修 士 論 文 の 和 文 要 旨

研究科・専攻	大学院　情報理工学研究科　情報・ネットワーク工学専攻　博士前期課程		
氏　　名	竹牟禮　薫	学籍番号	1931080
論 文 題 目	信頼性の高い仮定に基づいた証明可能安全性を持つ軽量な集約署名方式の提案		

要　　旨

集約署名は、複数署名者により生成される異なる文書における個別署名を、小さいサイズの署名に集約可能な暗号技術である。集約署名の概念は Boneh らによって提案された。同時に彼らはペアリングという特殊な代数構造を基にした集約署名方式を提案した。この方式は定数サイズ署名長を達成可能であり、署名者間通信なしに集約可能である。一方で、安全性の基となる計算困難性の仮定は、実用化された多くの暗号技術で用いられる離散対数仮定より強い仮定であり、実用時は少々大きいパラメタを取る必要がある。またペアリング計算は計算コストが高い。このようにペアリングにはいくつかの欠点が存在する。Zhao は初めてのペアリングフリー集約署名方式をビットコイン向けのアプリケーションとして提案した。この方式は、署名長が署名者数に線形に依存するが、軽い計算のみで構成されており、鍵設定に特に仮定を必要しない。しかし、安全性は新しく提案された計算困難性の仮定を基にしている。以上より、ペアリングフリーかつ信頼性の高い仮定に基づく安全性を担保可能な集約署名方式の構築は重要な課題である。

本稿では、主に 3 つの研究成果について述べる。1 つ目は、Zhao 方式に対する任意の文書における偽装を生成可能な準指数時間攻撃者を提案する。準指数時間であるため、理論的には致命的ではないが、実装時のパラメタ設定に影響を与える。具体的には、我々の攻撃者の存在により、当初 Zhao が想定したパラメタより大きいパラメタが必要であることが明らかとなり、これは Zhao 方式の利点を弱める。2 つ目は、新たな枠組みとして事前通信を用いる集約署名を提案し、離散対数仮定を基にした安全性を担保可能な事前通信モデルにおけるペアリングフリー集約署名方式を提案する。署名集約には署名者集約者間の通信が必要であるが、比較的小さい通信コストを達成可能である。一方で、鍵設定では各署名者が正当に鍵を生成したことを証明する必要があり、署名長は署名者数に線形に依存するが、Zhao 方式より小さいサイズを達成できる。また提案方式が Drijvers らの不可能性に抵触しないことの議論も行う。3 つ目は、One-Time 集約方式の提案である。この方式は、一度の鍵生成で一度の集約署名生成が可能な方式である。提案方式の安全性は One-More 離散対数仮定に基づいており、理論的な世界でしか存在しないランダムオラクルを用いずに安全性を証明可能である。署名長は定数サイズを達成可能であるが、信頼できる鍵生成が必要である。

The 2020 Academic Year Master's Thesis

Lightweight Aggregate Signatures with Provable Security under Standard Computational Assumptions

Department of Computer and Network Engineering
Graduate School of Informatics and Engineering
The University of Electro-Communications
Master's Program

Student Number: 1931080
Author: Kaoru Takemure
Advisor: Assoc. Prof. Bagus Santoso
Sub Advisors: Prof. Yasutada Oohama
Sub Advisors: Prof. Tsutomu Kawabata

Submission 25th January 2021

THE UNIVERSITY OF ELECTRO-COMMUNICATIONS
Department of Communications and Systems Engineering

Contents

Chapter 1

Introduction

Aggregate signatures are a cryptographic primitive, in which individual signatures on different messages generated by multiple signers into one compact signature. The notion of this was proposed by Boneh et al., and they proposed a pairing-based aggregate signature scheme [BGLS03]. Other aggregate signature schemes [BNN07, HKW15, Zha19] and several special aggregate signatures, e.g., sequential aggregate signatures [LMRS04, LOS$^+$06, FLS12, LLY13, Nev08, BGR12, BMP16], identity-base aggregate signatures [XZF05, GR06, BGOY07, BJ, HSW13], and synchronized aggregate signatures [GR06, HW18], have been proposed so far.

The aggregate signature scheme [BGLS03] achieves the constant signature size and the non-interactive aggregation because of the property of pairing. These achievements give practical efficiency in terms of storage and communication. In contrast, the security of this scheme is proved under the hardness of the pairing-based Diffie-Hellman problem, which is defined on groups with bilinear maps. The assumption, that this problem is hard to solve in polynomial-time, is stronger than the discrete logarithm (DL)assumption. Moreover, deploying pairing-based aggregate signature schemes in existing applications is expensive because it requires not only replacing the algorithms of a signature scheme with the those of the pairing-based scheme but also replacing an elliptic curve (EC) with pairing friendly ones. In addition, most pairing-based schemes have pairing computations that are still relatively quite costly. Therefore aggregate signatures from general EC groups are more attractive in terms of the computational complexity and the cost of deployment.

Zhao proposed the first pairing-free aggregate signature scheme from general EC [Zha19] for the application to the Bitcoin [Nak08], by extending the Γ-signature [YZ13]. This scheme achieves the security in the plain PK model [BN06], in which all signers generate public keys without proving the validity of those, and the non-interactive aggregation. The signature size depends on the number of signers linearly, but it achieves a smaller size than the concatenation of the Schnorr signature [Sch89]. In contrast, the security

is based on the non-standard hardness assumption, the non-malleable discrete logarithm (NMDL) assumption, which is newly introduced by Zhao in the same paper. This means that the security is not reliable.

Therefore, constructing an aggregate signature scheme with the following properties is a very important open problem from the practical and theoretical points of view: pairing-free, i.e., the scheme does not rely on pairing computations or pairing-based assumption, and provably secure based on well-established standard assumptions, e.g., standard discrete logarithm problem.

1.1 Our Contribution

In this thesis, there are roughly three contributions.

Contribution 1. The first Contribution is that we show a sub-exponential time universal forger under a key-only attack in the knowledge of secret key (KOSK) model [Bol03, LOS+06] against Zhao's aggregate signature scheme. The KOSK model is a model where an adversary is required to output the secret keys corresponding to the cosigners' public keys. Our universal forger can generate a forgery on arbitrary messages without making signing queries. On the other hand, our proposed forger runs in the sub-exponential time because it uses k-sum algorithm [Wag02]. Therefore, it is not fatal theoretically. But it affects the practical performance of Zhao's scheme. Specifically, in order to guarantee the security against our attack, Zhao's scheme requires the bit-length of the order of an underlying group to be approximately $\log n$ times the security parameter where n is the number of signers. This makes Zhao's scheme lose its advantage.

The root of our attack is that the NMDL problem can be solved in sub-exponential time by using k-sum algorithm. This analysis may help to prevent the some error when we analyze the security of newly constructed schemes.

Contribution 2. The second contribution is that we introduce a new type of aggregate signatures, aggregate signatures with pre-communication (PreCom), and a pairing-free aggregate signature with PreCom scheme PCAS based on the well-established hardness assumption by extending the Schnorr signature.

The pre-communication is an interaction protocol executed in the offline stage, not in the signing stage. The offline stage is the phase between generating keys and deciding the messages to be signed. The signing stage is the phase after deciding the messages to be signed. The communication in the signing stage loses the significance of aggregate

signatures because we can use the multi-signatures in such a situation, which allows combining individual signatures on the same message, by sharing messages among signers. The pre-communication of PCAS can still keep the most important feature of aggregate signatures, i.e., any signer is allowed to choose their individual message to be signed without interacting with other signers. Also the communication of PCAS is one-round communication between signers and an aggregator, not among signers, and the communication complexity is smaller than the Bellare-Neven multi-signature scheme [BN06].

We prove PCAS secure under the hardness of the DL problem in the random oracle model and the KOSK model. PCAS is similar to a multi-signature scheme CoSi [STV+16]. In [DEF+19], Drijvers et al. show the impossibility results and the sub-exponential forger against CoSi and other multi-signature scheme built from the Schnorr signature [BCJ08, MWLD10, MPSW18], and Benhamouda et al. proposed the polynomial-time forger against CoSi. To prevent these attacks, we introduce the random value t technique. For more details of this technique, see the construction of PCAS and the explanation in Section 4.4. Doe to this technique, the signature size depends on the number of signers. However, PCAS achieves a smaller size than Zhao's scheme. In practice, we need the proof-of-possession because of the KOSK model.

