DDH-based Multisignatures with Public Key Aggregation

Duc-Phong Le[1] Guomin Yang[2] Ali Ghorbani[1]

[1]Canadian Institute for Cybersecurity
University of New Brunswick
Frederiction, New Brunswick

[2]School of Computing and
Information Technology
University of Wollongong
Wollongong, Australia

July 2, 2019

Abstract

A multisignature scheme allows a group of signers to produce a joint signature on a common message, which is more compact than a collection of distinct signatures from all signers. Given this signature and the list of signers' public keys, a verifier is able to check if every signer in the group participated in signing. Recently, a multisignature scheme with public key aggregation has drawn a lot of attention due to their applications into the blockchain technology. Such multisignatures provide not only a compact signature, but also a compact aggregated public key, that is both the signature size and the public key size used to verify the correctness of the signature are independent from the number of signers. This is useful for a blockchain because of its duplication over a distributed network, and thus it is required to be as compact as possible. In this paper, we introduce a new multisignature scheme with such a feature. Our scheme is proven secure under the Decisional Diffie-Hellman assumption. In addition, in the presence of rogue key attacks, the security of our scheme is proven in the plain public key model.

Multisignatures, Public key Aggregation, Decisional Diffie-Hellman problem, Blockchain Technology.

1 Introduction

Multisignatures, introduced by Itakura and Nakamura in [11], allow a group of signers to jointly generate a signature σ on a common message m such that a verifier is convinced that each member of the group participated in signing. Such multisignatures offer advantages over standard signatures if their size is independent from the number of signers. There are a range of applications, in which we need to provide efficient batch verification of several signatures of the same message under different public keys, such as applications concerning the multi-cast (or one-to-many) communication: IP Multi-cast, Peer-to-Peer file sharing, mobile ad-hoc networks, etc.

Recently, it is shown that multisignatures have applications in transactions on blockchain (see a discussion in [17]). For instance, a multisignature in Bitcoin refers to requiring more than one key to authorize a Bitcoin transaction. While this increases the security of transaction, the storage overhead will be bigger as all public keys are stored on the blockchain. Thus, a multisignature scheme whose size of public keys is that of a single standard signature has particular interests in the blockchain technology.

Multisignatures with public key aggregation. Blockchain could be considered as a distributed ledger that will be duplicated in every node on the network. Every single transaction happened on this network will be signed and recorded in the blockchain. This data structure provides the integrity and authenticity of transactions without trust from a third party. Because of the duplication on each node, it is required that information recorded on the blockchain should be as compact as possible. For transactions involving multi-parties, multisignatures are required. The public keys of all co-signers also should be recorded on the blockchain for the purpose of later verification. Multisignatures with small public keys thus became an interesting and challenging research topic.

In [17], Maxell et al. introduced an interesting technique that allows us to aggregate the public keys of all involved signers into an aggregated public key. Their scheme is based on Schnorr's signature scheme [22] and the verification protocol is as simple as the underlying signature scheme. Basically, their scheme is an interactive protocol, which consists of 3 rounds of communication among signers and the security is proved under the discrete logarithm assumption. Subsequently, Boneh et al. [8] applied this aggregation technique and presented a non-interactive multi-signature scheme by using the pairing-based cryptography. Their scheme is based on the short BLS signatures [9], whose security is based on the Gap-Diffie-Hellman assumption. The non-interactive property is achieved due to using a bilinear map, a special cryptographic operation that could be constructed from pairing-friendly elliptic curves [10]. Both these above schemes are proven secure against *rogue-key attacks* in the *plain public key model.*

Rogue-key Attacks. Multisignature schemes have to be secure against the *rogue key* attack, which is related to some weaknesses in the key setup protocol. In such an attack, 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 users, allowing him to produce forgeries easily. Micali et al [18] introduced the first security model to prevent such an attack. This model basically requires the adversary to essentially prove a knowledge of secret key (KOSK) for every public key he/she chooses. They implemented the KOSK via an interactive pre-processing protocol involving all potential signers. This makes their scheme *impractical.* Another way to realize the KOSK assumption is to employ so-called Key Registration Model (KR) for Public Key Infrastructure (PKI), introduced in the context of multisignatures by Ristenpart and Yilek [21]. In the KR model, a Certification Authority (CA) can certify a public key only if its owner passes a special key registration procedure, called a *proof of possession* of the secret key (POP). The KR model thus shifts the proof verification overhead from multisignature verifiers to the CA's. This imposes a limitation on the use of those multisignatures. Then, Bagherzandi and Jarecki [3] removed this limitation by considering an alternative mode of PKI operation which we call the Key Verification (KV) Model. In the KV model each private key owner also produces a (POP) string, but instead of handing it to the CA during the key registration process she attaches it to her key (or a PKI certificate on the key). This POP message is then verified by a multisignature receiver instead of by the CA, for example together with verification of PKI certificates on that key [3].

Till date, the most practical model, so-called *plain public key model*, was proposed by Bellare and Neven in [5]. In this model, the set of potential signers is dynamic and these signers can choose their public keys as they wish and may register keys at any time.

