Multisignature Schemes with Tight Reduction in the Plain Public-Key Model*

Duc Phong Le (dle@etud.univ-pau.fr)*
Alexis Bonnecaze (alexis.bonnecaze@esil.univmed.fr)†
Alban Gabillon (alban.gabillon@upf.pf) ‡

Abstract: A multisignature scheme allows a group of signers to cooperate to generate a compact signature on a common document. The length of the multisignature depends only on the security parameters of the signature schemes and not on the number of signers involved. The most efficient multisignature scheme known in regards to key setup requirements is constructed by Bellare and Neven at CCS'06 in the *plain public-key model*. In this paper, we present two new efficient multisignature schemes whose security is *tightly* related to the Diffie-Hellman problems in the random oracle model. Like Bellare-Neven scheme, our multisignature schemes are also proved secure against *rogue-key attacks* in the plain public key model. Our construction derives from variants of *EDL* signatures.

Keywords: Multisignatures, Diffie-Hellman problems, tight reduction, random oracle model, plain public key model

1 Introduction

A multisignature scheme enables multiple signers to jointly authenticate a document producing a fixed length of digital signature. The goal of a multisignature is to prove that each member of the stated group signed the message. It is up to a particular application to decide which group is required to sign a message. A verifier might reject a multisignature not because it is invalid, but because the verifier is not satisfied with the group which signed the message. Multisignatures can be applied to provide efficient batch verification of several signatures of the same message under different public keys, e.g. applications concerning the multi-cast communication: IP Multi-cast, Peer-to-Peer file sharing, mobile ad hoc networks, etc.

The notion of multisignatures was first introduced by Itakura and Nakamura in [IN83], and has been followed by many other research works [Oka88, Boy89]. Those initial schemes were not very efficient and in particular there was no formal notion of security. In fact, the effective attacks on multisignature schemes have succeeded due to weaknesses related to key setup protocol, in particular the ability to mount a *rogue key attack*. Such an attack

* Laboratoire LIUPPA, Université de Pau et des Pays de l'Adour,
Tel : 33 (0) 5 58 51 37 18.
† Laboratoire IML, Université de la Méditerranée, 13288 Marseille cedex 09.
‡ Laboratoire GePaSud, Université de la Polynésie Française, 98702 FAA'A - Tahiti - Polynésie française.
* This work was supported by Conseil Général des Landes and the French Ministry for Research under Project ANR-07-SESU-FLUOR.

can be realized whenever an adversary is allowed to choose his public key as he wishes. Typically, the adversary chooses his public key as a function of public keys of honest user, allowing him to produce forgeries easily.

The first formal security model for multisignatures was formalized by Micali et al. in [MOR01]. Their scheme requires a *dedicated key generation* protocol amongst potential signers for the purpose of counteracting rogue-key attacks. This means that the set of potential signers must engage in an interactive key generation protocol, as a pre-processing step, to provide to each a public and secret key. Those requirements are *impractical*. Then, Boldyreva introduced a variant of their model by making use of the *knowledge of secret key* (KOSK) assumption which requires expensive zero-knowledge (ZK) proofs of knowledge (POKs) performed with the CA. It allows us to ensure that an user can only use its public key which is corresponding its secret key. This assumption, however, is not realized by existing public key infrastructure (PKI).

Plain public key model. In setting for multisignature schemes, the set of potential users should be dynamic. Users can choose his public key as they wish and may register keys at any time. In [BN06], Bellare and Neven discuss the drawbacks of multisignature schemes in [MOR01, Bol04, LOS+06] in detail and show that it is possible to dispense with both the dedicated key generation protocol [MOR01] and the KOSK assumption [Bol04, LOS+06]. They presented a multisignature scheme which is provably secure against rogue-key attacks in the *plain public-key model*, meaning that key registration with a Certification Authority (CA) requires nothing more than that each signer has a (certified) public key. Their model allows users to register keys at any time, concurrently with other users.

Tight reduction. As Micali and Reyzin [MR02] put it, if the reduction is efficient and hence the relative hardness of forging and that of breaking the underlying computational assumption is close, we call the reduction *tight*. If the reduction is less efficient, we call it *close*, and if it is significantly less efficient, we call it *loose*. Intuitively, a tight reduction means that the underlying cryptographic problem is almost as hard to solve as the scheme to break. Up to date, there is no discrete-logarithm-based multisignature scheme proposed with *tight security reduction*. The security proof of such a multisignature scheme [MOR01] is based on the "forking lemma" technique of Pointcheval and Stern [PS96] or the variant of forking lemma [BN06]. The disadvantage of this technique is that the so-obtained security reductions are loose.

Our contribution. We approach the problem of multisignatures with the goal of creating efficient multisignature schemes with tight security reduction under Diffie-Hellman assumptions in the random oracle model [BR93]. In this paper, we propose two multisignature schemes: the security of our first scheme relies on the hardness of the computational Diffie-Hellman (CDH) problem; the security of second scheme is based on the hardness of the decisional Diffie-Hellman (DDH) problem. Our multisignature schemes are provably secure, even against rogue-key attacks, in the plain public-key setting. Our constructions are based on variants of the *EDL* signature scheme presented in [KW03][CM05]. Basically, our schemes are interactive, i.e. we require interactions among cosigners during multisignature generation process.

Related works. The *EDL* signature scheme was independently presented by Chaum and Pedersen in [CP92] and Jakobsson and Schnorr in [JS99]. However, the first tight security reduction for this scheme was only showed by Goh and Jarecki in [GJ03]. The scheme

was then improved by Katz-Wang [KW03] and Chevallier-Mames [CM05] for shorter signatures. To date, they are the only signatures whose security is tightly related to the Diffie-Hellman problems in the random oracle. The technique to obtain a tight security reduction in their schemes is make use of a zero-knowledge proof of equality of discrete logarithms [CEG87].

Bellare-Neven scheme [BN06], based on the Schnorr signatures [Sch91], is the first multisignature scheme provably secure against rogue-key attacks in the plain public key model. Their scheme is more efficient than those in [MOR01][Bol04][LOS+06] in terms of key registration with a CA. However, the security reduction for their scheme (in the random oracle model), relies on the *general forking lemma* [BN06], is loose.

The shortest multisignatures which consist in only one group element were presented by Boldyreva in [Bol04]. Moreover, her multisignature scheme is *non-interactive*. However, this scheme is loosely related to the CDH problem. Besides, as pointed out above, its security model makes use of the KOSK assumption and it thus is *impractical*. Even through, there is also a multisignature scheme of Lu et al. [LOS+06] whose security is proved in the standard model. However, also as Boldyreva scheme, their scheme also makes use of KOKS assumption. Besides, the size of the system parameters in the scheme [LOS+06] is very large, namely 160 group elements.

Organization. The rest of the paper is organized as follows. Section 2 provides some preliminaries about bilinear maps, Diffie-Hellman problems and the security model for multisignatures. In Section 3, we present our construction based CDH problem and we analyze its security in Section 4. We present our multisignature scheme based DDH problem in Section 5. Finally, we conclude the paper in Section 6.