Contribution 3. We proposed a pairing-free one-time aggregate signature scheme OTAS by extending the Bellare-Shoup one-time signature scheme. In this protocol, one key generation allows generating one aggregate signature. We prove OTAS tightly secure based on the one-more discrete logarithm (OMDL) assumption [PV05] and the existence the collision-resistance hash function in the KOSK model. OTAS achieves the constant signature size and non-interactive aggregation, while the key size is larger than PCAS.

1.2 Related Works

Boneh et al. suggested the idea of aggregate signatures and proposed the first aggregate signature scheme using pairing [BGLS03]. Bellare et al. showed that the aggregate signature scheme [BGLS03] is secure even if the restriction of different pairs of a public key and a message between all signers is eliminated [BNN07]. There are many pairing-based aggregate signature schemes [Nev08, LOS+06, BGOY07, MT07, AGH10, HKW15, HSW13].

Lysyanskaya et al. introduced a notion of sequential aggregate signatures, where signers sequentially generate a signature on his message by using previous signers' messages and signatures and provided the first sequential aggregate signature scheme built from the RSA assumption [LMRS04]. After that, pairing-based sequential aggregate signature

schemes [LOS+06, FLS12, LLY13] and pairing-free sequential aggregate signature schemes [Nev08, BGR12, BMP16] were proposed.

Gentry and Ramzan proposed the first aggregate signature in the synchronized setting [GR06], and Ahn et al. formalized the synchronized aggregate signatures, in which signatures generated in the same period can be compressed into an aggregate signature. Hohenberger and Waters provided an RSA-based synchronized aggregate signature scheme [HW18]. We can implement an AS with PreCom scheme using a synchronized aggregate signature scheme as follows. In a PreCom phase, the signers can agree on the time period by pre-communication, and in the signing phase, each signer generates a signature using the time period agreed on in the PreCom phase. A restriction of this approach is that the number of the signatures the signers can issue is bounded at the setup time. Our proposed scheme does not have such a restriction. Another drawback of this approach is the difficulty in establishing the agreement on a time period. Namely, a malicious participant may try to make the parties reuse a time period (Notice that in the synchronized aggregate signature, the time period cannot be reused securely). An obvious solution for preventing this type of attacks is to use a Byzantine agreement protocol [LSP82], however, it removes the simplicity of the signature scheme.

Identity-based aggregate signatures [XZF05, GR06, BGOY07, BJ, HSW13] are the aggregate signatures in which each signer is assigned an ID and creates a signature by using a secret key that a private key generator generates by the master secret key and the signer's ID. This primitive is a type of aggregate signatures with a trusted key setup. In this primitive, rogue-key attacks are ineffective. It requires a trusted third-party, who is called the private key generator, for the management of all signers' keys and secure and authenticated channels for issuing a private key by the private key generator to each signer.

Katz and Lindell proposed the notion of the aggregate message authentication codes (MAC), which allows combining multiple MAC tags on multiple (possibly different) messages generated by different signers into one short tag [KL08]. They also formalized its definition and its security notion and proposed a construction of a secure aggregate MAC scheme from a secure MAC scheme by simply computing the XOR of all the individual MAC tag values. It can generate aggregate tags without interactions between signers in singing protocol. In Addition, rogue-key attacks are meaningless in this primitive since the aggregate MAC is a symmetric-key cryptographic primitive. Instead, it requires each signer to share his key with verifiers in secrecy by executing the key exchange protocols.

Bellare and Neven proposed a DL-based multi-signature scheme and mentioned the applicability of multi-signatures to (interactive) aggregates signature [BN06]. This appli-

cation presupposes that signers can share messages.

Zhao proposed an aggregate signature scheme for blockchain applications [Zha19]. This scheme is asynchronous and constructed from general elliptic curves. He stated that the proposed scheme is more applicable to blockchain applications than pairing-based aggregate signatures for the system complexity and the verification speed. His scheme is an extension of the Γ-signature [YZ13] to aggregate signatures. Though the signature size linearly depends on the number of signers, this scheme is proved secure in the plain PK model and requires no communication between signers for signing. The security of this scheme is based on the non-malleable discrete logarithm (NMDL) assumption. This assumption is only justified in the generic group model [Mau05] with random oracles, where an adversary is allowed to query both of the random oracle and the generic group oracle.

Boneh and Kim proposed a one-time aggregate signature scheme [BK20]. This scheme achieves the security based on the DL assumption in the plain PK model, while there is a large reduction loss.

Chapter 2

Preliminaries

In this chapter, we first show the notions and definitions of computational assumptions and problems. Second, we review the Bellare-Neven general forking lemma [BN06]. Finally, we show the definitions of the standard aggregate signatures and one-time aggregate signatures. Also, we show some security definitions for the above two primitives.

2.1 Notations

For a prime integer q, we denote the ring of integers modulo q by Z_q and the multiplicative group of Z_q by Z_q^*. Let G be a cyclic group of order q and let g be a generator of G. In Chapter 3, we regard G as an additive group. In Chapters 4 and 5, we regard G as a multiplicative group.

For a set A, we write $a \leftarrow\$\ A$ to mean that a is chosen at uniformly random from A. For a probabilistic algorithm B, we write $b \leftarrow \mathcal{B}(\beta_1, \ldots; \rho)$ to mean that B on inputs $\beta_1, \ldots$ and random tape ρ outputs b, and $b \leftarrow\$\ \mathcal{B}(\beta_1, \ldots)$ to mean that ρ is chosen at uniformly random and let $b \leftarrow \mathcal{B}(\beta_1, \ldots; \rho)$.

For strings $a_1, \ldots, a_n$, we denote the concatenation of them by $a_1 || \ldots || a_n$.

2.2 Computational Assumptions and Problems

In this section, we show the definitions of computational problems and assumptions which are used in this paper.

The discrete logarithm (DL) assumption is well-established. The security of many cryptographic schemes is proved under this assumption.

Definition 2.2.1 (Discrete Logarithm Assumption) *For (G, g, q), let $\mathcal{E}$ be a PPT algorithm that is given y chosen at uniformly random from a multiplicative cyclic group*

G. We say that $\mathcal{E}$ (t, ε)-breaks DL if $\mathcal{E}$ runs in time at most t and outputs x such that $y = g^x$ with probability at least ε.

One-more discrete logarithm (OMDL) assumption is a stronger assumption than the DL assumption. However, no algorithm which can break this assumption is found so far. Also, the lower bound on the complexity in the Generic Group Model (GGM) is given [CDG18].

Definition 2.2.2 (One-More Discrete Logarithm Assumption [PV05]) *Let $\mathrm{DL}(\cdot)$ be the discrete log oracle that on input y, outputs x such that $y = g^x$. For (G, g, q), let $\mathcal{E}$ be a PPT algorithm that is given $y_0, y_1, \ldots, y_\ell$ chosen at uniformly random from multiplicative cyclic group G and can access to $\mathrm{DL}(\cdot)$ at most ℓ times. We say that $\mathcal{E}$ (t, ε)-breaks $(\ell + 1)$-OMDL if $\mathcal{E}$ runs in time at most t and outputs $x_0, x_1, \ldots, x_\ell$ such that $y_i = g^{x_i}$ for all $i \in [1, \ell]$ with probability at least ε.*

The following is the definition of the collision resistance hash function.

Definition 2.2.3 (Collision-Resistant Hash Function) *Let K be a hash key chosen at uniformly random from $\{0,1\}^k$. For a hash function $H : \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^*$, we define the advantage of a PPT algorithm $\mathcal{A}$ that finds a collision of H.*

$$\mathrm{Adv}_H(\mathcal{A}) = \Pr[H(K, x_1) = H(K, x_2) \wedge x_1 \neq x_2 : K \leftarrow\$ \{0,1\}^k, (x_1, x_2) \leftarrow \mathcal{A}(K)]$$

A hash function H is called collision-resistant hash function if $\mathrm{Adv}_H(\mathcal{A})$ is negligible for all $\mathcal{A}$.