Our contribution. In this paper, we propose a new multisignature scheme with public key aggregation. We will show that our scheme is secure under the Decisional Diffie-Hellman Assumption against rogue key attacks. Moreover, our scheme is proven secure under the plain public key model, that is, the signers can choose their public keys as they wish and may register keys at any time. To the best of our knowledge, this is the first multisignature scheme supporting aggregated public keys and proved secure under a classical Diffie-Hellman assumption.

Organization. The rest of the paper is organized as follows. Section 2 provides some preliminaries about the Decisional Diffie-Hellman (DDH) problem, multisignatures and its security model, and the generalized forking lemma. Section 3 presents our construction based on DDH problem and analyzes its security. We provide some discussions and a comparison to other schemes in Section 4. Finally, we conclude the paper in Section 5.

2 Preliminary

We first recall notations and conventions used in the rest of the paper. If $\mathcal{A}$ is a randomized algorithm, then $\mathcal{A}(x_1, \ldots; R)$ denotes its output on the given inputs $x_1, \ldots$, and coins R, and $y \overset{\$}{\leftarrow} \mathcal{A}(x_1, \ldots)$ means that y is assigned the output of $\mathcal{A}(x_1, \ldots; R)$ with a chosen random R. If S is a (multi)set, then $\mid S \mid$ denotes its cardinality, i.e., the number of elements of S. We also denote $s \overset{\$}{\leftarrow} S$ a random selection of an element s in the set S.

2.1 The Decisional Diffie-Hellman Problem

Let $\mathbb{G}$ be a cyclic group of prime order p and let g be a generator of $\mathbb{G}$. The security of our scheme is based on the hardness of the following computational problem.

Definition 1 *(***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****) and tuples of the form (g^a, g^b, g^c), where $a, b, c \overset{R}{\leftarrow} \mathbb{Z}_q^*$. A distinguishing algorithm $\mathcal{A}$ has an advantage ϵ in solving the* DDH *problem in $\mathbb{G}$ if*

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

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 $\mathbb{G}$.*

Hardness of the DDH problem in $\mathbb{G}$ implies hardness of the discrete logarithm problem in $\mathbb{G}$. The inverse of these statements is not believed to be true in general. There are groups (e.g., gap Diffie-Hellman (GDH) groups [12]) in which the DDH problem is known to be easy, yet the discrete logarithm problem is still believed to be hard. However, surveys in [7, 16] showed that for a variety of groups of interest "the best known algorithm for DDH is a full discrete log algorithm".

2.2 Multisignature and its Security

Formally, a multisignature scheme with key aggregation consists of five algorithms MS = (Setup, KGen, KAgg, MSign, Vf) where MSign is distributed algorithm. Compared to a classical multisignature scheme, a scheme with key aggregation has one more algorithm KAgg that aggregates co-signers' public key.

- params $\rightarrow$ Setup(1^k). A central authority, on input the security parameter k, runs the algorithm Setup to produces the global information params. Algorithm Setup is probabilistic.
- $(sk, pk) \leftarrow$ KGen, executed by each signer on input params, generates this signer's secret key sk, the corresponding public key pk. Algorithm KGen is probabilistic.
- The KAgg algorithm on input a set of public keys outputs a single aggregate public key apk.

- The *multi-signing algorithm* MSign might be a probabilistic algorithm which, given a message m, the global information params and a list of signers L along with their public and secret keys, produces a multisignature σ. The multi-signing can be interactive or non-interactive.
- $\{0,1\} \leftarrow \mathsf{Vf}(\mathsf{params}, m, L, \sigma)$ verifies whether σ is a valid multisignature on the message m with respect to L. This algorithm is deterministic.

Table 1: Chosen Message Attack against Multisignature in the Plain public-key model

Experiment $\mathbf{Exp}^{\mathsf{uu-cma}}_{\mathsf{MS}}(\mathcal{A})$:

- $\mathsf{params} \leftarrow \mathsf{Setup}(1^k)$; $(sk^*, pk^*) \leftarrow \mathsf{KGen}(\mathsf{params})$; $List \leftarrow \emptyset$;

- Run $\mathcal{A}(\mathsf{params}, pk^*)$, and for every signature query m made by $\mathcal{A}$ do the following:

1. $List \leftarrow List \cup \{(m, L)\}$, where L is the list of users participating in signing the message m;

2. Execute protocol MSign on behalf of an honest player on inputs $(\mathsf{params}, m, sk^*, L)$, forwarding messages to and from $\mathcal{A}$.

- When $\mathcal{A}$ halt; parse its outputs as (m, L, σ).

- If $(m, L) \notin List$, $pk_1 = pk^*$ and $\mathsf{Vrfy}(\mathsf{params}, m, L, \sigma) = 1$

then return 1. Otherwise return 0.