2 Preliminary

2.1 Bilinear Map

Our first multisignature scheme uses a bilinear map, which is often called a pairing, to implement a decision procedure for the *Diffie-Hellman problem*. Typically, the pairing used is a modified Weil or Tate pairing. In this section, we briefly review the necessary facts about bilinear maps.

Let $\mathbb{G}$, $\mathbb{G}_T$ be cyclic groups of prime order p. A map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is called an *admissible pairing* if it satisfies the following properties:

1. **bilinearity:** for all $g_1,\ g_2 \in \mathbb{G}$ and $a,\ b \in \mathbb{Z},\ e(g_1^a,\ g_2^b) = e(g_1,\ g_2)^{ab}$;
2. **non-degeneracy:** if g is a generator of $\mathbb{G}$, then $e(g, g)$ is a generator of $\mathbb{G}_T$;
3. **computable:** there exists an efficient algorithm to compute $e(g_1, g_2)$ for $\forall g_1, g_2 \in \mathbb{G}$.

see [JN03] for a more detailed discussion about bilinear maps and bilinear groups.

2.2 Computational Assumptions

The security of our schemes is based on the hardness of the Diffie-Hellman problems. Let $\mathbb{G}$ be a cyclic group of prime order p and let g be a generator of $\mathbb{G}$.

Computational Diffie-Hellman. Informally, the CDH problem is to find g^{ab}, given $(g^a, g^b) \in \mathbb{G}$ as inputs, where $a, b \xleftarrow{\$} \mathbb{Z}_p^*$. An algorithm $\mathcal{A}$ has an advantage ϵ in solving

the CDH problem in $\mathbb{G}$ if

$$Pr\left[\mathcal{A}(g, g^a, g^b) = g^{ab} : g \overset{R}{\leftarrow} \mathbb{G}\ ;\ a, b \overset{R}{\leftarrow} \mathbb{Z}_p^*\right]$$

is at least ϵ. We say that the CDH problem is $(t, \epsilon) - hard$ in $\mathbb{G}$ if there exists no algorithm $\mathcal{A}$ which running in time at most t have advantage ϵ in solving the CDH problem in $\mathbb{G}$.

Decisional Diffie-Hellman. The DDH problem is informally to distinguish between tuples of the form (g^a, g^b, g^{ab}) (called **DDH triples** or **DDH tuples**), where $a, b \overset{R}{\leftarrow} \mathbb{Z}_p^*$ and tuples of the form (g^a, g^b, g^c), where $a, b, c \overset{R}{\leftarrow} \mathbb{Z}_p^*$.

A distinguishing algorithm $\mathcal{A}$ has an advantage ϵ in solving the DDH problem in $\mathbb{G}$ if

$$\left|Pr\left[\mathcal{A}(g^a, g^b, g^{ab}) = 1\right] - Pr\left[\mathcal{A}(g^a, g^b, g^c) = 1\right] : a, b, c \overset{R}{\leftarrow} \mathbb{Z}_p^*\right|$$

is at least ϵ. We say that the DDH problem is $(t, \epsilon) - hard$ in $\mathbb{G}$ if there exists no distinguishing algorithm $\mathcal{A}$ which running in time at most t have advantage ϵ in solving the DDH problem in $\mathbb{G}$.

2.3 Security Model for Multisignatures

The notion of security for an *interactive* multisignature scheme in the plain public key model is introduced by Bellare and Neven in [BN06]. We consider the following game associated to a multisignature scheme, which consists of four algorithms Setup, Keygen, Multsign, Vf, and an adversary $\mathcal{A}$:

- **Setup:** Adversary $\mathcal{A}$ is given the system parameters params which are obtained by running the Setup algorithm, params $\overset{R}{\leftarrow}$ Setup and a target public key pk^*.

- **Attack:** Adversary $\mathcal{A}$ requests a multisignature, under the challenge key pk^*, on a message m and a multiset $Pk = \{pk_1, \ldots, pk_n\}$ of purported cosigners L, where pk^* occurs in Pk at least once. $\mathcal{A}$ may either choose these public keys arbitrarily or as a function of pk^*. In interacting with the honest signer, $\mathcal{A}$ will play the role of rest signers in L and fully controls all messages exchanged in the network. Further, the forger can also schedule an arbitrary number of protocol instances concurrently, interacting with "clones" of the honest signer, where each clone maintains its own state and uses its own coins but all use the keys pk^*, sk^* and follow the protocol to compute their responses to received messages. For some (Pk, m), $\mathcal{A}$ receives either $\perp$ or a multisignature signature σ from the honest signer in response.

- **Forgery:** Eventually, after a polynomial number of queries, $\mathcal{A}$ outputs a forged multisignature σ^* on the input message m^* given by signers in L. $\mathcal{A}$ is said to win the game if $\mathsf{Vf}(m^*, L, \sigma^*) = 1$, pk^* is in the multiset $Pk = \{pk_1, \ldots, pk_n\}$ of purported cosigners L and $\mathcal{A}$ has never requested to execute the signing query on m^* with L.

We define MS Adv($\mathcal{A}$) to be the probability that the adversary $\mathcal{A}$ wins in the above game, taken over the coin tosses made by $\mathcal{A}$ and the challenger.

Definition 2.1 *An adversary $\mathcal{A}$ $(t, q_S, q_H, N, \epsilon)$-breaks multisignature scheme in the random oracle model if $\mathcal{A}$ runs in time at most t, $\mathcal{A}$ makes at most q_S signing queries with the honest signer, at most q_H random oracle queries, the number of signers in L involved in any signing query or in the forgery is at most N, and $\mathsf{MS\,Adv}(\mathcal{A})$ is at least ϵ. A multisignature scheme is said to be $(t, q_S, q_H, N, \epsilon)$-secure in the random oracle model if no forger $(t, q_S, q_H, N, \epsilon)$-breaks it.*

We stress that this security model is only for *interactive* multisignature schemes. In *non-interactive* multisignature schemes, there is no interaction between signers during multisignature generation process, an adversary is thus required that he has never requested to execute the signing query on m^* from the honest user.

3 A Multisignature Scheme Based on the CDH Problem

3.1 The Chevallier-Mames Signature Scheme

In order to give some intuition into our scheme, we briefly recall the variant of *EDL* signature scheme presented in [CM05]. Let $\mathbb{G}$ be a cyclic group of prime order p, g be a generator of $\mathbb{G}$ and let $\mathcal{H}, \mathcal{G}$ be two collision-resistant hash functions. To sign a message m, a signer U, having secret and public keys pair (x, y), does as follows:

- chooses $k \in \mathbb{Z}_p$ at random;
- computes $u = g^k$, $h = \mathcal{H}(u)$, $z = h^x$ and $v = h^k$;
- queries $c = \mathcal{G}(m, g, h, y, z, u, v)$ and computes $s = k + cx$;
- outputs $\sigma = (z, s, c) \in \mathbb{G} \times \mathbb{Z}_p^2$ as the signature of m.

To verify a signature $\sigma = (z, s, c)$ for m, one computes $u' = g^s y^{-c}$, $h' = \mathcal{H}(u')$ and $v' = h'^s z^{-c}$. The signature σ is accepted iff $c = \mathcal{G}(m, g, h', y, z, u', v')$.