We recall the definition of the k-sum problem.

Definition 2.2.4 (k-sum problem) *The k-sum problem in group $(Z_q; +)$ for an arbitrary q provides k lists $L_1, \ldots, L_k$ of equal sizes, each list containing s_L elements sampled uniformly and independently from Z_q, and requires to find $x_1 \in L_1, \ldots, x_k \in L_k$ s.t. $\sum_{i=1}^{k} x_i \equiv 0 \pmod{q}$.*

In [Wag02], Wagner proposed the k-tree algorithm which can solve the k-sum problem for $s_L = 2^{\log q/(1+\log k)}$ in time at most $O(k2^{\log q/(1+\log k)})$ with non-negligible probability. This algorithm is used to attack some cryptographic schemes, and we also exploit this algorithm in Chapter 3.

2.3 General Forking Lemma

We use the Bellare-Neven general forking lemma to prove the security of our proposed scheme in Chapter 4.

Lemma 2.3.1 (General Forking Lemma) *Fix an integer $q \geq 1$ and a set H of size $h \geq 2$. Let A be a randomized algorithm that on input $x, h_1, \ldots, h_q$ returns a pair, the first element of which is an integer in the range $0, \ldots, q$ and the second element of which we refer to as a side output. Let IG be a randomized algorithm that we call the input generator. The accepting probability of A, denoted* acc*, is defined as the probability that $J \geq 1$ in the experiment*

$$x \leftarrow\$\ \mathsf{IG};\ h_1, \ldots, h_q \leftarrow\$\ H;\ (J, \sigma) \leftarrow\$\ \mathsf{A}(x, h_1, \ldots, h_q).$$

The forking algorithm F_{A} associated to A is the randomized algorithm that takes input x proceeds as follows:

Algorithm $\mathsf{F}_{\mathsf{A}}(x)$
- *Pick coins ρ for A at random*
- $h_1, \ldots, h_q \leftarrow\$\ H$
- $(I, \sigma) \leftarrow \mathsf{A}(x, h_1, \ldots, h_q; \rho)$
- *If $I = 0$ then return* $(0, \varepsilon, \varepsilon)$
- $h'_I, \ldots, h'_q \leftarrow\$\ H$
- $(I', \sigma') \leftarrow\$\ \mathsf{A}(x, h_1, \ldots, h_{I-1}, h'_I, \ldots, h'_q; \rho)$
- *If* $(I = I' \wedge h_I \neq h'_I)$ *then return* $(1, \sigma, \sigma')$
- *Else return* $(0, \varepsilon, \varepsilon)$

Let

$$\mathrm{frk} = \Pr[b = 1 : x \leftarrow\$\ \mathsf{IG}; (b, \sigma, \sigma') \leftarrow\$\ \mathsf{F}_{\mathsf{A}}(x)] \tag{2.1}$$

Then

$$\mathrm{frk} \geq \mathrm{acc} \cdot \left(\frac{\mathrm{acc}}{q} - \frac{1}{h} \right) \tag{2.2}$$

2.4 Aggregate Signatures

Aggregate signatures (AS) is a cryptographic primitive which allows combining individual signatures on different messages into a compact one. In this section, We show the definitions of two types of aggregate signature schemes.

2.4.1 Standard Aggregate Signatures

Boneh et al. introduced a notion of AS and proposed the first pairing-based AS scheme [BGLS03]. In this paper, we call such AS the *standard aggregate signatures* (sAS). The feature of this type is that all signers can generate individual signatures on their messages without interactions. The definition of the sAS is as follows.

Definition 2.4.1 *An sAS scheme consists of the following six algorithms. Let n be the number of signers.*

Setup$(1^\lambda) \to pp$**.** *The public parameter generation algorithm takes as input a security parameter 1^λ, then it outputs a public parameter pp.*

KeyGen$(pp) \to (pk, sk)$**.** *The key generation algorithm takes as input a public parameter pp, then it outputs a public key pk and a secret key sk.*

Sign$(pp, pk, sk, m) \to \sigma$**.** *The signing algorithm takes as input a public parameter pp, a public key pk, a secret key sk, and a message m, then it outputs a individual signature σ.*

Verify$(pp, pk, m, \sigma) \to \{0, 1\}$ *The verification algorithm takes as input a public parameter pp, a public key pk, a message m, and a signature σ, then it outputs 0 (REJECT) or 1 (ACCEPT).*

Agg$(pp, \{(pk_i, m_i, \sigma_i)\}_{i=1}^n) \to \sigma_a$**.** *The aggregation algorithm takes as input a public parameter pp, and a set of all signers' public keys, messages, and signatures $\{(pk_i, m_i, \sigma_i)\}_{i=1}^n$, then it outputs an aggregate signature σ_a.*

AggVer$(pp, \{(pk_i, m_i)\}_{i=1}^n, \sigma_a) \to \{0, 1\}$**.** *The aggregate signature verification algorithm takes as input a public parameter pp, a set of all signers' public keys and messages $\{(pk_i, m_i)\}_{i=1}^n$, and an aggregate signature σ_a, then it outputs 0 (REJECT) or 1 (ACCEPT).*

For any set of messages $\{m_i\}_{i=1}^n$, if a public parameter pp, all signers' public keys $\{pk_i\}_{i=1}^n$, and an aggregate signature σ_a are generated honestly by the above algorithms, then we require that $\Pr[\textbf{AggVer}(pp, \{(pk_i, m_i)\}_{i=1}^n, \sigma_a) = 1] = 1$.

Attacks and Key-Setup Model

Before showing the definition of security, we should describe the key-setup assumptions. In aggregate signatures, one should consider an attack scenario which is called the *rogue-key attack*. In rogue-key attacks, an attacker generates public keys dishonestly and tries

to forge a combined signature involving such dishonest keys. In general, we can guarantee the security of the scheme against the rogue-key attacks using the following two basic strategies. The first is (i) *to prove directly that there exists no rogue-key attack*, and the second is (ii) *to exclude rogue-key attacks by a specific key registration protocol*. These two approaches are formally modeled by (i) *the plain public-key (PK) model* [BN06] and (ii) *the knowledge of secret keys (KOSK) model* [Bol03, LOS+06], respectively.

(i) *The plain PK model* is the model without any assumption in the key setup. In the security model, an adversary can freely choose all cosigners' public keys excluding at least one honest signer's key.

(ii) *The KOSK model* is the model where all signers need to prove the validity of their public key. In the security model, an adversary can freely pick all cosigners' public keys, but it must output the secret keys corresponding to these public keys. In practice, the KOSK model can be implemented using one of the following models: (1) a *trusted setup* model [MOR01], in which a dedicated key registration protocol is needed to be executed by each signer, (2) the *key verification* (KV) model [BCJ08], and (3) the *proof-of-possession* (PoP) model [RY07], where each signer submits a certificate to prove possession of a secret key.

Note that the KOSK model is a theoretical key-setup model and is a stronger assumption than the plain PK model. The security definitions in this paper consider either the plain PK model or the KOSK model.

Security Models for Standard Aggregate Signatures sAS

In this part, We show the two security model for sAS.

EUF-CMA in the Plain PK Model and the Random Oracle Model for sAS. Below, we show the definition of existential unforgeability under the chosen-message attack to sAS in the random oracle (RO) model and the plain PK model [BN06]. Existential unforgeability is that there is no forger who can generate at least one pair of a message and a forgery where a message has not been signed. A chosen-message attack is that a forger can obtain signatures on messages which are chosen by a forger. Specifically, in this model, a forger who corrupts an aggregator and signers except one honest signer is given on honest signer's public key and is allowed making signing queries. Finally, a forger needs to output a forgery on a message which has never appeared in signing queries and random oracle queries. Also, a forger can choose cosigners' public keys freely without proving the validity of keys.

Formally, we consider the following game.

Setup. The challenger chooses the parameter $pp \leftarrow\$\ \mathbf{Setup}(1^\lambda)$ and the key pair $(pk, sk) \leftarrow\$\ \mathbf{KeyGen}(pp)$. It runs a forger $\mathcal{F}$ on input pk and pp.