Security. A multisignature scheme should satisfy unforgeability. The attacks of an adversary $\mathcal{A}$ against multisignature schemes are to forge a group of signers L and a multisignature of some message such that the latter is accepted by a verifier whereas some signers of the group L did not sign the message. Table 1 shows how an attack against a multisignature scheme is performed. In principle, we give the adversary the power to request the private key on all but one signer and its goal is to frame this honest signer. The adversary can choose their public keys arbitrarily, even as a function of the public key of the honest signer. The adversary $\mathcal{A}$ is given the global information params, a challenging public key pk^* corresponding to the honest signer and signing and hash oracles. His goal is to output a forged message-group-multisignature tuple (m, L, σ), such that the honest signer, who did not complete the multisignature generation protocol on the input message m, is in L and $\mathsf{MS.Vf}(\mathsf{params}, m, L, \sigma) = 1$.

Let $\mathcal{A}$ be an adversary against the multisignature scheme, which consists of four algorithms Setup, KGen, MSign, and Vf. As in the previous works on multisignatures, e.g. [3, 18], we define multisignature security as Universal Unforgeability (UU) under a Chosen Message Attack (CMA) against a single honest player. Namely, we define $Adv_{\mathsf{MS}}^{\mathsf{uu-cma}}(\mathcal{A})$ to be the probability that experiment $\mathbf{Exp}_{\mathsf{MS}}^{\mathsf{uu-cma}}(\mathcal{A})$ described in Table 1 outputs 1. A multisignature scheme is said to be $(t, q_S, q_H, N, \epsilon)$-secure in the random oracle model if $Adv_{\mathsf{MS}}^{\mathsf{uu-cma}}(\mathcal{A}) \leq \epsilon$ for every adversary $\mathcal{A}$ that runs in time at most t, makes at most q_S signing queries with the honest signer, at most q_H random oracle queries, and the number of signers in L involved in any signing query or in the forgery is at most N.

2.3 Generalized Forking Lemma

The forking lemma, introduced by Pointcheval and Stern [20], is commonly used to prove the security of Schnorr-like signature schemes [22] in the random oracle model. The lemma was then generalized to a wider class of schemes in [5] [2]. We recall the generalized version introduced by Bagherzandi et al [2].

Let A be an algorithm, on input *input*, interacts with a random oracle $H : \{0,1\}^* \to \mathbb{Z}_q$. Let $f = (\rho, h_1, \dots, h_{q_h})$ be randomness involved in an execution of $\mathcal{A}$, where ρ is $\mathcal{A}$'s random input, h_j is the j-th response of H, and q_h is the maximum number of hash queries $\mathcal{A}$ makes. Let Ω be the space of all vectors f. Let $f_j = (\rho, h_1, \dots, h_{j-1})$, e.g., $f_1 = \rho$, $f_2 = (\rho, h_1)$, etc. We consider an execution of $\mathcal{A}$ on input *input* and randomness f, denoted $\mathcal{A}(input, f)$ as adversary's *success* if it outputs a pair $(J, \{\mathsf{out}_j\}_{j \in J})$, where J is a multi-set that is a (non-empty) subset of $\{1, \dots, q_h\}$. We assume that if $\mathcal{A}$ fails then it outputs $J = \emptyset$. For an given input *input*, the generalized forking lemma is defined as in Algorithm $\mathsf{GF}_{\mathcal{A}}$.

Lemma 2 *(Generalized Forking Lemma [2]) Let* IG *be a randomized algorithm that generates par and* $\mathcal{A}$ *be a randomized algorithm making at most* q_H *hash queries s.t.* $\mathcal{A}(input)$ *succeeds (i.e. outputs* $(J, \{\mathsf{out}_j\}_{j \in J})$ *s.t.* $\mid J \mid = n$*) with probability* ϵ*, where the probability goes over input* $\overset{\$}{\leftarrow} \mathsf{IG}$ *and* $f \overset{\$}{\leftarrow} \Omega$*. If* $q > 8nq_H/\epsilon$*, then algorithm* $\mathsf{GF}_{\mathcal{A}}(input)$ *runs in time at most* $\tau \cdot 8n^2 q_H/\epsilon \cdot ln(8n/\epsilon)$ *has a forking success with probability at least* $\epsilon/8$*, where the probability goes over coins of* IG *and* $\mathsf{GF}_{\mathcal{A}}$*.*

3 DDH-based Multisignature Scheme with Public Key Aggregation

In this section, we describe an interactive multisignature scheme which relies on Decisional Diffie-Hellman problem. As our construction is based on Katz-Wang signature scheme [13], we recall it first.

