be simulated), and (3) a proof of knowledge (has an extractor).

**Lemma 4.2.** *Protocol 1 is complete.*

*Proof.* If Alice is an honest prover in possession of an SDH pair $(A, x)$ she follows the computations specified for her in the protocol. In this case,

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

so (4) holds. For analogous reasons (5) holds. Further,

$$T_1^{s_x} u^{-s_{\delta_1}} = (u^\alpha)^{r_x + cx} u^{-r_{\delta_1} - cx\alpha} = (u^\alpha)^{r_x} u^{-r_{\delta_1}} = T_1^{r_x} u^{-r_{\delta_1}} = R_4 \quad ,$$

so (7) holds. For analogous reasons (8) holds. Finally,

$$\begin{aligned} & e(T_3, g_2)^{s_x} \cdot e(h, w)^{-s_\alpha - s_\beta} \cdot e(h, g_2)^{-s_{\delta_1} - s_{\delta_2}} \\ &= e(T_3, g_2)^{r_x + cx} \cdot e(h, w)^{-r_\alpha - r_\beta - c\alpha - c\beta} \cdot e(h, g_2)^{-r_{\delta_1} - r_{\delta_2} - cx\alpha - cx\beta} \\ &= e(T_3, g_2^x)^c \cdot e(h^{-\alpha-\beta}, w g_2^x)^c \cdot \big(e(T_3, g_2)^{r_x} \cdot e(h, w)^{-r_\alpha - r_\beta} \cdot e(h, g_2)^{-r_{\delta_1} - r_{\delta_2}}\big) \\ &= e(T_3 h^{-\alpha-\beta}, w g_2^x)^c \cdot e(T_3, w)^{-c} \cdot (R_3) \\ &= \big(e(A, w g_2^x)/e(T_3, w)\big)^c \cdot R_3 \\ &= \big(e(g_1, g_2)/e(T_3, w)\big)^c \cdot R_3 \quad . \end{aligned}$$

so (6) holds. ∎

**Lemma 4.3.** *For an honest verifier, transcripts of Protocol 1 can be simulated, under the Decision Linear assumption.*

*Proof.* We describe a simulator that outputs transcripts of Protocol 1. The simulator begins by picking $A \xleftarrow{R} G_1$ and $\alpha, \beta \xleftarrow{R} \mathbb{Z}_p$. It sets $T_1 \leftarrow u^\alpha$, $T_2 \leftarrow v^\beta$, and $T_3 \leftarrow Ah^{\alpha+\beta}$. Assuming the Decision Linear assumption holds on $G_1$, the tuples $(T_1, T_2, T_3)$ generated by the simulator are drawn from a distribution that is indistinguishable from the distribution output by any particular prover.

The remainder of this simulation does not assume knowledge of $A$, $x$, $\alpha$, or $\beta$, so it can also be used when $T_1$, $T_2$, and $T_3$ are pre-specified. When the pre-specified $(T_1, T_2, T_3)$ are a random Linear encryption of some $A$, the remainder of the transcript is simulated perfectly, as in a standard simulation of a Schnorr proof of knowledge.

The simulator chooses a challenge $c \xleftarrow{R} \mathbb{Z}_p$ and values $s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2} \xleftarrow{R} \mathbb{Z}_p$. It computes $R_1, R_2, R_3, R_4, R_5$ as:

$$R_1 \leftarrow u^{s_\alpha} \cdot T_1^{-c} \qquad R_2 \leftarrow v^{s_\beta} \cdot T_2^{-c} \qquad R_4 \leftarrow T_1^{s_x} \cdot u^{-s_{\delta_1}} \qquad R_5 \leftarrow T_2^{s_x} \cdot v^{-s_{\delta_2}}$$
$$R_3 \leftarrow e(T_3, g_2)^{s_x} \cdot e(h, w)^{-s_\alpha - s_\beta} \cdot e(h, g_2)^{-s_{\delta_1} - s_{\delta_2}} \cdot \big(e(T_3, w)/e(g_1, g_2)\big)^c \ .$$

The resulting values clearly satisfy Equations (4)–(8). Furthemore, the values $R_1, R_2, R_3, R_4, R_5$ are distributed as in a real transcript.

The simulator outputs the transcript $(T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5, c, s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2})$. As discussed above, this transcript is indistinguishable from transcripts of Protocol 1, assuming the Decision Linear assumption holds. □

**Lemma 4.4.** *There exists an extractor for Protocol 1.*

*Proof.* Suppose that an extractor can rewind a prover in the protocol above to the point just before the prover is given a challenge $c$. At the first step of the protocol, the prover sends $T_1, T_2, T_3$ and $R_1, R_2, R_3, R_4, R_5$. Then, to challenge value $c$, the prover responds with $s_\alpha$, $s_\beta$, $s_x$, $s_{\delta_1}$, and $s_{\delta_2}$. To challenge value $c' \neq c$, the prover responds with $s'_\alpha$, $s'_\beta$, $s'_x$, $s'_{\delta_1}$, and $s'_{\delta_2}$. If the prover is convincing, all five verification equations (4)–(8) hold for each set of values.

For brevity, let $\Delta c = c - c'$, $\Delta s_\alpha = s_\alpha - s'_\alpha$, and similarly for $\Delta s_\beta$, $\Delta s_x$, $\Delta s_{\delta_1}$, and $\Delta s_{\delta_2}$.

Now consider (4) above. Dividing the two instances of this equation (one instance using $c$ and the other using $c'$), 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 $u^{\tilde{\alpha}} = T_1$. Similarly, from (5), we obtain $\tilde{\beta} = \Delta s_\beta / \Delta c$ such that $v^{\tilde{\beta}} = T_2$.

Consider (7) above. Dividing the two instances gives $T_1^{\Delta s_x} = u^{\Delta s_{\delta_1}}$. Substituting $T_1 = u^{\tilde{\alpha}}$ gives $u^{\tilde{\alpha}\Delta s_x} = u^{\Delta s_{\delta_1}}$, or $\Delta s_{\delta_1} = \tilde{\alpha}\Delta s_x$. Similarly, from (8) we deduce that $\Delta s_{\delta_2} = \tilde{\beta}\Delta s_x$.