Signing Queries. The challenger receives m' as a signing query, computes $\sigma' \leftarrow \mathbf{Sign}(pp, pk, sk, m)$ and return σ' as an honest signer's individual signature on m'.

Output. $\mathcal{F}$ outputs n public keys $\{pk_i\}_{i=1}^n$, a set of messages $\{m_i^*\}_{i=1}^n$, and a forgery σ_a^* where the following holds.

- $(pk_1, m_1^*), \ldots, (pk_n, m_n^*)$ are mutually distinct.
- $pk \in \{pk_i\}_{i=1}^n$.

If $\mathbf{AggVer}(pp, \{(pk_i, m_i^*)\}_{i=1}^n, \sigma_a^*) = 1$ is true and m_i^* has never been queried to the signing oracle where i is such that $pk_i = pk$, then $\mathcal{F}$ is said to succeed in forgery.

Definition 2.4.2 *Let N be a maximum number of cosigners being involved in the forgery. We say that $\mathcal{F}$ $(t, q_S, q_H, N, \varepsilon)$-break sAS scheme in the plain PK model if $\mathcal{F}$ runs in at most t time, makes at most q_S signing queries and at most q_H random oracle queries, and succeeds in forgery in the above game with probability at least ε. For an sAS scheme, if there are no $\mathcal{F}$ that $(t, q_S, q_H, N, \varepsilon)$-breaks it in the plain PK model, we say the scheme is $(t, q_S, q_H, N, \varepsilon)$-secure in the plain PK model.*

UUF-KOA in the KOSK Model and the Random Oracle Model for sAS. For sAS, we define *universal unforgeability under key-only attacks* [GMR88] in the RO model and the KOSK model. Universal forgeability is that there is a forger who can generate a forgery on an arbitrary message and is more serious than existential forgeability. A key-only attack does not allow a forger to make a signing query. Specifically, in this security model, a forger who corrupts an aggregator and signers except one honest signer is given an honest signer's public key and a message and is required to generate a forgery on the given message by making random oracle queries. When outputting a forgery, it must output cosigners' secret keys corresponding to cosigners' public keys which are chosen arbitrarily.

If, for all m^*, a forger $\mathcal{F}$ wins the following game with non-negligible probability, then we say that $\mathcal{F}$ is a universal forger under a key-only attack in the KOSK model.

$\mathbf{Setup}(1^\lambda, m^*)$. The challenger chooses a public parameter $pp \leftarrow\$\ \mathbf{Setup}(1^\lambda)$, an honest signer's key pair $(pk, sk) \leftarrow\$\ \mathbf{KeyGen}(pp)$. It runs a forger $\mathcal{F}$ on input pp, pk and a message m^*.

Output. $\mathcal{F}$ outputs n key pairs $\{(pk_i, sk_i, m_i^*)\}_{i=1}^n$ and a forgery σ_a^* where the following holds.

- $(pk_1, m_1^*), \ldots, (pk_n, m_n^*)$ are mutually distinct.
- $(pk, m^*) \in \{(pk_i, m_i^*)\}_{i=1}^n$.
- sk_l is $\perp$ where l satisfies s.t. $pk_l = pk$.
- sk_i is a correct secrete key corresponding to pk_i for $i \in [1, n] \backslash \{l\}$.

If $\mathbf{AggVer}(pp, \{(pk_i, m_i^*)\}_{i=1}^n, \sigma_a^*) = 1$ holds, then $\mathcal{F}$ wins.

Definition 2.4.3 *Let N be a maximum number of cosigners being involved in the forgery. We say that $\mathcal{F}$ (t, q_H, N, ε)-break sAS in the KOSK model if $\mathcal{F}$ runs in at most t time, makes at most q_H random oracle queries, and succeeds in forgery in the above game with probability at least ε.*

2.4.2 One-Time Aggregate Signatures

Here, we define the one-time aggregate signatures (OTAS). In a protocol of this primitive, all signers are required to generate new key pairs each time they generate a new aggregate signature.

Definition 2.4.4 (OTAS) *OTAS consists of the following five algorithms. Let n be the number of signers and let i be the index of a signer.*

Setup$(1^\lambda) \to pp$**.** *The public parameter generation algorithm takes as input a security parameter 1^λ, then outputs a public parameter pp.*

KeyGen$(pp) \to (pk, sk)$**.** *The key generation algorithm takes as input a public parameter pp, then outputs a public key pk and a secret key sk.*

Sign$(pp, pk, sk, m) \to \sigma$**.** *The signing algorithm takes as input a public parameter pp, a public key pk, a secret key sk and a message m, then outputs a signature σ.*

Agg$(pp, \{(pk_i, m_i, \sigma_i)\}_{i=1}^n) \to \sigma_a$**.** *The aggregation algorithm takes as input a public parameter pp and a set of all signers' public keys, messages, and signatures $\{(pk_i, m_i, \sigma_i)\}_{i=1}^n$, then outputs an aggregate signature σ_a.*

AggVer$(pp, \{(pk_i, m_i)\}_{i=1}^n, \sigma_a) \to \{0, 1\}$**.** *The aggregate signature verification algorithm takes as input a public parameter pp, a set of all signers' public keys and messages $\{(pk_i, m_i)\}_{i=1}^n$ and an aggregate signature σ_a, then outputs 0 (REJECT) or 1 (ACCEPT).*

For any set of messages $\{m_i\}_{i=1}^n$*, if all signers and an aggregator behave honestly, then* $\Pr[\mathbf{AggVer}(pp, \{(pk_i, m_i)\}_{i=1}^n, \sigma_a) = 1] = 1$.

Security Models for OTAS

Below, we show a security models for OTAS. The important difference of the security models of OTAS from ones of sAS is that the forger $\mathcal{F}$ is allowed to make only one signing query.

EUF-CMA in the KOSK Model for OTAS. The following is the definition of existential unforgeability under chosen-message attacks of OTAS in the KOSK model. We should note that all signers' public keys are mutually distinct, not their pairs of a public key and a message, and this is not in the RO model. The security model here is defined by the three-phase game of the following.

Setup. The challenger chooses the parameters $pp \leftarrow\$\ \mathbf{Setup}(1^\lambda)$ and the key pair $(pk, sk) \leftarrow\$\ \mathbf{KeyGen}(pp)$. It runs the forger $\mathcal{F}$ on input pk and pp.

Signing Query. The challenger receives m' as a query, computes $\sigma' \leftarrow \mathbf{Sign}(pp, pk, sk, m')$, and returns σ' to $\mathcal{F}$. $\mathcal{F}$ is allowed to make only one query.

Output. After $\mathcal{F}$ terminates, it outputs $(n-1)$ cosigners' key pairs $\{(pk_i, sk_i)\}_{i=1}^n$, a set of messages $\{m_i^*\}_{i=1}^n$, and a forgery σ_a^* where the following holds.

- $pk_1, \ldots, pk_n$ are mutually distinct.
- $pk \in \{pk_i\}_{i=1}^n$.
- sk_k is $\perp$ where k is such that $pk_k = pk$.

If $\mathbf{AggVer}(pp, \{(pk_i, m_i)\}_{i=1}^n, \sigma_a) = 1$ is true and m_i^* has never been queried where i is such that $pk_i = pk$, then $\mathcal{F}$ is said to succeed in forgery.

Definition 2.4.5 (Unforgeability in KOSK Model for OTAS) *Let N be a maximum number of cosigners being involved in the forgery, we say that $\mathcal{F}$ (t, N, ε)-breaks the OTAS scheme in the KOSK model if $\mathcal{F}$ runs in at most t time and succeeds forgery in the above game with probability at least ε. For an OTAS scheme, if there are no $\mathcal{F}$ that (t, N, ε)-breaks it in the KOSK model, we say the scheme is (t, N, ε)-secure in the KOSK model.*

Chapter 3

Cryptanalysis of Aggregate Γ-Signature

Zhao proposed a standard aggregate signature (sAS) scheme [Zha19], which is named aggregate Γ-Signature Scheme, from *general elliptic curve (EC) groups* for application to Bitcoin [Nak08]. However, a sub-exponential time universal forger under a key-only attack in the plain PK model is proposed anonymously by ncklr. This forger executes a rouge-key attack by exploiting a k-sum algorithm [Wag02], which is the cause of sub-exponential time. Namely, this forger is effective only in the plain PK model, and we can prevent this attack by using proof-of-possession [RY07].