Let $\mathbb{G}$ be a cyclic group of prime order q, g be a generator of $\mathbb{G}$, $h \overset{\$}{\leftarrow} \mathbb{G}$ chosen randomly and let $H : \{0,1\}^* \to \mathbb{Z}_q$ be a hash function. A private/public key pairs is (x, pk), where $x \overset{\$}{\leftarrow} \mathbb{Z}_q$ and $pk = (y_1, y_2) = (g^x, h^x)$. To sign a message m, the signer chooses a random $r \overset{\$}{\leftarrow} \mathbb{Z}_q$ and computes $u = g^r$, $v = h^r$,

Table 2: Generalized Forking Lemma

```
Algorithm $\mathsf{GF}_{\mathcal{A}}(input)$
    pick random coins $\rho$ for $\mathcal{A}$, and $h_1, \ldots, h_{q_h} \xleftarrow{\$} \{0,1\}^\ell$
    $f = (\rho, h_1, \ldots, h_{q_h}) \xleftarrow{\$} \Omega$
    $(J, \{\mathsf{out}_j\}_{j \in J}) \leftarrow \mathcal{A}(input, f)$
    **If** $J = \emptyset$ **then return** *fail*
    Let $J = \{j_1, \ldots, j_n\}$ s.t. $j_1 \leq \ldots \leq j_n$, $X = \{(h_j, \mathsf{out_j}_{j \in J})$ and $X' = \{\}$
    **For** $i = 1$ **to** $n$ **do**
        Set $\mathsf{succ_i} = 0$, $k_i = 0$ *and* $k_{max} = 8nq_h/\epsilon \cdot ln(8n/\epsilon)$
        Repeat until $\mathsf{succ_i} = 1$, or $k_i > k_{max}$
            $f' \xleftarrow{\$} \Omega$ s.t. $f'_{j_i} = f_{j_i}$
            Let $f' = (\rho, h_1, \ldots, \ldots, h_{j_{i-1}}, h'_{j_i}, \ldots, h'_{q_h})$
            $(J', \{\mathsf{out}'_j\}_{j \in J'}) \leftarrow \mathcal{A}(input, f')$
            **If** $h'_{j_i} \neq h_{j_i}$, $J' \neq \emptyset$ *and* $j_i \in J'$ **then**
                add $(h'_{j_i}, \mathsf{out}'_{j_i})$ to $X'$ *and* set $\mathsf{succ_i} = 1$
    **If** for all $i = 1$ to $n$ $\mathsf{succ_i} = 1$ **then return** $(X, X')$
    **Else return** *fail*
```

$c = H(pk, u, v, m)$, and $s = r + cx$. The signature is the pair (c, s), and it can be verified by checking if $c = H(pk, u', v', m)$, where $u' = g^s y_1^{-c}$ and $v' = h^s y_2^{-c}$.

This signature scheme was then modified and extended to a multisignature scheme in plain public key model in [14]. Their signature consists of three elements (u, v, s), where u, v are group elements instead of two elements (c, s). Readers could refer to [14, Section 6] for more details.

3.1 Our Multisignature Scheme

Let $\mathbb{G}$ be a cyclic group of prime order q and let H_{com}, H_{agg}, and H_{sig} be three cryptographic hash functions from $\{0,1\}^*$ to $\mathbb{Z}_q$. Our multisignature scheme is defined as follows:

Parameter generation. A trusted center runs the algorithm Setup that chooses a generator $g \in \mathbb{G}^*$ and $h \xleftarrow{\$} \mathbb{G}$ at random. It then publishes $params = (\mathbb{G}, g, h, H_{com}, H_{agg}, H_{sig})$ as system wide parameters.

Key generation. The algorithm KGen, executed on each signer, picks a random number $x_i \xleftarrow{\$} \mathbb{Z}_q$ as a private key. The corresponding public keys are $pk_i = (y_i, z_i)(= (g^{x_i}, h^{x_i}))$.

Key Aggregation. On input $L = \{pk_1, \ldots, pk_n\}$, the key aggregation algorithm

KAgg computes:

$$apk_1 \leftarrow \prod_{i=1}^{n} y_i^{H_{agg}(pk_i,L)}$$

and

$$apk_2 \leftarrow \prod_{i=1}^{n} z_i^{H_{agg}(pk_i,L)}.$$

Signing. Suppose that a group of n signers wishes to sign a common message m, each has as input its own public and private key as well as a multiset of public keys $L = \{pk_1, \ldots, pk_n\}$ of the other signers. The signing algorithm MSign, which is *interactive*, consists of three rounds, where in each round signers send (and receive) a message to (from resp.) each other signer.

- **Round 1.** Each signer in the group:
  - picks a random number $r_i \overset{\$}{\leftarrow} \mathbb{Z}_q$;
  - computes its individual commitments $u_i = g^{r_i}$ and $v_i = h^{r_i}$, then queries H_{com} to compute challenges $h_i = H_{com}(u_i)$ and $t_i = H_{com}(v_i)$;
  - sends h_i, t_i to every other signer.
- **Round 2.** Each signer in the group:
  - 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 in the group:
  - receives u_j, v_j from signer j, for $1 \leq j \leq n, j \neq i$;
  - checks whether $h_j = H_{com}(u_j)$ and $t_j = H_{com}(v_j)$ 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 $v = \prod_{i=1}^{n} v_i$.
  - computes $a_i = H_{agg}(pk_i, L)$ and apk_1, apk_2 as in Algorithm KAgg;
  - queries $c = H_{sig}(apk_1, apk_2, u, v, L, m)$, then computes $c_i = a_i c$, and $s_i = r_i + x_i c_i \bmod q$.
  - sends to signer j: s_i.