Finally, dividing the two instances of (6), we obtain

$$\begin{aligned}\big(e(g_1, g_2)/e(T_3, w)\big)^{\Delta c} &= e(T_3, g_2)^{\Delta s_x} \cdot e(h, w)^{-\Delta s_\alpha - \Delta s_\beta} \cdot e(h, g_2)^{-\Delta s_{\delta_1} - \Delta s_{\delta_2}} \\ &= e(T_3, g_2)^{\Delta s_x} \cdot e(h, w)^{-\Delta s_\alpha - \Delta s_\beta} \cdot e(h, g_2)^{-\tilde{\alpha}\Delta s_x - \tilde{\beta}\Delta s_x} \ .\end{aligned}$$

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

$$e(g_1, g_2)/e(T_3, w) = e(T_3, g_2)^{\tilde{x}} \cdot e(h, w)^{-\tilde{\alpha} - \tilde{\beta}} \cdot e(h, g_2)^{-\tilde{x}(\tilde{\alpha} + \tilde{\beta})} \ .$$

This can be rearranged as

$$e(g_1, g_2) = e(T_3 h^{-\tilde{\alpha}-\tilde{\beta}}, w g_2^{\tilde{x}}) \enspace ,$$

or, letting $\tilde{A} = T_3 h^{-\tilde{\alpha}-\tilde{\beta}}$,

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

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 Linear encryption $(T_1, T_2, T_3)$. □

## 5 Short Group Signatures from SDH

Armed with Theorem 4.1, we obtain from Protocol 1 a regular signature scheme secure in the random oracle model by applying the Fiat-Shamir heuristic [17, 1]. Signatures obtained from a proof of knowledge via the Fiat-Shamir heuristic are often called signatures of knowledge. The resulting signature scheme is, in fact, also a group signature scheme and we describe it as such. In our construction we use a variant of the Fiat-Shamir heuristic, used also by Ateniese et al. [2], where the challenge $c$ rather than the values $R_1, \ldots, R_5$ is transmitted in the signature; the output of the random oracle acts as a checksum for those values not transmitted.

In describing the group signature, we use the terminology of Bellare et al. [6]. Consider a bilinear group pair $(G_1, G_2)$ with a computable isomorphism $\psi$, as in Section 2. Suppose further that the SDH assumption holds on $(G_1, G_2)$, and the Linear assumption holds on $G_1$. The scheme employs a hash function $H : \{0,1\}^* \to \mathbb{Z}_p$, treated as a random oracle in the proof of security.

***KeyGen***$(n)$**.** This randomized algorithm takes as input a parameter $n$, the number of members of the group, and proceeds as follows. Select a generator $g_2$ in $G_2$ uniformly at random, and set $g_1 \leftarrow \psi(g_2)$. Select $h \stackrel{\mathrm{R}}{\leftarrow} G_1 \backslash \{1_{G_1}\}$ and $\xi_1, \xi_2 \stackrel{\mathrm{R}}{\leftarrow} \mathbb{Z}_p^*$, and set $u, v \in G_1$ such that $u^{\xi_1} = v^{\xi_2} = h$. Select $\gamma \stackrel{\mathrm{R}}{\leftarrow} \mathbb{Z}_p^*$, and set $w = g_2^{\gamma}$.

Using $\gamma$, generate for each user $i$, $1 \le i \le n$, an SDH tuple $(A_i, x_i)$: select $x_i \stackrel{\mathrm{R}}{\leftarrow} \mathbb{Z}_p^*$, and set $A_i \leftarrow g_1^{1/(\gamma + x_i)} \in G_1$.

The group public key is $gpk = (g_1, g_2, h, u, v, w)$. The private key of the group manager (the party able to trace signatures) is $gmsk = (\xi_1, \xi_2)$. Each user's private key is her tuple $\boldsymbol{gsk}[i] = (A_i, x_i)$. No party is allowed to possess $\gamma$; it is only known to the private-key issuer.

***Sign***$(gpk, \boldsymbol{gsk}[i], M)$**.** Given a group public key $gpk = (g_1, g_2, h, u, v, w)$, a user's key $\boldsymbol{gsk}[i] = (A_i, x_i)$, and a message $M \in \{0,1\}^*$, compute the signature as follows:

1. Compute the values $T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5$ as specified in the first round of Protocol 1 (Equations (1) and (2)).
2. Compute a challenge $c$ using the hash function as:
$$c \leftarrow H(M, T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5) \in \mathbb{Z}_p \enspace . \tag{9}$$
3. Using $c$ construct the values $s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2}$ as in the third round of Protocol 1 (Equation (3)).
4. Output the signature $\sigma$, computed as $\sigma \leftarrow (T_1, T_2, T_3, c, s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2})$.

***Verify***$(gpk, M, \sigma)$**.** Given a group public key $gpk = (g_1, g_2, h, u, v, w)$, a message $M$, and a group signature $\sigma$, verify that $\sigma$ is a valid signature as follows:

1. Use Equations (4)–(8) to re-derive $R_1$, $R_2$, $R_3$, $R_4$, and $R_5$ as follows:
$$\tilde{R}_1 \leftarrow u^{s_\alpha} \cdot T_1^{-c} \qquad \tilde{R}_2 \leftarrow v^{s_\beta} \cdot T_2^{-c} \qquad \tilde{R}_4 \leftarrow T_1^{s_x} \cdot u^{-s_{\delta_1}} \qquad \tilde{R}_5 \leftarrow T_2^{s_x} \cdot v^{-s_{\delta_2}} \tag{10}$$
$$\tilde{R}_3 \leftarrow e(T_3, g_2)^{s_x} \cdot e(h, w)^{-s_\alpha - s_\beta} \cdot e(h, g_2)^{-s_{\delta_1} - s_{\delta_2}} \cdot \big(e(T_3, w)/e(g_1, g_2)\big)^c \ .$$

2. Check that these, along with the other first-round values included in $\sigma$, give the challenge $c$, i.e., that
$$c \stackrel{?}{=} H(M, T_1, T_2, T_3, \tilde{R}_1, \tilde{R}_2, \tilde{R}_3, \tilde{R}_4, \tilde{R}_5) \ . \tag{11}$$
Accepts if this check succeeds and reject otherwise.

***Open***$(gpk, gmsk, M, \sigma)$**.** This algorithm is used for tracing a signature to a signer. It takes as input a group public key $gpk = (g_1, g_2, h, u, v, w)$ and the corresponding group manager's private key $gmsk = (\xi_1, \xi_2)$, together with a message $M$ and a signature $\sigma = (T_1, T_2, T_3, c, s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2})$ to trace, and proceeds as follows. First, verify that $\sigma$ is a valid signature on $M$. Second, consider the first three elements $(T_1, T_2, T_3)$ as a Linear encryption, and recover the user's $A$ as $A \leftarrow T_3/(T_1^{\xi_1} \cdot T_2^{\xi_2})$, following the decryption algorithm given at the end of Section 3.2.1. If the group manager is given the elements $\{A_i\}$ of the users' private keys, he can look up the user index corresponding to the identity $A$ recovered from the signature.

**Signature Length.** A group signature in the system above comprises three elements of $G_1$ and six elements of $\mathbb{Z}_p$. Using any of the families of curves described in [9], 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 1533 bits or 192 bytes. With these parameters, security is approximately the same as a standard 1024-bit RSA signature, which is 128 bytes.