Here, we show a stronger forger than the above one. Specifically, our forger is a sub-exponential forger under a key-only attack in the KOSK model. We cannot this attack even if a trusted key-setup is executed because our attack is effective in the KOSK model. Note that our attack is not fatal theoretically since this is a sub-exponential time attack, but a practical performance is damaged. For more detail, see Section 3.2.2.

3.1 Aggregate Γ-Signature Scheme

In [Zha19], the aggregate Γ-signature (AGS) scheme is proposed by Zhao. This scheme is a standard aggregate signature scheme from general EC groups, while allows partially combining individual signatures, namely the size of an aggregate signature depend on the number of the signers lineally. AGS consists of the following six algorithms. Note that we regard the underlying group G as an additive cyclic group.

Setup$(1^\lambda) \to (G, q, P, H_0, H_1)$**.** It chooses (G, q, P), hash functions $H_0 : G \to Z_q$ and $H_1 : G \times M \to Z_q$ where M is the set of messages, then it outputs $pp = (G, q, P, H_0, H_1)$.

KeyGen$(pp) \to (X, x)$**.** It computes $x \leftarrow\$\ Z_q^*$ and $X \leftarrow xP$, then it outputs a public

key X and a secret key x.

Sign$(pp, X, x, m) \to \sigma$**.** It computes $r \leftarrow\$\ Z_q^*$, $A \leftarrow rP$, $d \leftarrow H_0(A)$, and $e \leftarrow H_1(X, m)$. It computes $z \leftarrow rd - ex \mod q$, then it outputs $\sigma = (z, d)$ as a signature.

Verify$(pp, X, m, \sigma) \to \{0, 1\}$ It computes $e \leftarrow H_1(X, m)$ and $A \leftarrow zd^{-1}P + ed^{-1}X$. If $H_0(A) \neq d$ holds, then it outputs 0. Otherwise it outputs 1.

Agg$(pp, \{(X_i, m_i, \sigma_i)\}_{i=1}^n) \to (\hat{T}, \hat{A}, z)$**.** It initializes $\hat{T} = \emptyset$, $\hat{A} = \emptyset$, and $z = 0$. For $i = 1$ to n, if **Verify**$(pp, X_i, m_i, \sigma_i) = 1 \wedge (X_i, m_i) \notin \hat{T} \wedge A_i \notin \hat{A}$ holds, it sets $\hat{T} \leftarrow \hat{T} \cup \{(X_i, m_i)\}$ and $\hat{A} \leftarrow \hat{A} \cup \{A_i\}$ and computes $z \leftarrow z + z_i \mod q$. Finally, it outputs $(\hat{T}, \hat{A}, z)$.

AggVer$(pp, (\hat{T}, \hat{A}, z)) \to \{0, 1\}$**.** If the elements in $\hat{T}$ are not mutually distinct, the elements in $\hat{A}$ are not mutually distinct, or $|\hat{T}| \neq |\hat{A}|$ holds, then outputs 0. It sets $n' \leftarrow |\hat{T}|$, and for $j = 1$ to n', it computes $d_j \leftarrow H_0(A_j)$ and $e_j \leftarrow H_1(X_j, m_j)$. If $\sum_{j=1}^{n'} d_j A_j = zP + \sum_{j=1}^{n'} e_j X_j$ holds, it outputs 1, Otherwise it outputs 0.

Zhao presented the ephemeral rouge-key attack against an intuitive AS scheme built from the Schnorr signature which combines only the response components of the Σ-protocol [Cra96] and showed that the above AS scheme can prevent this attack. Also the security of this scheme is proved based on the non-malleable discrete logarithm (NMDL) assumption. We review the definition of this assumption.

Definition 3.1.1 (Non-Malleable Discrete Logarithm (NMDL) Assumption) *Let $H_1, \ldots, H_K : \{0, 1\}^* \to Z_q^*$ be cryptographic hash functions, which may not be distinct. On input (G, P, q, X) where $X = xP$ for $x \leftarrow Z_q^*$ a PPT algorithm $\mathcal{A}$ (called an NMDL solver) succeeds in solving the NMDL problem, if it outputs $(\{b_i, Y_i, m_i\}_{i=1}^K, z)$ satisfying:*

- *$z \in Z_q$, and for any i, $1 \leq i \leq K$, $Y_i \in G$, $m_i \in \{0, 1\}^*$ that can be the empty string, and $b_i \in \{0, 1\}$.*
- *For any $1 \leq i, j \leq K$, it holds that $(Y_i, m_i) \neq (Y_j, m_j)$. It might be the case that $Y_i = Y_j$ or $m_i = m_j$.*
- *$X \in \{Y_i\}_1^K$, and $zP = \sum_{i=1}^K (-1)^{b_i} e_i Y_i$ where $e_i = H_i(Y_i, m_i)$.*

The NMDL assumption means that there are no PPT algorithm which succeeds in solving the NMDL problems with non-negligible probability in $\log q$.

For more detail of this assumption, see Section 5.1 of [Zha19].

3.2 Sub-Exponential Universal Forgery under a Key-Only Attack against Aggregate Γ-Signature in the KOSK Model

Here we present a sub-exponential universal forger under a key-only attack against the aggregate Γ-signature in the KOSK model. The cause of this cryptanalysis is that there is an algorithm that can solve the NMDL problem in sub-exponential time by using a k-sum algorithm.

The input and the goal of a forger against aggregate Γ-signature in the security game in Definition 2.4.3 are as follows:

Input: A challenge key X_1 and a target message m_1^*.

Goal: To output a forgery $(z^*, \{A_i\}_{i=1}^n)$ and a set of cosigners' keys and messages $\{(X_i, x_i, m_i^*)\}_{i=2}^n$ s.t. the following holds:

$$\sum_{i=1}^{n} d_i A_i = z^* P + \sum_{i=1}^{n} e_i X_i \tag{3.1}$$

where $X_i = x_i P$ for $i \in [2, n]$, $d_i = H_0(A_i)$, and $e_i = H_1(X_i, m_i^*)$ for $i \in [1, n]$.

Now, we explain an overview of our forger. To achieve the above goal, our forger generates ephemeral rogue-keys by exploiting a n-sum algorithm. Specifically, it chooses uniformly $r_i \leftarrow\$\ Z_q^*$ and computes an ephemeral rogue-key $A_i \leftarrow r_i P + X_1$ for each signer, respectively. In this case, for the equation (3.1), when we assume that (i) $\sum_{i=1}^n d_i = e_1$ holds, the terms related to X_1 are canceled out. Then this forger can compute a consistent z^* because it knows discrete logarithms corresponding to remaining terms. Thus, to achieve the goal, it is sufficient for the forger to obtain a set of ephemeral rogue-keys $\{A_i\}_{i=1}^n$ which make (i) hold. A n-sum algorithm is used for such a purpose. Concretely, the forger prepares many ephemeral keys and finds a set of such keys $\{A_i\}_{i=1}^n$ by using an n-sum algorithm.

Below, we show the procedure of our proposed forger $\mathcal{F}$.

Main Procedure

1. Choose arbitrary cosigners' secret keys $\{x_i\}_{i=2}^n \in (Z_q^*)^{(n-1)}$ and assign the public keys as follows:

$$X_2 \leftarrow x_2 P, \ldots, X_n \leftarrow x_n P. \tag{3.2}$$

2. Launch an n-sum attack via $n \cdot s_L$ times hash computations to obtain $\{(d_i, r_i, A_i)\}_{i=1}^{n}$ s.t. the following holds:

$$\sum_{i=1}^{n} d_i \equiv e_1 \pmod{q} \tag{3.3}$$

where $A_i = r_i P + X_1$, $d_i = H_0(A_i)$ for $i \in [1, n]$ and $e_1 = H_1(X_1, m_1^*)$.