After receiving s_j from signer j, each signer in the group:

- computes $s = \sum_{i=1}^{n} s_i \bmod q$;
- outputs the signature $\sigma = (c, s)$;

Verification. Given the valid signature σ, list of group of signers L and message m, the verification algorithm Vf computes $u' = g^s \cdot apk_1^{-c}$, $v' = h^s \cdot apk_2^{-c}$ and check whether $c = H_{sig}(apk_1, apk_2, u', v', m)$.

Correctness. It is not hard to see that the scheme is correct. If every signer produces his partial signature honestly, then the multi-signature $\sigma = (c, s)$, where $s = \sum_{i=1}^{n} s_i \bmod q$, and $c = H_{sig}(apk_1, apk_2, u, v, m)$. The verification algorithm computes:

$u' = g^s \cdot apk_1^{-c} = \prod_{i=1}^{n} g^{s_i} (y_i^{H_{agg}(pk_i, L)})^{-c} = \prod_{i=1}^{n} g^{s_i} g^{-x_i a_i c} = \prod_{i=1}^{n} g^{s_i} g^{-x_i c_i} = \prod_{i=1}^{n} g^{r_i} = u$

and,

$v' = h^s \cdot apk_2^{-c} = \prod_{i=1}^{n} h^{s_i} (z_i^{H_{agg}(pk_i, L)})^{-c} = \prod_{i=1}^{n} h^{s_i} h^{-x_i a_i c} = \prod_{i=1}^{n} h^{s_i} h^{-x_i c_i} = \prod_{i=1}^{n} h^{r_i} = v.$

It is hence indeed the case that $c = H_{sig}(apk_1, apk_2, u', v', m)$.

3.2 Security

Theorem 3 *The proposed multisignature scheme is (t, q_H, q_S, ϵ)-unforgeable in the random oracle model if $q > 8q_H/\epsilon$ and if the* DDH *problem is (t', ϵ')-unforgeable in $\mathbb{G}$, where*

$$\epsilon' \geq \epsilon/(8q_H)$$

and

$$t' \leq t + (q_H + q_S + 1)t_{exp}.$$

Assume we have a polynomial time forger $\mathcal{F}$ 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. Assume that an adversary is trying to attack the honest signer P^* who have the public keys $pk^* = (y^*, z^*)$.

We consider an algorithm $\mathcal{A}$, whose aim is informally to determine whether a tuple (h, y^*, z^*) is a random tuple or a Diffie-Hellman tuple. To do so, the algorithm $\mathcal{A}$ on inputs $pk^* = (y^*, z^*)$ and $(\mathbb{G}, g, h)$ proceeds as follows.

First, the algorithm $\mathcal{A}$ maintains initially empty associative arrays $\mathsf{H}_{com}[\cdot]$, $\mathsf{H}_{agg}[\cdot]$ and $\mathsf{H}_{sig}[\cdot]$ which are used to simulate random oracles H_{com}, H_{agg} and H_{sig}, respectively.

- **Queries to H_{com} and H_{sig}.** In response to a query $H_{com}(u_i)$ or $H_{com}(v_i)$ and H_{sig}, $\mathcal{A}$ first checks if the output of H_{com} on this input has been previously defined. If so, $\mathcal{A}$ returns the previously assigned value. Otherwise, $\mathcal{A}$ returns with a value chosen uniformly at random from $\{0,1\}^{l_0}$. All queries u_i, v_i are stored in the array $\mathsf{H}_{com}[\cdot]$, $\mathsf{H}_{sig}[\cdot]$.

- **Queries to H_{agg}.** In response to a query H_{agg}, we distinguish three types:

1. A query on (pk, L) with $pk \in L$ and $pk^* \in L$, and this is the first such a query with L.
2. A query on (pk, L) with $pk \in L$ and $pk^* \in L$, and a prior query of this form with L has been made.
3. Queries of any other form.

Algorithm $\mathcal{A}$ responds the i-th query of type (1) by choosing a random value for $H_{agg}(pk_i, L)$ for every $pk_i \neq pk^* \in L$. It fixes $H_{agg}(pk^*)$ to h_0, an returns the chosen value. All queries of type (1) are stored in the array $\mathsf{H}_{\mathsf{agg}}[\cdot]$. In response to a type (2) query, $\mathcal{A}$ returns the previously assigned value when type (1) query was made. Finally, in response to a type (3) query, Algorithm $\mathcal{A}$ simply chooses and returns a random value in $\mathbb{Z}_q$.