**Performance.** The pairings $e(h, w)$, $e(h, g_2)$, and $e(g_1, g_2)$ can be precomputed and cached by both signers and verifiers. The signer can cache $e(A, g_2)$, and, when signing, compute $e(T_3, g_2)$ without evaluating a pairing. Accordingly, creating a group signature requires eight exponentiations (or multi-exponentiations) and no pairing computations. The verifier can derive $\tilde{R}_3$ efficiently by collapsing the $e(T_3, g_2)^{s_x}$ and $e(T_3, w)^c$ pairings into a single $e(T_3, w^c g_2^{s_x})$ term. Thus verifying a group signature requires six multi-exponentiations and one pairing computation. With parameters selected as above, the exponents are in every case 170-bit numbers. For the signer, all bases for exponentiation are fixed, which allows further speedup by precomputation.

## 5.1 Group Signature Security

We now turn to proving security of the system. Bellare et al. [6] give three properties that a group signature scheme must satisfy:

- correctness, which ensures that honestly-generated signatures verify and trace correctly;
- full-anonymity, which ensures that signatures do not reveal their signer's identity; and

- full-traceability, which ensures that all signatures, even those created by the collusion of multiple users and the group manager, trace to a member of the forging coalition.

For the details of the definitions, see Bellare et al. [6]. We prove the security of our scheme using a variation of these properties. In our proofs, we relax the full-anonymity requirement. As presented [6, Sect. 2], the full-anonymity experiment allows the adversary to query the opening (tracing) oracle before and after receiving the challenge $\sigma$. In this respect, the experiment mirrors the indistinguishability experiment against an adaptive CCA2 adversary. We therefore rename this experiment CCA2-full-anonymity. We define a corresponding experiment, CPA-full-anonymity, in which the adversary cannot query the opening oracle. We prove privacy in this slightly weaker model.

Access to the tracing functionality will likely be carefully controlled when group signatures are deployed, so CPA-full-anonymity is a reasonable model to consider. In any case, anonymity and unlinkability, the two traditional group signature security requirements implied by full anonymity [6, Sect. 3], also follow from CPA-full-anonymity. Thus a fully-traceable and CPA-fully-anonymous group signature scheme is still secure in the traditional sense.

In the statements of the theorem, we use big-$O$ notation to elide the specifics of additive terms in time bounds, noting that, for given groups $G_1$ and $G_2$, operations such as sampling, exponentiation, and bilinear map evaluation are all constant-time.

**Theorem 5.1.** *The SDH group signature scheme is correct.*

*Proof.* For any group public key $gpk = (g_1, g_2, h, u, v, w)$, and for any user with key $\boldsymbol{gsk}[i] = (A_i, x_i)$, the key generation algorithm guarantees that $A_i^{\gamma+x_i} = g_1$, so $(A_i, x_i)$ is an SDH tuple for $w = g_2^{\gamma}$. A correct group signature $\sigma$ is a proof of knowledge, which is itself a transcript of the SDH protocol given in Section 4. Verifying the signature entails verifying that the transcript is correct; thus Lemma 4.2 shows that $\sigma$ will always be accepted by the verifier.

Moreover, an honest signer outputs, as the first three components of any signature $\sigma$, values $(T_1, T_2, T_3) = (u^{\alpha}, v^{\beta}, A_i \cdot h^{\alpha+\beta})$ for some $\alpha, \beta \in \mathbb{Z}_p$. These values form a Linear encryption of $A_i$ under public key $(u, v, h)$, which the group manager, possessing the corresponding private key $(\xi_1, \xi_2)$, can always recover. Therefore any valid signature will always be opened correctly. $\square$

**Theorem 5.2.** *If Linear encryption is $(t', \epsilon')$-semantically secure on $G_1$ then the SDH group signature scheme is $(t, q_H, \epsilon)$-CPA-fully-anonymous, where $\epsilon = \epsilon'$ and $t = t' - q_H O(1)$. Here $q_H$ is the number of hash function queries made by the adversary and $n$ is the number of members of the group.*

*Proof.* Suppose $\mathcal{A}$ is an algorithm that $(t, q_H, \epsilon)$-breaks the anonymity of the group signature scheme. We show how to construct a $t + q_H O(1)$-time algorithm $\mathcal{B}$ that breaks the semantic security of Linear encryption from Section 3.2.1 with advantage at least $\epsilon$.

Algorithm $\mathcal{B}$ is given a Linear encryption public key $(u, v, h)$. It generates the remaining components of the group signature public key by following the group signature's key generation algorithm. It then provides to $\mathcal{A}$ the group public key $(g_1, g_2, h, u, v, w)$, and the users' private keys $(A_i, x_i)$.

At any time, $\mathcal{A}$ can query the random oracle $H$. Algorithm $\mathcal{B}$ responds with elements selected uniformly at random from $\mathbb{Z}_p$, making sure to respond identically to repeated queries.

Algorithm $\mathcal{A}$ requests its full-anonymity challenge by providing two indices, $i_0$ and $i_1$, and a message $M$. Algorithm $\mathcal{B}$, in turn, requests its indistinguishability challenge by providing the two user private keys $A_{i_0}$ and $A_{i_1}$ as the messages whose Linear encryption it must distinguish. It is given a Linear encryption $(T_1, T_2, T_3)$ of $A_{i_b}$, where bit $b$ is chosen by the Linear encryption challenger.

Algorithm $\mathcal{B}$ generates from this Linear encryption a protocol transcript $(T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5, c, s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2})$ by means of the simulator of Lemma 4.3. This simulator can generate a trace given $(T_1, T_2, T_3)$, even though $\mathcal{B}$ does not know $\alpha$, $\beta$, or $x$. Since $(T_1, T_2, T_3)$ is a random Linear encryption of $A_{i_b}$, the remainder of the transcript is distributed exactly as in a real protocol with a prover whose secret $A$ is $A_{i_b}$.

Algorithm $\mathcal{B}$ then patches $H$ at $(M, T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5)$ to equal $c$. It encounters a collision only with negligible probability. In case of a collision, $\mathcal{B}$ declares failure and exits. Otherwise, it returns the valid group signature $\sigma \leftarrow (T_1, T_2, T_3, c, s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2})$ to $\mathcal{A}$.

Finally, $\mathcal{A}$ outputs a bit $b'$. Algorithm $\mathcal{B}$ returns $b'$ as the answer to its own challenge. Since the encryption of $A_{i_b}$ is turned by $\mathcal{B}$ into a group signature by user $i_b$, $\mathcal{B}$ answers its challenge correctly whenever $\mathcal{A}$ does.