The Chevallier-Mames signature scheme [CM05] is the most efficient in variants of *EDL* scheme [CP92, JS99, GJ03, KW03] under CDH assumption. For our goal of creating a new multisignature scheme, the Chevallier-Mames signatures first may be slightly modified as follows: let a signature of a message m under public key $y \in \mathbb{G}$ be a quadruplet $(u, v, z, s) \in \mathbb{G}^3 \times \mathbb{Z}_p$ such that $g^s = uy^c$ and $h^s = vz^c$, where $h = \mathcal{H}(u)$ and $c = \mathcal{G}(m, g, h, y, z, u, v)$. In order to aggregate individual signatures of a common message m, (u_i, v_i, z_i, s_i), for $1 \leq i \leq n$ under public keys $PK = \{y_1, y_2, \cdots, y_n\}$, we may be let a multisignature be a tuple $(u, v, s, \{z_i\}_{i=1}^n)$ such that: $g^s = u \cdot \prod_{i=1}^n y_i^{c_i}$ and $h^s = v \cdot \prod_{i=1}^n z_i^{c_i}$, where $u = \prod_{i=1}^n u_i, v = \prod_{i=1}^n v_i$ and $s = \sum_{i=1}^n s_i$. Because a value c_i is different in the exponent of each z_i contributed by each signer, we cannot aggregate individual shares z_i. The size of the multisignature thereby grows linearly with the number of signers. To solve this problem, we propose to use a pairing (bilinear map) whose cost is equivalent five exponential operations [BKLS02]. A multisignature may be a triple $(u, z, s) \in \mathbb{G}^2 \times \mathbb{Z}_p$ such that: $g^s = u \cdot \prod_{i=1}^n y_i^{c_i}$ and $e(z, g) = e(h, \prod_{i=1}^n y_i)$, where $h = \mathcal{H}(u)$, $c_i = \mathcal{G}(y_i, L, u, m, g, h)$. The values of u, z (and s) are typically computed as the product (the sum resp.) of individual shares of u_i, z_i (of s_i resp.) contributed by each signer.

3.2 Our Multisignature Scheme

In describing the scheme, we assume the signers directly send and receive messages to each other over a point-to-point network. Like in [BN06], to avoid using the rewinding technique in security proof, our scheme requires an additional communication round between signers, in which each signer first makes an additional random oracle query on its individual share u and then sends this *challenge* to every other signer before sending u. This prevents the forger to know the value of individual share u before the simulator does. The simulator thereby could imitate the oracle so as to produce commitments and challenges simultaneously.

Let $\mathbb{G}, \mathbb{G}_T$ be cyclic groups of prime order p in which $\mathbb{G}$ provides admissible parings, let k be a security parameter. Three cryptographic hash functions: $\mathcal{H}_0 : \mathbb{G} \to \{0,1\}^{l_0}, \mathcal{H}_1 : \mathbb{G} \to \mathbb{G}$ and $\mathcal{G} : \{0,1\}^* \to \mathbb{Z}_p$. We remark that $\mathcal{H}_0$, $\mathcal{H}_1$ and $\mathcal{G}$ will be viewed as random oracles in our security proof. The multisignature scheme $\mathsf{MS} = \mathsf{Setup}, \mathsf{Keygen}, \mathsf{Multsign}, \mathsf{Vf}$ works as follows:

Parameter generation (Setup): A trusted center generates a random generator $g \in \mathbb{G}^*$ and publishes $\mathsf{params} = (\mathbb{G}, \mathbb{G}_T, e, g, \mathcal{H}_0, \mathcal{H}_1, \mathcal{G})$ as system wide parameters.

Key generation (Keygen): On input 1^k, each signer picks a random number $x \xleftarrow{R} \mathbb{Z}_p$ as his private key. The corresponding public key is $y = g^x$.

Signing (Multsign): Suppose that $L = \{P_1, P_2, \ldots, P_n\}$ is a group of n signers that wish to sign a common message m, each having as input its own public and secret key as well as a multiset of public keys $Pk = \{y_1, y_2, \ldots, y_n\}$ of the other signers. We also stress that the signers $P_1, \ldots, P_n$ are merely local references to co-signers, defined by one signer within one protocol instance. The signing process, which is *interactive*, consists of four rounds:

Round 1. Each signer $P_i \in L$:

- picks a random number $r_i \in Z_p$;
- computes its individual commitment $u_i = g^{r_i}$;
- queries $\mathcal{H}_0$ to compute the challenge $h_i = \mathcal{H}_0(u_i)$;
- sends h_i to every other signer.

Round 2. Each signer $P_i \in L$:

- receives h_j from signer j, for $1 \leq j \leq n, j \neq i$;
- sends u_i to signer j.

Round 3. Each signer $P_i \in L$:

- receives u_j from signer j, for $1 \leq j \leq n, j \neq i$;
- checks whether $h_j = \mathcal{H}_0(u_j)$ for all $1 \leq j \leq n, j \neq i$. If not, abort the protocol. Otherwise,
- computes $u = \prod_{i=1}^{n} u_i$, $h = \mathcal{H}_1(u)$ and $z_i = h^{x_i}$.
- queries $c_i = \mathcal{G}(y_i, u, Pk, m, g, h)$ and computes $s_i = r_i + x_i c_i \bmod p$.
- sends to signer j: $z_i,\ s_i$, for $1 \leq j \leq n, j \neq i$.

Round 4. Each signer $P_i \in L$:

- receives $z_j,\ s_j$ from signer j, for $1 \leq j \leq n, j \neq i$;
- computes $z = \prod_{i=1}^{n} z_i,\ s = \sum_{i=1}^{n} s_i \bmod p$;
- outputs the signature $\sigma = (u, z, s)$;

Verification (Vf): To verify a signature σ of a message m of a group L, whose public keys is multiset $Pk = \{y_1, \ldots, y_n\}$, one does as follows:

- Compute $h = \mathcal{H}_1(u)$ and $c_i = \mathcal{G}(y_i, u, Pk, m, g, h)$ for all $1 \leq i \leq n$;
- Check whether:

$$g^s = u \cdot \prod_{i=1}^{n} y_i^{c_i} \quad \text{and} \quad e(z, g) = e(h, \prod_{i=1}^{n} y_i).$$

3.3 Efficiency

Our multisignatures consists of three elements (two elements in $\mathbb{G}$ and one element in $\mathbb{Z}_p$). In above description, we use a "symmetric" pairing which is found on supersingular curves, a very limited class of curves. In practice, the bit-length of representation in $\mathcal{G}$ is about 300 bits. Thus, the size of our signatures is 760 bits to provide the 1024-bit RSA level of security [GPS06]. However, our scheme can easily be generalized to work with an "asymmetric" pairing of the form $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$, allowing the use of wider classes of elliptic curves [GPS06]. This allows us to take advantage of certain families of algebraic curves in order to obtain the shortest possible signatures. Specifically, elements of $\mathbb{G}_1$ have a short representation over the ground field $\mathbb{F}_p$ whereas elements of $\mathbb{G}_2$, which may be defined over an extension field $\mathbb{F}_{p^\alpha}$, have a longer representation than those of $\mathbb{G}_1$. The group element representation sizes in $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ are 160 bits, $6 \cdot 160$ bits and $6 \cdot 160$, respectively [GPS06]. Thus, the size of our signatures is 480 bits. On the other hand, public keys of signers belong the group $\mathbb{G}_2$, and we will need an efficient computable isomorphism $\psi : \mathbb{G}_2 \mapsto \mathbb{G}_1$ to map public keys y_i to elements in $\mathbb{G}_1$ [SV07] for the purpose of checking the first verification equality. The size of public key y_i of each signer thus increases to $6 \cdot 160$ bits.