- **Signing query on m with L.** When $\mathcal{F}$ makes a signing query on message m, with the list of signers L, $\mathcal{A}$ computes $(apk_1, apk_2) \leftarrow \mathsf{KAgg}(params, L)$. Then, a signing query to the honest signer P^* consists of three rounds. First, the forger $\mathcal{F}$ provides m, L to P^* and receives the individual challenge h^*, t^* from P^* in response. Second, playing the role of rest signer, the forger $\mathcal{F}$ provides the challenges h_i, t_i to P^* and receives u^*, v^* from P^* in response. Third, $\mathcal{F}$ provides the commitments u_i, v_i to P^* and receives s^* from P^* in response. As stated above, in the simulation, it is not the adversary providing the joint commitment u, v to simulator, we do not thus need to use rewinding. In detail, answering signature queries works as follows:

  First, Algorithm $\mathcal{A}$ checks if $pk* \notin L$, then it returns $\perp$ to $\mathcal{F}$. Otherwise, it parses the public keys of signers $L = \{(y_1, z_1) = (y^*, z^*), (y_2, z_2), ldots, (y_n, z_n)\}$. $\mathcal{A}$ then chooses at random $c_1, s_1 \xleftarrow{\$} \mathbb{Z}_q$ and computes $u_1 \leftarrow g^{s_1} y_1^{-h_0 c_1}$, $v_1 \leftarrow h^{s_1} z_1^{-h_0 c_1}$, $h_1 \leftarrow H_{com}(u_1)$, $t_1 \leftarrow H_{com}(v_1)$, and then sends h_1, t_1 to all signers.

  After receiving $(h_2, t_2), \cdots, (h_n, t_n)$ from the adversary $\mathcal{F}$, $\mathcal{A}$ looks up in the array $\mathsf{H}_{com}[\cdot]$ for values u_j, v_j such that $\mathsf{H}_{com}(u_j) = h_j$ and $\mathsf{H}_{com}(v_j) = t_i$. If multiple such values are found for some i, the algorithm $\mathcal{A}$ stops the execution of $\mathcal{F}$ and outputs 0. 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, $\mathcal{A}$ computes $u \leftarrow \prod_{i=1}^{n} u_i$ and $v \leftarrow \prod_{i=1}^{n} v_i$, $c \leftarrow H_{sig}(apk_1, apk_2, u, v, L, m)$ and sets $\mathsf{H}_{sig}[apk_1, apk_2, u, v, L, m] \leftarrow c$ or it aborts the execution and outputs 0 if this entry in $\mathsf{H}_{sig}[\cdot]$ was already defined. $\mathcal{A}$ sends u_1, v_1 to all other cosigners.

  After receiving $(u_2, v_2), \ldots, (u_n, v_n)$ from $\mathcal{F}$, $\mathcal{A}$ verifies that $h_i = H_{com}(u_i)$ and $t_i = H_{com}(v_i)$ for all $1 \leq i \leq n$. If one of these tests fails, $\mathcal{A}$ halts this signing protocol and returns $\perp$ to $\mathcal{F}$. If $alert = true$, $\mathcal{A}$ aborts the execution of $\mathcal{F}$ and outputs 0. Otherwise, it sends s_1 to all cosigners.

After receiving $s_2, \ldots, s_n$ from $\mathcal{F}$, $\mathcal{A}$ computes $s = \sum_{i=1}^{n} s_i \bmod q$ and returns the valid signature (c, s).

Eventually, $\mathcal{F}$ halts and outputs an attempted forgery $\sigma = (c, s)$ on some message m along with $L = \{pk^*, pk_2, \cdots, pk_n\}$. It must not previously have requested a signature on m with L. When $\mathcal{F}$ fails to output a successful forgery, then $\mathcal{A}$ outputs $(0, \perp)$. Otherwise, $\mathcal{F}$ outputs a forgery (c, s, L, m) such that $\mathsf{Vf}(L, m, c, s) = 1$, $pk^* \in L$ and $\mathcal{F}$ never queried (L, m) to the signing oracle.

Let j_f be the index such that $H_{agg}(pk^*, L) = h_{j_f}$, let $(apk_1, apk_2) \leftarrow \mathsf{KAgg}(params, L)$ and let $a_j \leftarrow H_{agg}(pk_j, L)$ for $L = \{pk_1, \ldots, pk_n\}$. Then, $\mathcal{A}$ outputs $(J = j_f, \{(c, s, L, apk_1, apk_2, a_1, a_2, \ldots, a_n)\})$.

The running time o $\mathcal{A}$ is that of $\mathcal{F}$ plus the additional computation $\mathcal{A}$ makes. Let q_H be the total hash queries to all three hash functions H_{com}, H_{agg}, and H_{sig}. Let q_S denote the signing queries. $\mathcal{A}$ needs two multi-exponentiations in $\mathbb{G}$ to answer H_{com}. We assume that multi-exponentiations take time t_{exp} and all other operations take unit time. Finally, $\mathcal{A}$ spends two multi-exponentiations for verification. $\mathcal{A}$'s running time is therefore at most $t + (q_S + 2)t_{exp} + O(q_H + Nq_S)$, where N is the maximum number of cosigners.

The successful probability of $\mathcal{A}$ is the probability that $\mathcal{F}$ succeeds, except the failure probability. This success probability is $\epsilon_{\mathcal{A}} = \epsilon - \frac{(q_H + Nq_S + 1)^2}{2^{l_0}} - \frac{2q_S(q_H + Nq_S) + 1}{q}$ (the calculation is similar to [14, Section 6]).