3. Choose any messages $\{m_i^*\}_{i=2}^{n}$ and assign the followings:

$$e_2 \leftarrow H_1(X_2, m_2^*), \ldots, e_n \leftarrow H_1(X_n, m_n^*), \tag{3.4}$$

$$z^* \leftarrow -\sum_{i=2}^{n} x_i e_i + \sum_{i=1}^{n} r_i d_i. \tag{3.5}$$

4. Output $(z^*, \{A_i\}_{i=1}^{n})$ and $\{(X_i, x_i, m_i^*)\}_{i=2}^{n}$.

In Step 2 of the above, $\mathcal{F}$ executes the n-sum algorithm according to the following.

n-sum Attack Procedure

1. Choose $\{r_{i,j}\}_{i=1,j=1}^{n,s_L} \in (Z_q^*)^{n \times s_L}$ and computes $\{A_{i,j}\}_{i=1,j=1}^{n,s_L}$ where

$$A_{i,j} = r_{i,j} P + X_1. \tag{3.6}$$

2. Compute $d_{i,j} \leftarrow H_0(A_{i,j})$ for $i \in [1, n], j \in [1, s_L]$.
3. Make lists as follows:

$$L_1 \leftarrow \{d_{1,j} - e_1\}_{j=1}^{s_L},$$
$$\text{and } L_i \leftarrow \{d_{i,j}\}_{j=1}^{s_L} \text{ for } i \in [2, n].$$

4. Run the n-sum algorithm on input the $n-1$ lists $\{L_i\}_{i=1}^{n}$ to obtain $\{d_{i,j_i}\}_{i=1}^{n}$ s.t. Eq. (3.3) holds.
5. Output $\{(d_{i,j_i}, r_{i,j_i}, A_{i,j_i})\}_{i=1}^{n}$.

Correctness

Now we confirm the correctness of the above attack procedure. For an output of $\mathcal{F}$, $(z^*, \{A_i\}_{i=1}^{n})$ and $\{(X_i, x_i, m_i^*)\}_{i=2}^{n}$, we have the following equations hold:

$$
\begin{aligned}
& z^*P + \sum_{i=1}^{n} e_i X_i \\
&= (-\sum_{i=2}^{n} x_i e_i + \sum_{i=1}^{n} r_i d_i)P + e_1 X_1 + \sum_{i=2}^{n} e_i x_i P \quad (\text{from, Eq.(3.5)}) \\
&= \sum_{i=1}^{n} r_i d_i P + e_1 X_1 \\
&= \sum_{i=1}^{n} r_i d_i P + \left(\sum_{i=1}^{n} d_i\right) X_1 \quad (\text{from Eq.(3.3)}) \\
&= \sum_{i=1}^{n} d_i (r_i P + X_1) \\
&= \sum_{i=1}^{n} d_i A_i \quad (\text{from Eq.(3.6)}).
\end{aligned}
$$

3.2.1 Computational Complexity

By using Wagner's k-tree algorithm, Step 4 of **n-sum Attack Procedure** takes at most $O\left(n2^{\log q/(1+\log n)}\right)$ time. In addition, in **Main Procedure**, there are $n-1$ exponentiations and n computations of the hash function in Steps 1 and 3, respectively. Also, in **n-sum Attack Procedure**, there are respectively $n \times s_L$ exponentiations and $n \times s_L$ computations of the hash function in Steps 1 and 2 where s_L is $2^{\log q/(1+\log n)}$.

A value of $n2^{\log q/(1+\log n)}$ is minimized when $n = 2^{\sqrt{\log q}-1}$. In particular, assuming that the bit-length of q is 256-bits, the running time of the above forger is minimized to $O(2^{31})$ when n is approximately 2^{15}. In this parameter (i.e., $n = 2^{15}$), the number of cosigners should be fixed to $2^{15}-1$ and cannot be chosen flexibly. Instead, if we want to reduce the number of cosigners, we can mount the above attack with a smaller n at the cost of much time and space complexity. In the reality of the Bitcoin system, n is about 2^{12} and the time complexity is about $O(2^{32})$. Note that this complexity is almost the same as the optimal. Fig. 3.1 shows the relation between $n2^{\log q/(1+\log n)}$ and $\log n$, namely, the one between the complexity of an n-sum algorithm and the number of signers.

3.2.2 Impact in the Implementation

The sub-exponential time forger is not fatal theoretically. But our forger breaks the primary advantage of sAS scheme from EC groups. Specifically, in order to guarantee the

security against our attack, the aggregate Γ-signature scheme requires the bit-length of the order of an underlying group to be approximately $\log n$ times the security parameter where n is the number of signers. In contrast, for most other schemes based on general EC groups, the bit-length of the order of the underlying group is only twice as long as the security parameter (due to the ρ-method [Pol78]).

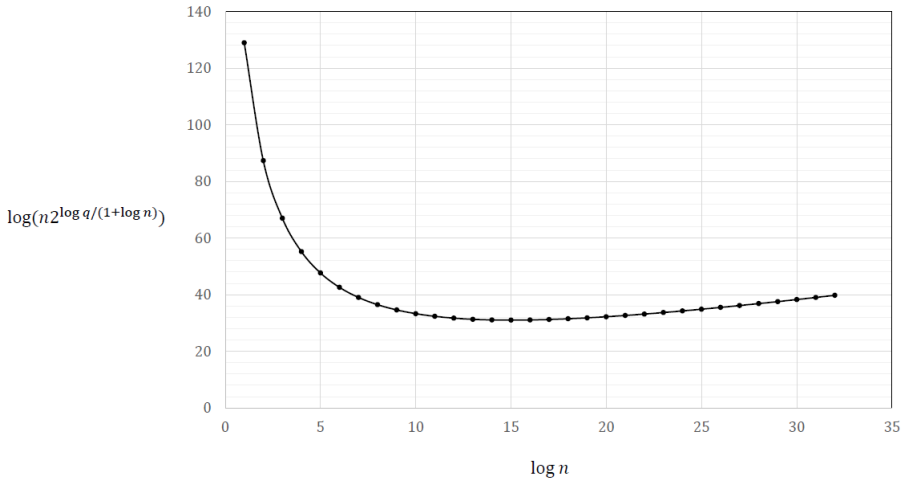

Figure 3.1: Complexity of an n-sum algorithm where $\log q = 256$.

3.3 Inapplicability in Bitcoin System

In this section, we explain that our forger is not a threat in the Bitcoin system and two countermeasures to obstruct our forger. First, we roughly describe the background of Bitcoin and the blockchain. After that, we show the real impact of our forger in the Bitcoin system and two countermeasures.

3.3.1 Blockchain

Blockchain is one of the technologies for realizing Bitcoin which is a cryptocurrency scheme introduced by Satoshi Nakamoto [Nak08]. This allows managing a ledger, guaranteeing unforgeability, and achieving decentralization. Namely, nobody can tamper with transactions that are managed by a publicly verifiable distributed ledger without reliable administers. Blockchain is gathering attention globally in recent years due to the increasing popularity of Bitcoin and is applied not only to a cryptocurrency but also to other industries.

In Bitcoin, the EC-DSA signature scheme [JMV01] over the secp256k1 curve [Res10] is used to authenticate transactions. The size of signatures and verification time are important terms for designing the Bitcoin system because Bitcoin nodes need to verify all updates to the ledger. Because of the non-linearity of the EC-DSA signature, it is hard to combine signatures into a compact one while keeping verifiability, and thus transactions contain a concatenation of all individual signatures. Namely, the signature size depends on the number of signatures and the signatures occupy a large part of the size of Bitcoin transactions. Recently, there are interests in deploying the Schnorr signature scheme [Sch01] in Bitcoin instead of the EC-DSA signature scheme in terms of linearity, well-established security, and small computational complexity. Specifically, the linearity of the Schnorr signature helps extensions to multi-signatures [BN06, MPSW18, BDN18].

AS can also overcome the above bottlenecks of blockchain, and to construct a sAS scheme based on the Schnorr signature scheme is gathering attention. But most intuitive schemes from such the scheme are cannot prove secure based on the standard computational assumption. Zhao showed the subtlety in constructing a secure sAS scheme from general EC groups [Zha19].