4 Security Analysis

In this section, we reduce the security of the proposed multisignature scheme to the CDH problem in the group $\mathbb{G}$ with bilinear map e. The main technique used to obtain a tight proof of security is make use of proving equality of discrete logarithms (see [GJKW07] for a discussion more details). Let N be the maximum number of signers which participate signing in one protocol instance, the following theorem implies that the proposed multisignature scheme is secure if the CDH assumption is hold in $\mathbb{G}$.

Theorem 4.1 *The proposed multisignature scheme is $(t, q_H, q_S, N, \epsilon)$-unforgeable if the CDH problem is (t', ϵ')-unforgeable in $\mathbb{G}$, where*

$$\epsilon' \geq \epsilon - \frac{(q_H + Nq_S + 1)^2}{2^{l_0}} + \frac{q_S(2q_H + 3Nq_S)}{2^k}$$

and

$$t' \leq t + 6q_S t_{exp} + O((q_S + q_H)(1 + q_H + Nq_S)),$$

where t_{exp} is the time of an exponentiation in $\mathbb{G}$.

Proof We are given a group $\mathbb{G}$ and a CDH challenge (g, g^x, g^a). Let $\mathcal{A}$ be a polynomial time forger that (t, q_H, q_S, ϵ)-breaks the proposed scheme. We need to construct an algorithm $\mathcal{B}$ which, by interacting with the adversary $\mathcal{A}$, (t', ϵ')-breaks this challenge, i.e. to find g^{ax}. The forger $\mathcal{A}$, after q_H hash queries to random oracles ($\mathcal{H}_0$, $\mathcal{H}_1$ and $\mathcal{G}$) and q_S signature queries, is able to produce a multisignature forgery with probability ϵ within time t.

Assume that $\mathcal{A}$ is trying to attack the honest signer P^*. $\mathcal{B}$ runs the forger $\mathcal{A}$ on input system parameters and target public key $y^* = g^x$. Algorithm $\mathcal{B}$ initializes three list H, U and G to simulate random oracles $\mathcal{H}_0, \mathcal{H}_1, \mathcal{G}$, respectively. Like [BN06], we also make use of a list T which assigns a unique index $1 \leq i \leq q_H + Nq_S$ to each public key y occurring either as a cosigner's public key in one of $\mathcal{A}$'s signature queries, or as the first item in the argument of one of $\mathcal{A}$'s queries to $\mathcal{G}$. Algorithm $\mathcal{B}$ uses a counter *ctr* indicating the current index of this list, initially set to zero. $\mathcal{B}$ assigns $\mathsf{T}[y^*] \leftarrow 0$. It responds to $\mathcal{A}$'s oracle queries, essentially, at random as follows:

Queries to $\mathcal{H}_0$. In response to a query $\mathcal{H}_0(u_i)$, $\mathcal{B}$ first checks if the output of $\mathcal{H}_0$ on this input has been previously defined. If so, $\mathcal{B}$ returns the previously assigned value. Otherwise, $\mathcal{B}$ returns with a value chosen uniformly at random from $\{0,1\}^{l_0}$. All queries u_i are stored in the list H.

Queries to $\mathcal{H}_1$. In response to a query of the forger $\mathcal{A}$ to $\mathcal{H}_1(u)$, algorithm $\mathcal{B}$ generates a random number $d \in Z_p$, and returns $(g^a)g^d$. All queries u are stored in the list U.

Queries to $\mathcal{G}$. In response to a query $\mathcal{G}$, we first parse the argument of the query into two portion as Y, Q. If $\mathsf{T}[Y]$ is undefined then $\mathcal{B}$ increases *ctr* and sets $\mathsf{T}[Y] \leftarrow ctr$. If $\mathsf{G}[ctr, Q]$ is undefined, then $\mathcal{B}$ assigns $\mathsf{G}[i, Q]$, for all $1 \leq i \leq q_H + Nq_S$ with random numbers, and picks in advance at random as $e_1, \ldots, e_{q_H+q_S} \in \mathbb{Z}_p$ to assign for $\mathsf{G}[0, Q]$.

Signing query on m with group of users L: Signature queries to the honest signer P^* consists of three rounds. First, the adversary provides m, L to P^* and receives the individual challenge h^* from P^* in response. Second, playing the role of rest signer, the adversary $\mathcal{A}$ provides the challenges h_i to P^* and receives u^* from P^* in response. Third, the adversary provides the commitments u_i to P^* and receives z^*, s^* from P^* in response. Note that, in the simulation, it is not the adversary providing the joint commitment u to simulator, we do not thus need to use rewinding. In detail, answering signature queries works as follows:

First, $\mathcal{B}$ checks whether $P^* \notin L$, if so algorithm $\mathcal{B}$ returns $\perp$ to $\mathcal{A}$. If not, it parses the public keys of signers in L as $Pk = \{y_1 = y^*, y_2, \ldots, y_n\}$. Then, $\mathcal{B}$ checks whether $\mathsf{T}[y_i]$, for $i \in \{2, \ldots, n\}$, has already been defined. If not, it increases *ctr* and sets $\mathsf{T}[y_i] \leftarrow ctr$. Then, $\mathcal{B}$ sets c_1 at random as $e_1, \ldots, e_{q_H+q_S}$ in advance. $\mathcal{B}$ generates $(\gamma, s_1) \in \mathbb{Z}_p^2$ at random, computes $u_1 = g^{s_1} y^{-c_1}$. It sets $h_1 = \mathcal{H}_0(u_1)$ and sends to all signers.

After receiving $h_2, \cdots, h_n$ from the adversary $\mathcal{A}$, $\mathcal{B}$ looks up in the list H for values u_j such that $h_i = \mathcal{H}_0(u_j)$. If multiple such values are found for some i, the algorithm $\mathcal{B}$ stops (**Event 1**). If no such value was found for some i then it sets $alert \leftarrow true$ and sends u_1 to all cosigners; otherwise, $\mathcal{B}$ computes $u = \prod_{i=1}^{n} u_i$. If $\mathcal{H}_1(u)$ is already set, algorithm $\mathcal{B}$ fails and stops (**Event 2**). Else, algorithm $\mathcal{B}$ sets $h = \mathcal{H}_1(u) = g^\gamma$ and computes $z_1 = y_1^\gamma = (g^x)^\gamma = h^x$, remark that $DL_g(y) = DL_h(z)(= x)$. Then, $\mathcal{B}$ checks whether $\mathsf{G}[0, Q]$ has already been defined for $Q = \langle u, Pk, m, g, h\rangle$. If so, it fails and stops (**Event 3**). If not, it sets $\mathcal{G}(y_1, u, Pk, m, g, h) = \mathsf{G}[0, Q] = c_1$, randomly chooses $\mathsf{G}[i, Q] \stackrel{R}{\leftarrow} \mathbb{Z}_p$ for all $1 \leq i \leq q_H + Nq_S$ and sends u_1 to all cosigners.