The keys given to $\mathcal{A}$, and the answers to $\mathcal{A}$'s queries, are all valid and properly distributed. Therefore $\mathcal{A}$ succeeds in breaking the anonymity of the group signature $\sigma$ with advantage $\epsilon$, and $\mathcal{B}$ succeeds in distinguishing the Linear encryption $(T_1, T_2, T_3)$ with the same advantage.

Algorithm $\mathcal{B}$'s running time exceeds $\mathcal{A}$'s by the amount it takes to answer $\mathcal{A}$'s queries. Each hash query can be answered in constant time, and there are at most $q_H$ of them. Algorithm $\mathcal{B}$ can also create the challenge group signature $\sigma$ in constant time. If $\mathcal{A}$ runs in time $t$, $\mathcal{B}$ runs in time $t + q_H O(1)$. $\square$

The following theorem proves full traceability of our system. The proof is based on the Forking Lemma [26].

**Theorem 5.3.** *If SDH is $(q, t', \epsilon')$-hard on $(G_1, G_2)$, then the SDH group signature scheme is $(t, q_H, q_S, n, \epsilon)$-fully-traceable, where $n = q - 1$, $\epsilon = 4n\sqrt{2\epsilon' q_H} + n/p$, and $t = \Theta(1) \cdot t'$. Here $q_H$ is the number of hash function queries made by the adversary, $q_S$ is the number of signing queries made by the adversary, and $n$ is the number of members of the group.*

*Proof.* Our proof proceeds in three parts. First, we describe a framework for interacting with an algorithm that wins a full-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 [26] to the framework instances, obtaining SDH solutions.

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

**Setup.** We are given groups $(G_1, G_2)$ as above. We are given generators $g_1$ and $g_2$ such that $g_1 = \psi(g_2)$. 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 pick a generator $h \stackrel{\mathrm{R}}{\leftarrow} G_1 \setminus \{1_{G_1}\}$ and values $\xi_1, \xi_2 \stackrel{\mathrm{R}}{\leftarrow} \mathbb{Z}_p^*$, and compute $u, v \in G_1$ such that $u^{\xi_1} = v^{\xi_2} = h$. We then run $\mathcal{A}$, giving it the group public key $(g_1, g_2, h, u, v, w)$ and the group manager's private key $(\xi_1, \xi_2)$. We answer its oracle queries as follows.

**Hash Queries.** When $\mathcal{A}$ asks for the hash of $(M, T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5)$, we respond with a random element of $G_1$, memoizing the answer in case the same query is made again.

**Signature Queries.** Algorithm $\mathcal{A}$ asks for a signature on message $M$ by a key at index $i$. If $x_i \neq \star$, we follow the group signing procedure with key $(A_i, x_i)$ to obtain a signature $\sigma$ on $M$, and return $\sigma$ to $\mathcal{A}$. If $x_i = \star$, we pick $\alpha, \beta \xleftarrow{\mathrm{R}} \mathbb{Z}_p$, set $T_1 \leftarrow u^\alpha$, $T_2 = v^\beta$, and $T_3 \leftarrow A g_1^{\alpha+\beta}$ and run the Protocol 1 simulator with values $T_1, T_2, T_3$. The simulator returns a transcript $(T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5, c, s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2})$, from which we derive a group signature $\sigma = (T_1, T_2, T_3, c, s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2})$. In addition, we must patch the hash oracle at $(M, T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5)$ 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 $\sigma$ 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 group signature $\sigma = (T_1, T_2, T_3, c, s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2})$ on a message $M$. We use the group manager's key $(\xi_1, \xi_2)$ to trace $\sigma$, obtaining some $A^*$. If $A^* \neq A_i$ for all $i$, we output $\sigma$. Otherwise, $A^* = A_{i^*}$ for some $i^*$. If $s_{i^*} = \star$, we output $\sigma$. 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 $\sigma$ on a message $M$ that traces to some identity $A^* \notin \{A_1, \ldots, A_n\}$. Type II forgers output a forgery that traces to 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 [8], obtaining generators $g_1 \in G_1$, $g_2 \in G_2$, $w = g_2^\gamma$, and $q-1$ SDH pairs $(A_i, x_i)$ such that $e(A_i, w g_2^{x_i}) = e(g_1, g_2)$ 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 $\epsilon$.

**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 $A_{i^*} \xleftarrow{\mathrm{R}} G_1$, 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 nine 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 $\sigma$, 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 $\epsilon/n$.

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 [26].

Let $\mathcal{A}$ be a forger (of either type) for which the framework succeeds with probability $\epsilon'$. From here on, we abbreviate signatures as $(M, \sigma_0, c, \sigma_1)$, where $\sigma_0 = (T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5)$, the values given, along with $M$, to the random oracle $H$, and from which $c$ is derived, and where $\sigma_1 = (s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2})$. Those values normally omitted from the signature can be recovered as in Equation (10).

A run of the framework on $\mathcal{A}$ is completely described by the randomness string $\omega$ used by the framework and $\mathcal{A}$, and by the vector $f$ of responses made by the hash oracle. Let $S$ be the set of pairs $(\omega, h)$ 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 on $(M, \sigma_0)$. In this case, let $\mathrm{Ind}(\omega, f)$ be the index of $f$ at which $\mathcal{A}$ queried $(M, \sigma_0)$. We define $\nu = \Pr[S] = \epsilon' - 1/p$, where the $1/p$ term accounts for the possibility that $\mathcal{A}$ guessed the hash of $(M, \sigma_0)$ without the hash oracle's help. For each $j$, $1 \le j \le q_H$, let $S_j$ be the set of pairs $(\omega, h)$ as above, and such that $\mathrm{Ind}(\omega, f) = j$. 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 [26, Lemma 1] with rows $X = (\omega, f|_1^{j-1})$ and columns $Y = (f|_j^{q_H})$. Clearly $\Pr_{(x,y)}[(x, y) \in S_j] \ge \nu/(2q_H)$. Let the heavy rows $\Omega_j$ 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 $(\omega, f)$ such $\Pr_{f'}\left[(\omega, f') \in S_j \;\middle|\; f'|_1^{j-1} = f|_1^{j-1}\right] \ge \nu/(4q_H)$.

If we now rewind the framework and $\mathcal{A}$ to the $j$th query, and proceed with an oracle vector $f'$ that differs from $f$ from the $j$th 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 $(M, \sigma_0)$ still queried at $\mathcal{A}$'s $j$th hash query.