3.3.2 Real Impact in Bitcoin System and Countermeasures

Our proposed forger is not fatal in the Bitcoin system because it can be detected easily. The full nodes (where the miner is a special kind of full nodes) check the validity of all individual signatures and transactions which are submitted and broadcast to all full nodes by remitters and add pairs of individual signatures and transactions to the transaction pool if that pairs are valid. Our forger can generate forgery of an aggregate signature, however our forger cannot generate valid individual signatures. Thus in the case where the remitter launches our attack, forgery is detected by full nodes and honest miners because the remitter needs to send invalid individual signatures to all nodes. Also a mined block is broadcasted to all full nodes by the miner who won the proof-of-work (PoW) hashing and is verified the validity of the transactions and individual signatures contained in the mined block by checking whether they appear in the transaction pool. Therefore, a forgery and a victim transaction generated by a malicious miner can be detected easily by full nodes. Also, the malicious miner needs to execute not only sub-exponential time our attack but also the PoW hashing which requires high computational power to complete. So, exploiting our forger in Bitcoin system is computationally expensive and infeasible.

We also prevent our attack by modifying the algorithms of the aggregate Γ-signature scheme and Bitcoin system. More concretely, we add the nonce in the last confirmed

block to the input of H_0. Then, a forger needs to complete to generate a forgery in the limited short time (e.g., ten minutes) since it requires obtaining such a nonce to launch our attack.

For more detail of the real impact and countermeasures, see Section 5 in [HOS+20].

Chapter 4

Aggregate Signatures with Pre-Communication

Boneh et al. proposed the first pairing-based sAS scheme. This scheme can achieve a constant signature size and the non-interactive aggregation. On the other hand, it requires the assumption in groups with the bilinear map and the costly computation in the verification. Specifically, since the weakness of the pairing-based problems are discovered [KB16, Gui20], we need to use the pairing group of which the size is the 75% larger than the initial recommendation of the parameter (e.g., 448-bit for 128-bit security).

Zhao proposed an sAS scheme from general EC curves based on the hardness of the non-standard computational problem, the non-malleable discrete logarithm (NMDL) problem [Zha19]. However, we present the cryptoanalysis of this scheme in the previous chapter. Constructing pairing-free AS schemes with the security based on the standard computational assumption is important from both practical and theoretical points of view.

In this chapter, we propose an aggregate signature scheme with pre-communication which is the interaction before deciding the messages to be signed. Also, we prove this scheme secure based on the discrete logarithm assumption in the KOSK model. Due to the pre-communication, our scheme has the offline-online feature, namely, the signing algorithm consists of the only lightweight computation. The size of the aggregate signature depends on the number of the signers linearly. The linear component is the element with the same bit-length as the security parameter. For more detail of the feature of our scheme, see the comparison in Section 4.3.

4.1 Aggregate Signatures with Pre-Communication

4.1.1 Definition

Particularly, we introduce a model where, before signing, each signer communicates with the aggregator and shares information in advance, which we hereafter call helper information. Note that the communication in this model is the n-to-one communication between all signers and the aggregator where n is the number of signers. We now describe the definition of aggregate signature (AS) with pre-communication (PreCom) below. We illustrate pre-communication and aggregation in Fig.4.1.

Definition 4.1.1 (AS with PreCom) *An AS with PreCom consists of the following five algorithms and one protocol. Let n be the number of signers and let i be the index of a signer.*

Setup$(1^\lambda) \to pp$**.** *The public parameter generation algorithm takes as input a security parameter 1^λ, then it outputs a public parameter pp.*

KeyGen$(pp) \to (pk, sk)$**.** *The key generation algorithm takes as input a public parameter pp, then it outputs a public key pk and a secret key sk.*

PreCom$\langle \mathcal{S}_1(pk_1, sk_1), \ldots, \mathcal{S}_n(pk_n, sk_n), \mathcal{AG}(\{pk_i\}_{i=1}^n)\rangle \to (\tilde{h}_1, \ldots, \tilde{h}_n, z)$. *The pre-communication protocol is executed between each signer $\mathcal{S}_i$ with input a public key pk_i and a secret key sk_i and an aggregator $\mathcal{AG}$ with input all the signers' public keys $\{pk_i\}_{i=1}^n$. After the protocol terminates, each $\mathcal{S}_i$ and $\mathcal{AG}$ obtain $\tilde{h}_i$ and z as helper information, respectively.*

Sign$(pp, pk, sk, \tilde{h}, m) \to \sigma$**.** *The signing algorithm takes as input a public parameter pp, a public key pk, a secret key sk, helper information $\tilde{h}$, and a message m, then it outputs a signature σ.*

Agg$(pp, z, \{(pk_i, m_i, \sigma_i)\}_{i=1}^n) \to \sigma_a$**.** *The aggregation algorithm takes as input a public parameter pp, helper information z, and a set of all signers' public keys, messages, and signatures $\{(pk_i, m_i, \sigma_i)\}_{i=1}^n$, then it outputs an aggregate signature σ_a.*

AggVer$(pp, \{(pk_i, m_i)\}_{i=1}^n, \sigma_a) \to \{0, 1\}$**.** *The aggregate signature verification algorithm takes as input a public parameter pp, a set of all signers' public keys and messages $\{(pk_i, m_i)\}_{i=1}^n$, and an aggregate signature σ_a, then it outputs 0 (REJECT) or 1 (ACCEPT).*

For any set of messages $\{m_i\}_{i=1}^n$*, if all signers and an aggregator behave honestly, then* $\Pr[\mathbf{AggVer}(pp, \{(pk_i, m_i)\}_{i=1}^n, \sigma_a) = 1] = 1$ *holds.*

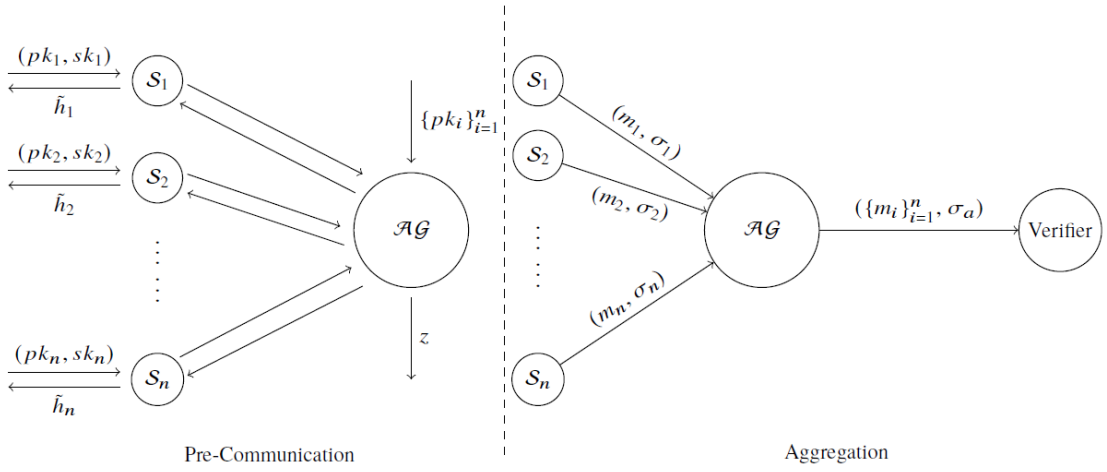

Figure 4.1: Aggregate Signature with Pre-Communication: The arrows that denote the communication are simplified to one round communication as in the proposed scheme in this paper. In our model, we do not restrict the number of rounds to one.

Security Model of AS with PreCom. Below, we show the definition of existential unforgeability under the chosen-message attack to AS with PreCom in the RO model and KOSK model. This security definition requires that it be infeasible to forge aggregate signatures involving at least one honest signer. In the security model here, as a forger $\mathcal{F}$, we consider aggregators who corrupt signers except for one honest signer. Also the forger $\mathcal{F}$ can execute the pre-communication and the aggregation protocols with an honest signer several times, and after that, it tries to output a forgery. Then the forger $\mathcal{F}$ can arbitrarily choose the corrupted cosigners' public keys even though it must output secret keys corresponding to these public keys.