After receiving $u_2, \ldots, u_n$ from $\mathcal{A}$, $\mathcal{B}$ verifies that $h_i = \mathcal{H}_0(u_i)$ for all $1 \leq i \leq n$. If not, it returns $\perp$ to $\mathcal{A}$. If $alert = true$, $\mathcal{B}$ fails and stops (**Event 4**). Else, it sends (z_1, s_1) to all cosigners.

After receiving $(z_2, s_2), \cdots, (z_n, s_n)$ from cosigners ($\mathcal{A}$), $\mathcal{B}$ computes $z = \prod_{i=1}^{n} z_i$ and $s = \sum_{i=1}^{n} s_i$ and returns the valid signature (u, z, s).

As we can see, this simulator is valid, except for some events:

- **Event 1**: In this case, there exists two values $u_i \neq u_i'$ such that $h_i = \mathcal{H}_0(u_i) = \mathcal{H}_0(u_i')$ for some i, i.e, there is at least one collision occurred in $\mathcal{H}_0$. As outputs of $\mathcal{H}_0$ are chosen at random from $\{0,1\}^{l_0}$ and since there are at most $q_{\mathcal{H}_0} + Nq_S$ queries to $\mathcal{H}_0$, the probability that at least one collision occurs is upper bounded by $((q_{\mathcal{H}_0} + Nq_S)(q_{\mathcal{H}_0} + Nq_S + 1)/2)/2^{l_0} \leq (q_{\mathcal{H}_0} + Nq_S + 1)^2/2^{l_0+1}$.
- **Event 2**: As u is a random element in $\mathbb{G}$, the probability that the $\mathcal{H}_1(u)$ is already set is less than $(q_{\mathcal{H}_1} + Nq_S)/p$, for one signature query. For q_S signature queries, the failure probability is thus upper bounded by $q_S(q_{\mathcal{H}_1} + Nq_S)/p \leq q_S(q_{\mathcal{H}_1} + Nq_S)/2^k$.
- **Event 3**: The algorithm $\mathcal{B}$ only aborts at event 3 if it has run into an input string $\langle 0, u, Pk, m, g, h\rangle$ on which $\mathcal{G}$ has been already queried. We distinguish between the case that $\mathcal{H}_0(u_1)$ was previously queried by the forger, and the case that it was not. In the first case, $\mathcal{A}$ probably knows u and may have deliberately queried $\mathcal{G}(y, u, Pk, m, g, h)$ for some y. But since u_1 was chosen by $\mathcal{B}$ independently from $\mathcal{A}$'s view at the beginning of the signing protocol, the probability that $\mathcal{A}$ queried $\mathcal{H}_0(u_1)$ is at most $(q_{\mathcal{H}_0} + Nq_S)/p$, for one signature query. In the second case, $\mathcal{A}$'s view is completely independent of u_1, and hence of u. The probability that u occurred by chance in a previous query to $\mathcal{G}$ or was set by $\mathcal{B}$ in one of the $i-1$ previous signature simulations is at most $(q_{\mathcal{H}_0} + q_S)/p$, for one signature query. For q_S signature queries, the failure probability is thus upper bounded by $q_S((q_{\mathcal{H}_0} + Nq_S) + (q_{\mathcal{H}_0} + q_S))/p \leq 2q_S(q_{\mathcal{H}_0} + Nq_S)/2^k$.
- **Event 4**: In this case, $\mathcal{A}$ must have predicted the value of $\mathcal{H}_0(u_i)$ for at least one $1 \leq i \leq n$, which it can do with probability at most $N/2^{l_0}$, for one signature query. For q_S signature queries, the failure probability is thus upper bounded by $q_S N/2^{l_0}$.

As a conclusion, except with a failure probability:

$$\epsilon_{stop} = \frac{(q_{\mathcal{H}_0} + Nq_S + 1)^2}{2^{l_0+1}} + \frac{q_S(q_{\mathcal{H}_1} + Nq_S)}{2^k} + \frac{2q_S(q_{\mathcal{H}_0} + Nq_S)}{2^k} + \frac{q_S N}{2^{l_0}},$$

the simulation is successful.

Eventually, $\mathcal{A}$ halts and outputs an attempted forgery $\sigma = (\hat{u}, \hat{z}, \hat{s})$ on some message $\hat{m}$ along with $L = \{P^*, P_2, \cdots, P_n\}$. It must not previously have requested a signature on $\hat{m}$ with L. In addition, it outputs the private keys $(x_2, \cdots, x_n)$ for all secret keys except the key x of the challenge P^*. Algorithm $\mathcal{B}$ first computes additional random oracle queries $\mathcal{G}_1(y_i, \hat{u}, Pk, \hat{m}, g, \hat{h})$ for $1 \leq i \leq n$, thereby making sure that $\mathsf{G}[y_i]$ is defined. Let $\hat{h} = \mathcal{H}_1(\hat{u})$, $\mathcal{B}$ computes $\hat{z}_1 = \hat{z}/\prod_{i=2}^{n} \hat{h}^{x_i}$. If $\mathcal{A}$'s forgery is valid, the simulator returns $(\hat{u}, \hat{z}, \hat{s}, \hat{h}, \hat{z}_1)$.

We argue that, with all but negligible probability, $\hat{z}_1 = \hat{h}^x$; if so, say $\hat{z}_1$ is *good*. Indeed, if $\hat{z}_1$ is not good then for any A, B there is at most one possible value of c for which there exists an s satisfying $A = g^s y^c$ and $B = \hat{h}^s \hat{z}_1^c$ (*lemma* 1 in [GJKW07]). If $\hat{z}_1$ is not good, then, for any hash query $\mathcal{G}(y_1, \hat{u}, Pk, \hat{m}, g, \hat{h})$ made by $\mathcal{B}$ the probability that the query returns a c for which there exists an s as above is at most $1/2^k$. It follows that the probability that $\mathcal{B}$ outputs a valid forgery where $\hat{z}_1$ is not good is at most $q_{\mathcal{G}}/2^k$. The probability that $\mathcal{B}$ outputs a valid forgery such that $\hat{z}_1$ is good at least $\epsilon - \epsilon_{stop} - q_{\mathcal{G}}/2^k$. In that case, the CDH challenge is solved as follows:

$$\hat{z}_1/y_1^d = \hat{h}^{x_1}/y_1^d = (g^a g^d)^x/(g^x)^d = g^{ax},$$

as desired.