We now use the algorithm $\mathcal{A}$ to construct an algorithm $\mathcal{B}$ that on input a DDH instance (h, y^*, z^*) and a forger $\mathcal{F}$, runs the generalized forking algorithm $\mathsf{GF}_{\mathcal{A}}$ as defined in Section 2.3. If $\mathsf{GF}_{\mathcal{A}}$ outputs $(0, \perp)$, then $\mathcal{B}$ outputs fail. If $\mathsf{GF}_{\mathcal{A}}$ outputs $(\{j_f\}, \{out\}, \{out'\})$, then $\mathcal{B}$ proceeds as follows:

- Parse *out* as $(c, s, L, apk_1, apk_2, a_1, a_2, \ldots, a_n)$, where $a_i = H_{agg}(pk_i, L)$.
- Parse *out'* as $(c', s', L', apk'_1, apk'_2, a'_1, a'_2, \ldots, a'_n)$, where $a'_i = H_{agg}(pk_i, L)$.

  From the construction of $\mathsf{GF}_{\mathcal{A}}$, we know that *out* an *out'* were obtained from two executions are identical up to the j_f-th H_{agg} query of type (1). In particular, this means that the arguments of this query are identical, that is, $L = L'$ and $n = n'$. If $H_{agg}(pk^*, L)$ was programmed to two values h_0, h'_0, an by the forking lemma it holds that h_0 and h'_0 are distinct, that is, $h_0 \neq h'_0$.

  Suppose $L = \{pk^* = pk_1, pk_2, \ldots, pk_n\}$. By construction of A, we know that $apk_1 = y_1^{h_0} \prod_{i \geq 2} y_i^{a_i}$, $apk_2 = z_1^{h_0} \prod_{i \geq 2} z_i^{a_i}$, and $apk'_1 = y_1^{h'_0} \prod_{i \geq 2} y_i^{a_i}$, $apk'_2 = z_1^{h'_0} \prod_{i \geq 2} z_i^{a_i}$. Since $\mathcal{A}$ assigned $H_{agg}(pk_i, L) \leftarrow a_j$ for all $i \geq 2$ before the forking point, we have that $a_i = a'_i$ for $i \geq 2$. Therefore, $apk_1/apk'_1 = y_1^{h_0 - h'_0}$ and $apk_2/apk'_2 = z_1^{h_0 - h'_0}$.

If (h, y_1, z_1) is a Diffie Hellman tuple, so $(h^{h_0-h_0'}, y_1^{h_0-h_0'}, z_1^{h_0-h_0'})$ is, and the simulation is perfect.

Using Lemma 2, we know that if $q > 8q_H/\epsilon$, then $\mathcal{B}$ runs in time at most $(t+(q_S+2)t_{exp}+O(q_H+Nq_S))\cdot 8q_H^2/\epsilon \cdot ln(8q_H/\epsilon)$ and succeeds with probability $\epsilon' \geq \epsilon_A/8 = (\epsilon - \frac{(q_H+Nq_S+1)^2}{2^{l_0}} - \frac{2q_S(q_H+Nq_S)+1}{q})/8$.

4 Discussions

Comparing to the multi-signatures in [14], which are also based on Decisional Diffie-Hellman assumption, the proposed multisignature consists of only two elements (c, s) instead of three elements (u, v, s), where u, v are group elements. The proposed multisignature scheme also supports key aggregation that could be applicable to the blockchain technology. On the other hand, the scheme in [14] offered a tight security reduction to the DDH problem as its security proof does not require a forking lemma.

Instead of sending (R, s) as in Schnorr-based multisignature scheme [17], our scheme sends (c, s) as a signature. If we consider the group $\mathbb{G}$ as a multiplicative group of $\mathbb{Z}_q^*$, then $c = H_{sig}(apk_1, apk_2, u', v', m)$ is much shorter. For example, at 128 security level, the size of c is 256 bits, while $|q| = 3072$ as recommended by NIST [4]. On the other hand, if $\mathbb{G}$ is a group of elliptic curve points, then both the two multisignature schemes have roughly the same signature size.

In the round 1 of Algorithm 3.1, each user can compute and exchange one value $h_i = H_{com}(u_i \| v_i)$ instead of two values $h_i = H_{com}(u_i)$ and $t_i = H_{com}(v_i)$. This small change pointed out in [6], would speed up the signature generation process and reduce the bandwidth exchanged on the network as well.

In terms of communication complexity, the non-interactive multisignature scheme in [8] is the most efficient. However, this scheme requires a cryptographic pairing, which is defined over elliptic curves with special properties, such as the embedding degree must be small (i.e., smaller than 50 as pointed out in [10]). Such elliptic curves are called *pairing-friendly* elliptic curves, while our scheme could be implemented on ordinary elliptic curves with arbitrary embedding degrees.

Blockchain, the technology underpinning Bitcoin [19], is a *trusted* and *distributed* ledger, in which users are able to record information, prove and transfer their ownership from one to another without a trusted third party intermediary. All these transactions are *fully traceable*, supervised by parties on the blockchain system and no single party has control over the data. Roughly, the blockchain technology provides *proof-of-existence*, *proof-of-ownership* or *proof-of-chronology* for digital assets. It thus could be applied in many applications such as financial services [1], supply chain [1], digital forensics [15], and many other applications.