By using the extractor of Lemma 4.4, we obtain from $(\sigma_0, c, \sigma_1)$ and $(\sigma_0, c', \sigma_1')$ an SDH tuple $(A, x)$. The extracted $A$ is the same as the $A$ in the Linear encryption $(T_1, T_2, T_3)$ in $\sigma_0$. The framework declares success only when the $A$ encrypted in $(T_1, T_2, T_3)$ 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 [8], 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 $\mathcal{A}$, 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 $\mathcal{A}$, 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. □

## 6 Revocation

We now discuss how to revoke users in the SDH group signature scheme of Section 5. A number of revocation mechanisms for group signatures have been proposed [4, 12]. All these mechanisms can be applied to our system. Here we describe a revocation mechanism along the lines of [12].

Recall that the group's public key in our system is $(g_1, g_2, h, u, v, w)$ where $w = g_2^\gamma \in G_2$ for random $\gamma \in \mathbb{Z}_p^*$ and random $h, u, v \in G_1$. User $i$'s private key is a pair $(A_i, x_i)$ where $A_i = g_1^{1/(\gamma+x_i)} \in G_1$.

Now, suppose we wish to revoke users $1, \dots, r$ without affecting the signing capability of other users. To do so, the Revocation Authority (RA) publishes a Revocation List (RL) containing the private keys of all revoked users. More precisely, $\mathrm{RL} = \{(A_1^*, x_1), \dots, (A_r^*, x_r)\}$, where $A_i^* = g_2^{1/(\gamma+x_i)} \in G_2$. Note that $A_i = \psi(A_i^*)$. Here the SDH secret $\gamma$ is needed to compute the $A_i^*$'s. In case $G_1$ equals $G_2$ then $A_i = A_i^*$ and consequently the Revocation List can be derived directly from the private keys of revoked users without having to use $\gamma$.

The list RL is given to all signers and verifiers in the system. It is used to update the group public key used to verify signatures. Let $y = \prod_{i=1}^r (\gamma + x_i) \in \mathbb{Z}_p^*$. The new public key is $(\bar{g}_1, \bar{g}_2, h, u, v, \bar{w})$ where $\bar{g}_1 = g_1^{1/y}$, $\bar{g}_2 = g_2^{1/y}$, and $\bar{w} = (\bar{g}_2)^\gamma$. We show that, given RL, anyone can compute this new public key, and any unrevoked user can update her private key locally so that it is well formed with respect to this new public key. Revoked users are unable to do so.

We show how to revoke one private key at a time. By repeating the process $r$ times (as the revocation list grows over time) we can revoke all private keys on the Revocation List. We first show how given the public key $(g_1, g_2, h, u, v, w)$ and one revoked private key $(A_1^*, x_1) \in \mathrm{RL}$ anyone can construct the new public key $(\hat{g}_1, \hat{g}_2, h, u, v, \hat{w})$ where $\hat{g}_1 = g_1^{1/(\gamma+x_1)}$, $\hat{g}_2 = g_2^{1/(\gamma+x_1)}$, and $\hat{w} = (\hat{g}_2)^\gamma$. This new public key is constructed simply as:

$$\hat{g}_1 \leftarrow \psi(A_1^*) \qquad \hat{g}_2 \leftarrow A_1^* \qquad \text{and} \qquad \hat{w} \leftarrow g_2 \cdot (A_1^*)^{-x_1} \ ;$$

then $\hat{g}_1 = \psi(A_1)^* = g_1^{1/(\gamma+x_1)}$ and $\hat{w} = g_2 \cdot (A_1^*)^{-x_1} = g_2^{1 - \frac{x_1}{\gamma+x_1}} = (A_1^*)^\gamma = (\hat{g}_2)^\gamma$, as required.

Next, we show how unrevoked users update their own private keys. Consider an unrevoked user whose private key is $(A, x)$. Given a revoked private key, $(A_1^*, x_1)$ the user computes $\hat{A} \leftarrow \psi(A_1^*)^{1/(x-x_1)}/A^{1/(x-x_1)}$ and sets his new private key to be $(\hat{A}, x)$. Then, indeed,

$$(\hat{A})^{\gamma+x} = \psi(A_1^*)^{\frac{\gamma+x}{x-x_1}} / A^{\frac{\gamma+x}{x-x_1}} = \psi(A_1^*)^{\frac{(\gamma+x_1)+(x-x_1)}{x-x_1}} / g_1^{\frac{1}{x-x_1}} = \psi(A_1^*) = \hat{g}_1 \ ,$$

as required. Hence, $(\hat{A}, x)$ is a valid private key with respect to $(\hat{g}_1, \hat{g}_2, h, u, v, \hat{w})$.

By repeating this process $r$ times (once for each revoked key in RL) anyone can compute the updated public key $(\bar{g}_1, \bar{g}_2, h, u, v, \bar{w})$ defined above. Similarly, an unrevoked user with private key $(A, x)$ can compute his updated private key $(\bar{A}, x)$ where $\bar{A} = (\bar{g}_1)^{1/(\gamma+x)}$. We note that it is possible to process the entire RL at once (as opposed to one element at a time) and compute $(\bar{g}_1, \bar{g}_2, h, u, v, \bar{w})$ directly; however this is less efficient when keys are added to RL incrementally.

A revoked user cannot construct a private key for the new public key $(\bar{g}_1, \bar{g}_2, h, u, v, \bar{w})$. In fact, the proof of Theorem 5.3 shows that, if a revoked user can generate signatures for the new public key $(\bar{g}_1, \bar{g}_2, h, u, v, \bar{w})$, then that user can be used to break the SDH assumption. Very briefly, the reason is that given an SDH challenge one can easily generate a public key tuple $(\bar{g}_1, \bar{g}_2, h, u, v, \bar{w})$ along with the private key for a revoked user $(g_1^{1/(x+\gamma)}, x)$. Then an algorithm that can forge signatures given these two tuples can be used to solve the SDH challenge.

In the revocation mechanism above a user is revoked by the publication of a value that exposes that user's private key. Consequently, it is crucial that updates to the revocation list be sent simultanously to all verifiers. Otherwise, someone who obtains a new entry on the revocation list can fool a verifier who has not yet updated its copy of the revocation list.

Brickell [11] proposes an alternate mechanism where revocation messages are only sent to signature verifiers, so that there is no need for unrevoked signers to update their keys. Similar mechanisms were also considered by Ateniese et al. [4] and Kiayias et al. [21]. We refer to this as Verifier-Local Revocation (VLR) group signatures. Boneh and Shacham [10] show how to modify our group signature scheme to support this VLR revocation mechanism. Using this revocation mechanism, only a fragment of the user's private key is placed on the revocation list and hence the limitation discussed in the previous paragraph is not an issue.

# 7 Exculpability

In Bellare et al. [6], exculpability (introduced by Ateniese and Tsudik [3]) is informally defined as follows: No member of the group and not even the group manager — the entity that is given the tracing key — can produce signatures on behalf of other users. Thus, no user can be framed for producing a signature he did not produce. They argue that a group signature secure in the sense of full-traceability also has the exculpability property. Thus, in the terminology of Bellare et al. [6], our group signature has the exculpability property.