Summing the probabilities, we can that the algorithm $\mathcal{B}$ solve the CDH problem with probability:

$$\begin{aligned}
\epsilon' &\geq \epsilon - \epsilon_{stop} - (q_{\mathcal{G}} + 1)/2^k \\
&\geq \epsilon - \frac{(q_{\mathcal{H}_0} + Nq_S + 1)^2}{2^{l_0+1}} - \frac{q_S(q_{\mathcal{H}_1} + Nq_S)}{2^k} - \frac{2q_S(q_{\mathcal{H}_0} + Nq_S)}{2^k} - \frac{q_S N}{2^{l_0}} - \frac{q_{\mathcal{G}}}{2^k} \\
&\geq \epsilon - \frac{(q_H + Nq_S + 1)^2}{2^{l_0}} - \frac{q_S(2q_H + 3Nq_S)}{2^k}
\end{aligned}$$

and the running time t' satisfies

$$t' \leq t + 6q_S t_{exp} + O((q_S + q_H)(1 + q_H + Nq_S)),$$

where $q_H = q_{\mathcal{H}_0} + q_{\mathcal{H}_1} + q_{\mathcal{G}}$, t_{exp} is the time of an exponentiation in $\mathbb{G}$.

5 A Multisignature Scheme Based on the DDH Problem

In the previous scheme, our scheme makes use of groups with bilinear maps. In this section, we present a more efficient multisignature scheme which relies on decisional Diffie-Hellman problem, stronger than CDH assumption, in group completely arbitrary. Our construction is based on Katz-Wang signature scheme [KW03] that works as follows:

Let $\mathbb{G}$ be a cyclic group of prime order p, g be a generator of $\mathbb{G}$, $h \in \mathbb{G}$ chosen randomly and let $\mathcal{H} : \{0,1\}^* \to \{0,1\}^{l_0}$ be a hash function. A Katz-Wang signature of a message m under public keys (y_1, y_2) is a triplet (A, B, s), such that $g^s = Ay_1^c$ and $h^s = By_2^c$, where $A = g^r$, $B = h^r$ and $c = \mathcal{H}(A, B, m)$.

Note that the Katz-Wang signature [KW03] consists of only two elements (c, s), we however modified slightly their scheme for easy extending to multisignatures. The idea of using the Katz-Wang signatures for constructing multisignatures was also reminded by Bellare and Neven in section 6 of [BN06] as further results.

5.1 Our Multisignature Scheme

As before, we assume that $\mathbb{G}, \mathbb{G}_T$ be cyclic groups of prime order p, k be a security parameter. Two cryptographic hash functions: $\mathcal{H} : \mathbb{G} \to \{0,1\}^{l_0}$ and $\mathcal{G} : \{0,1\}^* \to \mathbb{Z}_p$. Our second scheme is defined as follows:

Parameter generation. A trusted center chooses a generator $g \in \mathbb{G}^*$ and $h \in \mathbb{G}$ at random. It then publishes $params = (\mathbb{G}, e, g, h, \mathcal{H}, \mathcal{G})$ as system wide parameters.

Key generation. On input 1^k, each signer picks a random number $x \overset{R}{\leftarrow} \mathbb{Z}_p$ as his private key. The corresponding public keys are $PK = (y_1, y_2)(= (g^x, h^x))$.

Signing (Multsign): Suppose that $L = \{P_1, P_2, \ldots, P_n\}$ is a group of n signers that wish to sign a common message m, each having as input its own public and secret key as well as a multiset of public keys $Pk = \{PK_1, \ldots, PK_n\}$ of the other signers. We also stress that the signers $P_1, \ldots, P_n$ are merely local references to co-signers, defined by one signer within one protocol instance. The signing process, which is *interactive*, consists of four rounds, where in each round signers send (and receive) a message to (from resp.) each other signer.

Round 1. Each signer $P_i \in L$:

- picks a random number $r_i \in Z_p$;
- computes its individual commitments $u_i = g^{r_i}$ and $v_i = h^{r_i}$, then queries $\mathcal{H}$ to compute challenges $h_i = \mathcal{H}(u_i)$ and $t_i = \mathcal{H}(v_i)$;
- sends h_i, t_i to every other signer.

Round 2. Each signer $P_i \in L$:

- receives h_j, t_j from signer j, for $1 \leq j \leq n, j \neq i$;
- sends u_i, v_i to signer j.

Round 3. Each signer $P_i \in L$:

- receives u_j, v_j from signer j, for $1 \leq j \leq n, j \neq i$;
- checks whether $h_j = \mathcal{H}(u_j)$ and $t_j = \mathcal{H}(v_j)$ for all $1 \leq j \leq n, j \neq i$. If not, abort the protocol. Otherwise, computes $u = \prod_{i=1}^{n} u_i$ and $v = \prod_{i=1}^{n} v_i$.
- queries $c_i = \mathcal{G}(PK_i, u, v, Pk, m, g, h)$ and computes $s_i = r_i + x_i c_i \bmod p$.
- sends to signer j: s_i.

Round 4. Each signer $P_i \in L$:

- receives s_j from signer j;
- computes $s = \sum_{i=1}^{n} s_i \bmod p$;
- outputs the signature $\sigma = (u, v, s)$;

Verification. Given the valid signature σ, list of group of users L and message m, the verifier computes $c_i = \mathcal{G}(PK_i, u, v, Pk, m, g, h)$ for all $1 \leq i \leq n$ and tests whether:

$$g^s = u \cdot \prod_{i=1}^{n} y_{1i}^{c_i} \quad \text{and} \quad h^s = v \cdot \prod_{i=1}^{n} y_{2i}^{c_i}.$$

5.2 Efficiency

Our second scheme is more efficient than the first one. It does not make use of GDH groups. Thus, results obtained are more general and the length of signatures is shorter. As compared to Bellare-Neven multisignatures [BN06], our multisignature has more than one element. On the other hand, the security reduction of our second scheme is tight under DDH assumption.

5.3 Security

Theorem 5.1 *The proposed multisignature scheme is $(t, q_H, q_S, N, \epsilon)$-unforgeable if the DDH problem is (t', ϵ')-unforgeable in $\mathbb{G}$, where*

$$\epsilon' \geq \epsilon - \frac{(q_{\mathcal{H}} + Nq_S + 1)^2}{2^{l_0}} - \frac{q_S(2q_{\mathcal{H}} + 3Nq_S) + 1}{2^k} \quad \textit{and} \quad t' \leq t + O(q_S t_{exp}).$$

Proof Assume we have a polynomial time forger that runs in time at most t, makes at most q_H hash queries and at most q_S signature queries and outputs a valid multisignature with probability at least ϵ. We need to construct an algorithm $\mathcal{B}$ which, by interacting with the forger $\mathcal{A}$, solves DDH problem with probability ϵ' within time t'.

Algorithm $\mathcal{B}$ given as input a group $\mathbb{G}$ and a tuple (g, h, y_1, y_2), informally, is to determine whether this is a random tuple or a Diffie-Hellman tuple (cf. Section 2.2). Assume that $\mathcal{A}$ is trying to attack the honest signer P^* who have the public keys $PK^* = (y_1, y_2)$. $\mathcal{B}$ sets $PK = (y_1, y_2)$ and runs $\mathcal{A}$ on input PK^*. Algorithm $\mathcal{B}$ simulates the signing and hash oracle for $\mathcal{A}$ as follows:

First, $\mathcal{B}$ initializes two list H, G to simulate random oracles $\mathcal{H}, \mathcal{G}$, respectively. A list T assigns a unique index $1 \leq i \leq q_H + Nq_S$ to each public key PK occurring either as a cosigner's public key in one of $\mathcal{A}$'s signature queries, or as the first item in the argument of one of $\mathcal{A}$'s queries to $\mathcal{G}$. $\mathcal{B}$ uses a counter ctr indicating the current index of this list, initially set to 0 and assigns $\mathsf{T}[PK^*] \leftarrow 0$. It responds to $\mathcal{A}$'s queries at random as follows:

Queries to $\mathcal{H}$. In response to a query $\mathcal{H}(u_i)$ or $\mathcal{H}(v_i)$, $\mathcal{B}$ first checks if the output of $\mathcal{H}$ on this input has been previously defined. If so, $\mathcal{B}$ returns the previously assigned value. Otherwise, $\mathcal{B}$ returns with a value chosen uniformly at random from $\{0,1\}^{l_0}$. All queries u_i, v_i are stored in the list H.