As discussed in [17, Section 5], multisignatures with public key aggregation have applications in blockchains, e.g., Bitcoin [19]. Multisignatures allow more

[1] https://www.ibm.com/developerworks/cloud/library/cl-adopting-blockchain-for-enterprise-asset-management-eam/

than one person to be involved into one transaction on the blockchain network. While such an authentication mechanism enhances the security of transactions on blockchain, it makes the blockchain bigger. With an aggregated public key multisignature scheme, the signature as well as the corresponding public keys are as small as the underlying signature scheme, making blockchains smaller.

5 Conclusion

Multisignatures supporting public key aggregation have a great interest in blockchain technology as such signatures could make blockchains smaller. In this paper, we introduced a new multisignature scheme with such a feature. The proposed scheme is the first multisignature scheme proven secure under the Decisional Diffie-Hellman assumption. In addition, the security of our scheme in the presence of rogue key attacks was proven in the plain public key model.

References

[1] ASTRI. Whitepaper On Distributed Ledger Technology. Technical report, Hong Kong Applied Science and Technology Research Institute Company Limited, November 2016.

[2] A. Bagherzandi, J.-H. Cheon, and S. Jarecki. Multisignatures secure under the discrete logarithm assumption and a generalized forking lemma. In *CCS '08: Proceedings of the 15th ACM conference on Computer and communications security*, pages 449–458, New York, NY, USA, 2008. ACM.

[3] A. Bagherzandi and S. Jarecki. Multisignatures using proofs of secret key possession, as secure as the diffie-hellman problem. In *SCN '08: Proceedings of the 6th international conference on Security and Cryptography for Networks*, pages 218–235, Berlin, Heidelberg, 2008. Springer-Verlag.

[4] E. Barker and A. Roginsky. Recommendation for the transitioning of cryptographic algorithms and key lengths. NIST Special Publication, 2015. `https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-131Ar1.pdf`.

[5] M. Bellare and G. Neven. Multi-signatures in the plain public-key model and a general forking lemma. In *CCS '06: Proceedings of the 13th ACM conference on Computer and communications security*, pages 390–399, New York, NY, USA, 2006. ACM Press.

[6] M. Bellare and G. Neven. New multi-signature schemes and a general forking lemma, 2017. http://soc1024.ece.illinois.edu/teaching/ece498ac/fall2018/forkinglemma.pdf.

[7] D. Boneh. The Decision Diffie-Hellman problem. In Joe P. Buhler, editor, *Algorithmic Number Theory*, pages 48–63, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[8] D. Boneh, M. Drijvers, and G. Neven. Compact multi-signatures for smaller blockchains. Cryptology ePrint Archive, Report 2018/483, 2018. `https://eprint.iacr.org/2018/483`.

[9] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. In *ASIACRYPT '01: Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security*, pages 514–532, London, UK, 2001. Springer-Verlag.

[10] D. Freeman, M. Scott, and E. Teske. A taxonomy of pairing-friendly elliptic curves. *J. Cryptol.*, 23(2):224–280, April 2010.

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

[12] A. Joux and K. Nguyen. Separating Decision Diffie-Hellman from Computational Diffie-Hellman in cryptographic groups. *J. Cryptology*, 16(4):239–247, 2003.

[13] J. Katz and N. Wang. Efficiency improvements for signature schemes with tight security reductions. In *CCS 2003: Proceedings of the 10th ACM Conference on Computer and Communications Security*, pages 155–164, 2003.

[14] D.-P. Le, A. Bonnecaze, and A. Gabillon. Multisignatures as secure as the diffie-hellman problem in the plain public-key model. In *Pairing 2009: The third International Conference on Pairing-based Cryptography*, pages 35–51. Springer, August 2009.

[15] D.-P. Le, H. Meng, L. Su, S. L. Yeo, and V. Thing. BIFF: A Blockchain-based IoT Forensics Framework with Identity Privacy. In *TENCON 2018 - 2018 IEEE Region 10 Conference*, pages 2372–2377, Oct 2018.

[16] U. M. Maurer and S. Wolf. The diffie–hellman protocol. *Designs, Codes and Cryptography*, 19(2):147–171, Mar 2000.

[17] G. Maxwell, A. Poelstra, Y. Seurin, and P. Wuille. Simple schnorr multi-signatures with applications to bitcoin. Cryptology ePrint Archive, Report 2018/068, 2018. `https://eprint.iacr.org/2018/068`.

[18] S. Micali, K. Ohta, and L. Reyzin. Accountable-subgroup multisignatures. In *CCS '01: Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 245–254, New York, NY, USA, 2001. ACM Press.

[19] S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System, 2008. https://bitcoin.org/bitcoin.pdf.

[20] D. Pointcheval and J. Stern. Security proofs for signature schemes. In *EUROCRYPT'06 : Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques*, pages 387–398, 1996.

[21] T. Ristenpart and S. Yilek. The power of proofs-of-possession: Securing multiparty signatures against rogue-key attacks. In Moni Naor, editor, *EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 228–245. Springer-Verlag, 2007.

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