A stronger notion of exculpability is considered in Ateniese et al. [2], where one requires that even the entity that *issues* user keys cannot forge signatures on behalf of users. Formalizations of strong exculpability have recently been proposed by Kiayias and Yung [22] and by Bellare, Shi, and Zhang [7].

To achieve this stronger property the system of Ateniese et al. [2] uses a protocol (called JOIN) to issue a key to a new user. At the end of the protocol, the key issuer does not know the full private key given to the user and therefore cannot forge signatures under the user's key.

Our group signature scheme can be extended to provide strong exculpability using a similar mechanism. Instead of simply giving user $i$ the private key $(g_1^{1/(\gamma+x_i)}, x_i)$, the user and key issuer engage in a JOIN protocol where at the end of the protocol user $i$ has a triple $(A_i, x_i, y_i)$ such that $A_i^{\gamma+x_i} h_1^{y_i} = g_1$ for some public parameter $h_1$. The value $y_i$ is chosen by the user and is kept secret from the key issuer. The ZKPK of Section 4 can be modified to prove knowledge of such a triple. The resulting system is a short group signature with strong exculpability.

# 8 The Linear Problem in Generic Bilinear Groups

To provide more confidence in the Decision Linear assumption introduced in Section 3.2 we prove a lower bound on the computational complexity of the Decision Linear problem for generic groups in the sense of Shoup [29]. In this model, elements of $G_1$, $G_2$, and $G_T$ appear to be encoded as unique random strings, so that no property other than equality can be directly tested by the adversary.

Five oracles are assumed to perform operations between group elements, such as computing the group action in each of the three groups $G_1$, $G_2$, $G_T$, as well as the isomorphism $\psi : G_2 \to G_1$, and the bilinear pairing $e : G_1 \times G_2 \to G_T$ (where possibly $G_1 = G_2$). The opaque encoding of the elements of $G_1$ is modeled as an injective function $\xi_1 : \mathbb{Z}_p \to \Xi_1$, where $\Xi_1 \subset \{0,1\}^*$, which maps all $a \in \mathbb{Z}_p$ to the string representation $\xi_1(g^a)$ of $g^a \in G_1$. Analogous maps $\xi_2 : \mathbb{Z}_p \to \Xi_2$ for $G_2$ and $\xi_T : \mathbb{Z}_p \to \Xi_T$ for $G_T$ are also defined. The attacker $\mathcal{A}$ communicates with the oracles using the $\xi$-representations of the group elements only.

Let $x, y, z, a, b, c \xleftarrow{\text{R}} \mathbb{Z}_p^*$, $T_0 \leftarrow g^{z(a+b)}$, $T_1 \leftarrow g^c$, and $d \xleftarrow{\text{R}} \{0,1\}$. We show that no generic algorithm $\mathcal{A}$ that is given the encodings of $g^x, g^y, g^z, g^{xa}, g^{yb}, T_d, T_{1-d}$ and makes up to $q$ oracle queries can guess the value of $d$ with probability greater than $\frac{1}{2} + O(q^2/p)$. Note that here $g^x$, $g^y$, and $g^z$ play the role of the generators $u$, $v$, and $h$ in the Decision Linear problem definition.

**Theorem 8.1.** *Let $\mathcal{A}$ be an algorithm that solves the Decision Linear problem in the generic group model. Assume that $\xi_1, \xi_2, \xi_T$ are random encoding functions for $G_1, G_2, G_T$. If $\mathcal{A}$ makes a total of at most $q$ queries to the oracles computing the group action in $G_1, G_2, G_T$, the isomorphism $\psi$, and the bilinear pairing $e$, then*

$$\left| \Pr\left[ \begin{array}{r} \mathcal{A}\left( \begin{array}{l} p,\ \xi_1(1),\ \xi_1(x), \xi_1(y), \xi_1(z), \\ \xi_1(xa), \xi_1(yb),\ \xi_1(t_0), \xi_1(t_1),\ \xi_2(1) \end{array} \right) = d\ : \\ x, y, z, a, b, c \xleftarrow{\text{R}} \mathbb{Z}_p^*,\ d \xleftarrow{\text{R}} \{0,1\}, \\ t_d \leftarrow z(a+b),\ t_{1-d} \leftarrow c \end{array} \right] - \frac{1}{2} \right| \le \frac{8(q+9)^2}{p} \ .$$

*Proof.* Consider an algorithm $\mathcal{B}$ that plays the following game with $\mathcal{A}$.

$\mathcal{B}$ maintains three lists of pairs, $L_1 = \{(F_{1,i}, \xi_{1,i}) : i = 0, \ldots, \tau_1 - 1\}$, $L_2 = \{(F_{2,i}, \xi_{2,i}) : i = 0, \ldots, \tau_2 - 1\}$, $L_T = \{(F_{T,i}, \xi_{T,i}) : i = 0, \ldots, \tau_T - 1\}$, under the invariant that, at step $\tau$ in the game, $\tau_1 + \tau_2 + \tau_T = \tau + 9$. Here, the $F_{\star,\star} \in \mathbb{Z}_p[X, Y, Z, A, B, T_0, T_1]$ are polynomials in the indeterminates $X, Y, Z, A, B, T_0, T_1$ with coefficients in $\mathbb{Z}_p$. The $\xi_{\star,\star} \in \{0,1\}^*$ are arbitrary distinct strings.

The lists are initialized at step $\tau = 0$ by initializing $\tau_1 \leftarrow 8$, $\tau_2 \leftarrow 1$, $\tau_T \leftarrow 0$, and setting $F_{1,0} = 1$, $F_{1,1} = X$, $F_{1,2} = Y$, $F_{1,3} = Z$, $F_{1,4} = XA$, $F_{1,5} = YB$, $F_{1,6} = T_0$, $F_{1,7} = T_1$, and $F_{2,0} = 1$. The corresponding strings are set to arbitrary distinct strings in $\{0,1\}^*$.

We may assume that $\mathcal{A}$ only makes oracle queries on strings previously obtained form $\mathcal{B}$, since $\mathcal{B}$ can make them arbitrarily hard to guess. We note that $\mathcal{B}$ can determine the index $i$ of any given string $\xi_{1,i}$ in $L_1$ (resp. $\xi_{2,i}$ in $L_2$, or $\xi_{T,i}$ in $L_T$), where ties between multiple matches are broken arbitrarily.

$\mathcal{B}$ starts the game by providing $\mathcal{A}$ with the encodings $\xi_{1,0}$, $\xi_{1,1}$, $\xi_{1,2}$, $\xi_{1,3}$, $\xi_{1,4}$, $\xi_{1,5}$, $\xi_{1,6}$, and $\xi_{2,0}$. The simulator $\mathcal{B}$ responds to algorithm $\mathcal{A}$'s queries as follows.