Queries to $\mathcal{G}$. In response to a query $\mathcal{G}$, we first parse the argument of the query into two portion as PK, Q. If $\mathsf{T}[PK]$ is undefined then $\mathcal{B}$ increases ctr and sets $\mathsf{T}[PK] \leftarrow ctr$. If $\mathsf{G}[ctr, Q]$ is undefined, then $\mathcal{B}$ assigns $\mathsf{G}[i, Q]$, for all $1 \leq i \leq q_H + Nq_S$ with random numbers, and picks in advance at random as $e_1, \ldots, e_{q_H+q_S} \in \mathbb{Z}_p$ to assign for $\mathsf{G}[0, Q]$.

Signing query on m with group of users L: First, $\mathcal{B}$ checks whether $P^* \notin L$, if so algorithm $\mathcal{B}$ returns $\perp$ to $\mathcal{A}$. If not, it parses the public keys of signers in L as $Pk = \{PK_1 = PK^*, PK_2, \ldots, PK_n\}$. Then, $\mathcal{B}$ checks whether $\mathsf{T}[PK_i]$, for $i \in \{2, \ldots, n\}$, has already been defined. If not, it increases ctr and sets $\mathsf{T}[PK_i] \leftarrow ctr$. Then, $\mathcal{B}$ sets c_1 at random as $e_1, \ldots, e_{q_H+q_S}$ in advance. $\mathcal{B}$ generates $(\gamma, s_1) \in \mathbb{Z}_p^2$ at random, computes $u_1 = g^{s_1} y_1^{-c_1}$ and $v_1 = h^{s_1} y_2^{-c_1}$. It sets $h_1 = \mathcal{H}(u_1)$, $v_1 = \mathcal{H}(v_1)$ and sends to all signers.

After receiving $h_2, \cdots, h_n$ and $t_2, \cdots, t_n$ from the adversary $\mathcal{A}$, $\mathcal{B}$ looks up in the list H for values u_j, v_j such that $h_i = \mathcal{H}(u_j)$ and $t_i = \mathcal{H}(v_j)$. If multiple such values are found for some i, the algorithm $\mathcal{B}$ stops (**Event 1**). If no such value was found for some i then it sets $alert \leftarrow true$ and sends u_1, v_1 to all cosigners; otherwise, $\mathcal{B}$ computes $u = \prod_{i=1}^{n} u_i$ and $v = \prod_{i=1}^{n} v_i$. Then, $\mathcal{B}$ checks whether $\mathsf{G}[0, Q]$ has already been defined for $Q = \langle u, v, Pk, m, g, h\rangle$. If so, it fails and stops (**Event 2**). If not, it sets $\mathcal{G}(PK_1, u, v, Pk, m, g, h) = \mathsf{G}[0, Q] = c_1$, randomly chooses $\mathsf{G}[i, Q] \stackrel{R}{\leftarrow} \mathbb{Z}_p$ for all $1 \leq i \leq q_H + Nq_S$ and sends u_1, v_1 to all cosigners.

After receiving $u_2, v_2, \ldots, u_n, v_n$ from $\mathcal{A}$, $\mathcal{B}$ verifies that $h_i = \mathcal{H}(u_i)$ and $t_i = \mathcal{H}(v_i)$ for all $1 \leq i \leq n$. If not, it returns $\perp$ to $\mathcal{A}$. If $alert = true$, $\mathcal{B}$ fails and stops (**Event 3**). Else, it sends s_1 to all cosigners.

After receiving $s_2, \cdots, s_n$ from cosigners ($\mathcal{A}$), $\mathcal{B}$ computes $s = \sum_{i=1}^{n} s_i$ and returns the valid signature (u, v, s).

As we can see, this simulator is valid, except for some events:

- **Event 1**: In this case, there exists two values $u_i \neq u_i'$ or $v_i \neq v_i'$ such that $h_i = \mathcal{H}(u_i) = \mathcal{H}(u_i')$ or $t_i = \mathcal{H}(v_i) = \mathcal{H}(v_i')$ for some i, i.e, there is at least one collision occurred in $\mathcal{H}$. As outputs of $\mathcal{H}$ are chosen at random from $\{0,1\}^{l_0}$ and since there are at most $q_{\mathcal{H}} + Nq_S$ queries to $\mathcal{H}$, the probability that at least one collision occurs is upper bounded by $((q_{\mathcal{H}} + Nq_S)(q_{\mathcal{H}} + Nq_S + 1)/2)/2^{l_0} \leq (q_{\mathcal{H}} + Nq_S + 1)^2/2^{l_0+1}$.
- **Event 2**: The algorithm $\mathcal{B}$ only aborts at event 2 if it has run into an input string $\langle 0, u, v, Pk, m, g, h\rangle$ on which $\mathcal{G}$ has been already queried. We distinguish between the case that $\mathcal{H}(u_1)$ and $\mathcal{H}(v_1)$ were previously queried by the forger, and the case that they were not. In the first case, $\mathcal{A}$ probably knows u, v and may have deliberately queried $\mathcal{G}(PK, u, v, Pk, m, g, h)$ for some PK. But since u_1, v_1 was chosen by $\mathcal{B}$ independently from $\mathcal{A}$'s view at the beginning of the signing protocol, the probability that $\mathcal{A}$ queried $\mathcal{H}(u_1)$ and $\mathcal{H}(v_1)$ is at most $(q_{\mathcal{H}} + Nq_S)/p$, for one signature query. In the second case, $\mathcal{A}$'s view is completely independent of u_1 and v_1, and hence of u and v. The probability that u and v occurred by chance in a previous query to $\mathcal{G}$ or was set by $\mathcal{B}$ in one of the $i-1$ previous signature simulations is at most $(q_{\mathcal{H}} + q_S)/p$, for one signature query. For q_S signature queries, the failure probability is thus upper bounded by $q_S((q_{\mathcal{H}} + Nq_S) + (q_{\mathcal{H}} + q_S))/p \leq 2q_S(q_{\mathcal{H}} + Nq_S)/2^k$.
- **Event 3**: $\mathcal{A}$ must have predicted the value of $\mathcal{H}(u_i)$, $\mathcal{H}(v_i)$ for at least one $1 \leq i \leq n$, which it can do with probability at most $N/2^{l_0}$, for one signature query. For q_S signature queries, the failure probability is thus upper bounded by $q_S N/2^{l_0}$.