**Group action.** Given a multiply/divide selection bit and two operands $\xi_{1,i}$ and $\xi_{1,j}$ with $0 \le i, j < \tau_1$, compute $F_{1,\tau_1} \leftarrow F_{1,i} \pm F_{1,j}$ depending on whether a multiplication or a division is requested. If $F_{1,\tau_1} = F_{1,l}$ for some $l < \tau_1$, set $\xi_{1,\tau_1} \leftarrow \xi_{1,l}$; otherwise, set $\xi_{1,\tau_1}$ to a string in $\{0,1\}^*$ distinct from $\xi_{1,0}, \ldots, \xi_{1,\tau_1-1}$. Add $(F_{1,\tau_1}, \xi_{1,\tau_1})$ to the list $L_1$ and give $\xi_{1,\tau_1}$ to $\mathcal{A}$, then increment $\tau_1$ by one. Group action queries in $G_2$ and $G_T$ are treated similarly.

**Isomorphism.** Given a string $\xi_{2,i}$ with $0 \le i < \tau_2$, set $F_{1,\tau_1} \leftarrow F_{2,i}$. If $F_{1,\tau_1} = F_{1,l}$ for some $l < \tau_1$, set $\xi_{1,\tau_1} \leftarrow \xi_{1,l}$; otherwise, set $\xi_{1,\tau_1}$ to a string in $\{0,1\}^* \setminus \{\xi_{1,0}, \ldots, \xi_{1,\tau_1-1}\}$. Add $(F_{1,\tau_1}, \xi_{1,\tau_1})$ to the list $L_1$, and give $\xi_{1,\tau_1}$ to $\mathcal{A}$, then increment $\tau_1$ by one.

**Pairing.** Given two operands $\xi_{1,i}$ and $\xi_{2,j}$ with $0 \le i < \tau_1$ and $0 \le j < \tau_2$, compute the product $F_{T,\tau_T} \leftarrow F_{1,i} F_{2,j}$. If $F_{T,\tau_T} = F_{T,l}$ for some $l < \tau_T$, set $\xi_{T,\tau_T} \leftarrow \xi_{T,l}$; otherwise, set $\xi_{T,\tau_T}$ to a string in $\{0,1\}^* \setminus \{\xi_{T,0}, \ldots, \xi_{T,\tau_T -1}\}$. Add $(F_{T,\tau_T}, \xi_{T,\tau_T})$ to the list $L_T$, and give $\xi_{T,\tau_T}$ to $\mathcal{A}$, then increment $\tau_T$ by one.

Observe that at any time in the game, the total degree of any polynomial in each of the three lists is bounded as follows: $\deg(F_{1,i}) \le 2$, $\deg(F_{2,i}) = 0$ (or $\deg(F_{2,i}) \le 2$ if $G_1 = G_2$), and $\deg(F_{T,i}) \le 2$ (or $\deg(F_{T,i}) \le 4$ if $G_1 = G_2$).

After at most $q$ queries, $\mathcal{A}$ terminates and returns a guess $\hat{d} \in \{0,1\}$. At this point $\mathcal{B}$ chooses random $x, y, z, a, b, c \stackrel{\mathrm{R}}{\leftarrow} \mathbb{Z}_p$. Consider $t_d \leftarrow z(a+b)$ and $t_{1-d} \leftarrow c$ for both choices of $d \in \{0,1\}$. The simulation provided by $\mathcal{B}$ is perfect and reveals nothing to $\mathcal{A}$ about $d$ unless the chosen random values for the indeterminates give rise to a non-trivial equality relation between the simulated group elements that was not revealed to $\mathcal{A}$, i.e., when we assign $X \leftarrow x, Y \leftarrow y, Z \leftarrow z, A \leftarrow a, B \leftarrow b$, and either $T_0 \leftarrow z(a+b), T_1 \leftarrow c$ or the converse $T_0 \leftarrow c, T_1 \leftarrow z(a+b)$. This happens only if for some $i, j$ one of the following holds:

1. $F_{1,i}(x, y, z, a, b, z(a+b), c) - F_{1,j}(x, y, z, a, b, z(a+b), c) = 0$, yet $F_{1,i} \neq F_{1,j}$,
2. $F_{2,i}(x, y, z, a, b, z(a+b), c) - F_{2,j}(x, y, z, a, b, z(a+b), c) = 0$, yet $F_{2,i} \neq F_{2,j}$,
3. $F_{T,i}(x, y, z, a, b, z(a+b), c) - F_{T,j}(x, y, z, a, b, z(a+b), c) = 0$, yet $F_{T,i} \neq F_{T,j}$,
4. any relation similar to the above in which $z(a+b)$ and $c$ have been exchanged.

We first need to argue that the adversary is unable to engineer any of the above equalities, so that they can only occur due to an unfortunate random choice of $x, y, z, a, b, c$. First, observe that the adversary can only manipulate the polynomials on the three lists through additions and substractions (disguised as multiplications and divisions in the groups $G_1$, $G_2$, and $G_T$) as well as mutiplications between polynomials which are not the result of a previous multiplication (disguised as pairings between elements of $G_1$ and $G_2$). Now, notice that in the initial population of the lists, the only occurrence of the variable $A$ is within the monomial $XA$, the only occurrence of the variable $B$ is within the monomial $YB$, and the only occurrence of the variable $Z$ is by itself. Given the available operations, it is easy to see that, in the three group representations:

1. the adversary is unable to generate any polynomial that contains at least one of the monomials $mZA$ and $mZB$ for any integer $m \neq 0$, which is a prerequisite to synthesize a multiple of $Z(A+B)$ in $G_1$ or $G_2$ (recall that the maximum degree in those groups is 2);
2. the adversary is unable to simultaneously generate the terms $FZA$ and $FZB$ for any non-zero monomial $F$ of degree at most 2, which is a prerequisite to synthesize a multiple of the polynomial $Z(A+B)$ in $G_T$ (the maximum degree in this group being 4).

Since in the above polynomial differences all arguments to the polynomials are independent except for $z(a+b)$, it is easy to see that the adversary will not be able to cause any of them to cancel identically and non-trivially without knowledge of a multiple of $Z(A+B)$. The adversary is thus reduced to find a numeric cancellation for random assignments of the variables.

We now determine the probability of a random occurrence of a non-trivial numeric cancellation. Since $F_{1,i} - F_{1,j}$ for fixed $i$ and $j$ is a polynomial of degree at most 2, it vanishes for random assignment of the indeterminates in $\mathbb{Z}_p$ with probability at most $2/p$. Similarly, for fixed $i$ and $j$, the second case occurs with probability 0 (or $\leq 2/p$ when $G_1 = G_2$), and the third with probability $\leq 2/p$ (or $\leq 4/p$ when $G_1 = G_2$). The same probabilities are found in the analogous cases where $z(a+b)$ and $c$ have been exchanged.

Now, absent any of the above events, the distribution of the bit $d$ in $\mathcal{A}$'s view is independent, and $\mathcal{A}$'s probability of making a correct guess is exactly $\frac{1}{2}$. Thus, by summing over all valid pairs $i, j$ in each case, we find that $\mathcal{A}$ makes a correct guess with advantage $\epsilon \leq 2 \cdot (\binom{\tau_1}{2}\frac{2}{p} + \binom{\tau_2}{2}\frac{2}{p} + \binom{\tau_T}{2}\frac{4}{p})$. Since $\tau_1 + \tau_2 + \tau_T \leq q + 9$, we have $\epsilon \leq 8(q+9)^2/p$, as required. $\square$

### 8.1 Implications of DDH Hardness on $G_1$

When $G_1$ and $G_2$ are distinct groups, the proof above implies that, in the generic model, the standard Decision Diffie-Hellman (DDH) problem is hard in the group $G_1$ (even though DDH in $G_2$ is easy). For DDH to be hard in a specific group $G_1$, the map $\psi : G_2 \to G_1$ must be computationally one-way. This requirement may hold when the bilinear groups are instantiated using the Weil or Tate pairing over MNT curves [25] [9, Sect. 4.3] where $G_1$ and $G_2$ are distinct ($G_1$ is defined over the ground field of the curve where as $G_2$ is defined over a low-degree extension). Supersingular curves do not have this property since DDH is known to be easy on all cyclic subgroups [18].

Now suppose that for MNT curves the DDH assumption holds in $G_1$. In this case we can construct even shorter group signatures and group signatures that satisfy CCA2-full-anonymity.

**Shorter Group Signatures.** If DDH holds in $G_1$ then ElGamal encryption is secure in $G_1$ and can be used as the encryption in an SDH group signature: $T_1 = u^\alpha$, $T_2 = A \cdot v^\alpha$. (The preimages $\psi^{-1}(u), \psi^{-1}(v) \in G_2$ of $u, v \in G_1$ must not be revealed.) The group signature then comprises only two elements of $G_1$ and four of $\mathbb{Z}_p$. With parameters chosen as in Section 5, we obtain a 1022-bit group signature whose security is comparable to that of standard 1024-bit RSA signatures. This is about 30% shorter than the signatures in Section 5.

**Full-CCA-Anonymity.** Likewise, if DDH holds in $G_1$ then Cramer-Shoup encryption [15] is secure in $G_1$, and can be used in the group signature scheme. Since Cramer-Shoup encryption is semantically secure against an adaptive CCA2 attack, the resulting group signature scheme is CCA2-fully-anonymous and thus secure in the full BMW model. Cramer-Shoup encryption entails a four-tuple $(T_1, T_2, T_3, T_4)$ of elements of $G_1$. The proof of security entails four elements of $\mathbb{Z}_p$. Instantiated with the same parameters as above, the resulting group signature is 1364 bits long.

We emphasize that currently nothing is known about the complexity of the DDH problem in the ground field of an MNT curve and relying on this assumption seems risky. This question deserves further study.

## 9 Conclusions

We presented a group signature scheme based on the Strong Diffie-Hellman (SDH) and Linear assumptions. The signature makes use of a bilinear map $e : G_1 \times G_2 \to G_T$. When any of the curves described in [9] are used, the group $G_1$ has a short representation and consequently we get a group signature whose length is under 200 bytes — less than twice the length of an ordinary RSA signature (128 bytes) with comparable security. Signature generation requires no bilinear pairing computations, and verification requires a single pairing; both also require a few exponentiations with short exponents.

## Acknowledgments

The authors thank the anonymous referees of Crypto 2004 for their valuable feedback.

## References

[1] M. Abdalla, J. An, M. Bellare, and C. Namprempre. From identification to signatures via the Fiat-Shamir transform: Minimizing assumptions for security and forward-security. In L. Knudsen, editor, *Proceedings of Eurocrypt 2002*, volume 2332 of *LNCS*, pages 418–33. Springer-Verlag, May 2002.

[2] 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.

[3] G. Ateniese and G. Tsudik. Some open issues and directions in group signatures. In *Proceedings of Financial Cryptography 1999*, volume 1648, pages 196–211. Springer-Verlag, Feb. 1999.

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

[5] 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.

[6] 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.

[7] M. Bellare, H. Shi, and C. Zhang. Foundations of group signatures: The case of dynamic groups. Cryptology ePrint Archive, Report 2004/077, 2004. `http://eprint.iacr.org/`.

[8] 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.

[9] 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`.

[10] D. Boneh and H. Shacham. Group signatures with verifier-local revocation, 2004. Manuscript.

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

[12] 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.

[13] J. Camenisch and A. Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In M. Franklin, editor, *Proceedings of Crypto 2004*, LNCS. Springer-Verlag, Aug. 2004.

[14] 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, 1991.

[15] R. Cramer and V. Shoup. A practical public key encryption system provably secure against adaptive chosen ciphertext attack. In H. Krawczyk, editor, *Proceedings of Crypto 1998*, volume 1642 of *LNCS*, pages 13–25. Springer-Verlag, Aug. 1998.

[16] 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.

[17] 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.

[18] S. Galbraith and V. Rotger. Easy decision-diffie-hellman groups. Cryptology ePrint Archive, Report 2004/070, 2004. `http://eprint.iacr.org/`.

[19] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: A virtual machine-based platform for trusted computing. In *Proceedings of SOSP 2003*, pages 193–206, Oct. 2003.

[20] IEEE P1556 Working Group, VSC Project. Dedicated short range communications (DSRC), 2003.

[21] 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.

[22] A. Kiayias and M. Yung. Group signatures: Efficient constructions and anonymity from trapdoor-holders. Cryptology ePrint Archive, Report 2004/076, 2004. `http://eprint.iacr.org/`.

[23] A. Lysyanskaya, R. Rivest, A. Sahai, and S. Wolf. Pseudonym systems. In H. Heys and C. Adams, editors, *Proceedings of SAC 1999*, volume 1758 of *LNCS*, pages 184–99. Springer-Verlag, Aug. 1999.

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

[25] 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.

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

[27] K. Rubin and A. Silverberg. Supersingular Abelian varieties in cryptology. In M. Yung, editor, *Proceedings of Crypto 2002*, volume 2442 of *LNCS*, pages 336–53. Springer-Verlag, Aug. 2002.

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

[29] 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.

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

[31] 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.