As a conclusion, except with a failure probability:

$$\epsilon_{stop} = \frac{(q_{\mathcal{H}} + Nq_S + 1)^2}{2^{l_0+1}} + \frac{2q_S(q_{\mathcal{H}} + Nq_S)}{2^k} + \frac{q_S N}{2^{l_0}} \leq \frac{(q_{\mathcal{H}} + Nq_S + 1)^2}{2^{l_0}} + \frac{2q_S(q_{\mathcal{H}} + Nq_S)}{2^k},$$

the simulation is successful.

Eventually, $\mathcal{A}$ halts and outputs an attempted forgery $\sigma = (\hat{u}, \hat{v}, \hat{s})$ on some message $\hat{m}$ along with $L = \{P^*, P_2, \cdots, P_n\}$. It must not previously have requested a signature on $\hat{m}$

with L. In addition, it outputs the private keys $(x_2, \cdots, x_n)$ for all secret keys except the key x of the challenge P^*. Algorithm $\mathcal{B}$ first computes additional random oracle queries $\mathcal{G}_1(PK_i, \hat{u}, \hat{v}, Pk, \hat{m}, g, h)$ for $1 \leq i \leq n$, thereby making sure that $\mathsf{G}[PK_i]$ is defined. If $\mathcal{A}$'s forgery is valid, i.e. g, h, y_1, y_2 is a Diffie-Hellman tuple, the simulator outputs 1 with the probability $\epsilon - \epsilon_{stop}$; otherwise it outputs 0.

On the other hand, if (g, h, y_1, y_2) is a random tuple, then it is not a Diffie-Hellman tuple with probability $1 - 1/p$. In this case, for any u, v and any query $\mathcal{G}(PK_1, u, v, Pk, m, g, h)$ made by $\mathcal{A}$ then there is at most one possible value of c for which there exists an s satisfying $u = g^s y_1^c$ and $v = h^s y_2^c$ (*lemma* 1 in [GJKW07]). Thus, $\mathcal{A}$ outputs a forgery (and hence $\mathcal{B}$ outputs 1) with probability at most $1/p + q_{\mathcal{G}}/2^k \leq (q_{\mathcal{G}} + 1)/2^k$. (As in the previous proof, the additive factor of 1 occurs in case $\mathcal{A}$ did not make the relevant query to $\mathcal{G}$ for its forgery.)

Summing the probabilities, we see that:

$$\begin{aligned} |Pr\left[\mathcal{B}(g, g^x, g^y, g^{xy}) = 1\right] - Pr\left[\mathcal{B}(g, g^x, g^y, g^z) = 1\right]| &\geq \epsilon - \epsilon_{stop} - (q_{\mathcal{G}} + 1)/2^k \\ &\geq \epsilon - (q_{\mathcal{H}} + Nq_S + 1)^2/2^{l_0} - 2q_S(q_{\mathcal{H}} + Nq_S)/2^k - (q_{\mathcal{G}} + 1)/2^k \\ &\geq \epsilon - (q_{\mathcal{H}} + Nq_S + 1)^2/2^{l_0} - (q_S(2q_{\mathcal{H}} + 3Nq_S) + 1)/2^k \end{aligned}$$

and the running time t' satisfies $t' \leq t + O(q_S t_{exp})$, where t_{exp} is the time of an exponentiation in $\mathbb{G}$.

6 Conclusion

At CCS'06, Bellare and Neven introduced the first secure multisignature scheme in the plain public key model. In this paper, we presented two multisignature schemes provably secure in the random oracle model. We proved the security of our schemes by reducing it to Diffie-Hellman problems with tight security reductions. Further, our schemes are secure against rogue-key attacks in the plain public key model.

References

[BKLS02] P. S. L. M. Barreto, H. Y. Kim, B. Lynn, and M. Scott. Efficient algorithms for pairing-based cryptosystems. In *CRYPTO*, 354–368, 1992.

[BN06] M. Bellare and G. Neven. Multi-signatures in the plain public-key model and a general forking lemma. In *ACM CCS*, 2006.

[Bol04] A. Boldyreva. Efficient threshold signature, Multisignature and Blind signature schemes based on the Gap-Diffie-Hellman-group signature scheme. In *PKC*, 2003.

[Boy89] C. Boyd. Digital multisignatures. In *Cryptography and Coding*, pages 241–246. Oxford University Press, 1989.

[BR93] M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *ACM CCS*, 1993.

[CEG87] D. Chaum and J.-H. Evertse and J. Van de Graaf. An improved protocol for demonstrating possession of discrete logarithms and some generalizations. In *EUROCRYPT*, pages 127–141, 1987.

[CM05] B. Chevallier-Mames. An Efficient CDH-Based Signature Scheme with a Tight Security Reduction. In *CRYPTO*, pages 511–526, 2005.

[CP92] D. Chaum and T. P. Pedersen. Wallet Databases with Observers. In *CRYPTO '92*, pages 89–105, 1992.

[GJ03] E-J. Goh and S. Jarecki. A signature scheme as secure as the diffie-hellman problem. In *EUROCRYPT*, pages 401–415, 2003.

[GJKW07] E-J. Goh, S. Jarecki, J. Katz, and Nan Wang. Efficient signature schemes with tight security reductions to the diffie-hellman problems. *Journal of Cryptology*, 20(4):493–514, 2007.

[GPS06] S. Galbraith, K. Paterson, and N. Smart. Pairings for cryptographers, 2006.

[IN83] K. Itakura and K. Nakamura. A public key cryptosystem suitable for digital multisignatures. *NEC Research and Development*, 71:1–8, 1983.

[JN03] A. Joux and K. Nguyen. Separating Decision Diffie-Hellman from Computational Diffie-Hellman in cryptographic groups. *J. Cryptology*, 239–247, 2003.

[JS99] M. Jakobsson and C-P. Schnorr. Efficient Oblivious Proofs of Correct Exponentiation. In *IFIP TC6/TC11 Joint Working CMS '99*, pages 71–86.

[KW03] J. Katz and N. Wang. Efficiency improvements for signature schemes with tight security reductions. In *ACM CCS*, pages 155–164, 2003.

[LOS+06] S. Lu, R. Ostrovsky, A. Sahai, H. Shacham, and B. Waters. Sequential aggregate signatures and multisignatures without random oracles. In *EUROCRYPT*, pages 465–485, 2006.

[MOR01] S. Micali, K. Ohta, and L. Reyzin. Accountable-subgroup multisignatures. In *ACM CCS '01*, pages 245–254, 2001.

[MR02] S. Micali and L. Reyzin. Improving the exact security of digital signature schemes. *J. Cryptology*, 15(1):1–18, 2002.

[Oka88] T. Okamoto. A digital multisignature scheme using bijective public-key cryptosystems. *ACM Trans. Comput. Syst.*, 6(4):432–441, 1988.

[PS96] D. Pointcheval and J. Stern. Security proofs for signature schemes. In *EUROCRYPT*, pages 387–398, 1996.

[Sch91] C-P. Schnorr. Efficient signature generation by smart cards. *J. Cryptology*, 4(3):161–174, 1991.

[SV07] N. Smart and F. Vercauteren. On computable isomorphisms in efficient asymmetric pairing based systems. *Discrete Applied Mathematics*, 155:538–547, April 2007.