Formally, the security model here is defined by the three-phase game of the following.

Setup. The challenger chooses the parameter $pp \leftarrow\$\ \mathbf{Setup}(1^\lambda)$ and the key pair $(pk, sk) \leftarrow\$\ \mathbf{KeyGen}(pp)$. It runs a forger $\mathcal{F}$ on input pk and pp.

Signing Queries. The challenger receives (j, i_j) as *a PreCom signing query.* The challenger and $\mathcal{F}$ execute the pre-communication protocol $\mathbf{PreCom}\langle \mathcal{S}_1(pk_1, sk_1), \ldots, \mathcal{S}_n(pk_n, sk_n), \mathcal{AG}(\{pk_i\}_{i=1}^n)\rangle \rightarrow (\tilde{h}_1, \ldots, \tilde{h}_n, z)$ where the challenger behaves as $\mathcal{S}_{i_j}(pk, sk)$ and all the other parties are controlled by $\mathcal{F}$. Then, the challenger obtains the helper information $\tilde{h}_{i_j}$ and stores this information with the PreCom signing query. The challenger receives (j, m') as *a message signing query.* It reads out $\tilde{h}_{i_j}$ and computes $\sigma'_j \leftarrow \mathbf{Sign}(pp, pk, sk, \tilde{h}_{i_j}, m')$. The challenger returns σ'_j to $\mathcal{F}$. $\mathcal{F}$ is allowed to concurrently make any number of above queries where it is allowed to make only one message signing query per one PreCom signing query.[1]

Output. After $\mathcal{F}$ terminates, it outputs n key pairs $\{(pk_i, sk_i)\}_{i=1}^n$, a set of messages $\{m_i^*\}_{i=1}^n$, and a forgery σ_a^* where the following holds.

- $\{pk_i\}_{i=1}^n$ is distinct to each other.
- $pk \in \{pk_i\}_{i=1}^n$.
- sk_k is $\perp$ where k is such that $pk_k = pk$.

If $\mathbf{AggVer}(pp, \{(pk_i, m_i^*)\}_{i=1}^n, \sigma_a^*) = 1$ is true and m_i^* has never been queried where i is such that $pk_i = pk$, then $\mathcal{F}$ is said to succeed in forgery.

Definition 4.1.2 *Let N be a maximum number of cosigners being involved in the forgery. We say that $\mathcal{F}$ $(t, q_S, q_H, N, \varepsilon)$-break AS with PreCom if $\mathcal{F}$ runs in at most t time, makes at most q_S signing queries and at most q_H random oracle queries, and succeeds in forgery in the above game with probability at least ε. For an AS scheme with PreCom, if there are no $\mathcal{F}$ that $(t, q_S, q_H, N, \varepsilon)$-breaks it, we say the scheme is $(t, q_S, q_H, N, \varepsilon)$-secure.*

4.2 Proposed Scheme PCAS and Security Proof

In this section, we propose the AS scheme with PreCom PCAS based on the discrete logarithm assumption in the RO model and the KOSK model. This scheme is an extension of the Schnorr signature scheme to aggregate signatures. We introduce the pre-communication to share information to aggregate between all signers and an aggregator without the communication in the signing phase.

[1]This restriction is essential. If this restriction is omitted, there is an attack against our proposed scheme. See Remark 4.2.1 for more detail.

4.2.1 The Algorithms and Protocol of PCAS

Below, we now show the algorithms and protocol of PCAS.

Setup$(1^\lambda) \to pp$**.** It chooses (G, q, g), a hash function $H : \{0,1\}^* \to Z_q$, and a parameter κ, then it outputs $pp = (G, q, g, H, \kappa)$.

KeyGen$(pp) \to (pk, sk)$**.** It computes $x \leftarrow\$\ Z_q$ and $X \leftarrow g^x$, then it outputs the public key $pk = X$ and the secret key $sk = x$.

PreCom$\langle \mathcal{S}_1(pk_1, sk_1), \ldots, \mathcal{S}_n(pk_n, sk_n), \mathcal{AG}(\{pk_i\}_{i=1}^n)\rangle \to (\tilde{h}_1, \ldots, \tilde{h}_n, z)$. For all $i \in [1, n]$, firstly, each signer $\mathcal{S}_i$ computes $r_i \leftarrow\$\ Z_q$ and $R_i \leftarrow g^{r_i}$ and sends R_i to the aggregator. The aggregator generates $\widetilde{R} \leftarrow \prod_{i=1}^n R_i$ from given $\{R_i\}_{i=1}^n$, and returns $\widetilde{R}$ to all the signers. Each signer $\mathcal{S}_i$ and the aggregator store $\tilde{h}_i = (r_i, \widetilde{R})$ and $z = \widetilde{R}$ as the helper information, respectively.

Sign$(pp, pk, sk, \tilde{h}, m) \to \sigma$**.** It chooses a value $t \leftarrow\$\ \{0,1\}^\kappa$ at uniformly random, computes $c \leftarrow H(\widetilde{R}, X, t, m)$ and $s \leftarrow cx + r \mod q$, then it outputs $\sigma = (s, t)$ as a signature.

Agg$(pp, z, \{(pk_i, m_i, \sigma_i)\}_{i=1}^n) \to \sigma_a$**.** It computes $\tilde{s} \leftarrow \sum_{i=1}^n s_i \mod q$, then it outputs the aggregate signature $\sigma_a = (\tilde{s}, \{t_i\}_{i=1}^n, \widetilde{R})$.

AggVer$(pp, \{(pk_i, m_i)\}_{i=1}^n, \sigma_a) \to \{0,1\}$**.** If $\{pk_i\}_{i=1}^n$ are not distinct to each other, it outputs 0. For all $i \in [1, n]$, it computes $c_i \leftarrow H(\widetilde{R}, X_i, t_i, m_i)$. If $\widetilde{R} = g^{\tilde{s}} \prod_{i=1}^n X_i^{-c_i}$ holds, then it outputs 1. Otherwise it outputs 0.

For the verification formula, it holds that $g^{\tilde{s}} \prod_{i=1}^n X_i^{-c_i} = g^{\sum_{i=1}^n x_i c_i + r_i} g^{-\sum_{i=1}^n x_i c_i} = g^{\sum_{i=1}^n r_i} = \widetilde{R}$. Thus, an aggregate signature is accepted with probability 1 when it is generated honestly.

Remark 4.2.1 *Note that already used helper information cannot be reused because the adversary can obtain two distinct signatures generated from the same helper information and extract a secret key by exploiting the special soundness property. Using a secure channel in the pre-communication is not necessary to guarantee the unforgeability in the security model in Section 4.1.1. This is because we suppose that any forger who can corrupt all cosigners and execute the pre-communication maliciously. Moreover, in the aggregation phase, if several signers fail to participate in this phase, the protocol terminates, and it should be restarted from pre-communication.*

4.2.2 The Security of PCAS

The random value t is crucial to prove our scheme secure based on the DL assumption in the RO model. Now we consider the scheme which is omitted t from the proposed scheme. In a security proof, a reduction needs to simulate an honest signer whose public key is Y. The intuitive way to simulate it is as follows: after receiving a PreCom signing query, the simulator first generated (R', s', c') by using the honest-verifier zero-knowledge (HVZK) property, and returns R' to a forger. Final of the pre-communication phase, the simulator obtains and stores cosigners' public keys and the information $\widetilde{R}$ and (R', s', c'). After that, it is given a message m' as a message signing query from the forger and needs to assign c' to $H(\widetilde{R}', Y, m')$ in the RO table. However, a forger can prevent this simulate from assigning c' by making a RO query $H(\widetilde{R}', Y, m')$ between the pre-communication phase and the signing phase because it can know all input $(\widetilde{R}', Y, m')$ of the hash function H before the simulator. To circumvent this difficulty, we added the fresh nonce t in the input of the hash function. For the case of PCAS, the simulator chooses the nonce t' in the signing phase and can assign c' to $H(\widetilde{R}', Y, t', m')$ in the RO table because a forger can no longer guess t' and cannot make the hash query $H(\widetilde{R}', Y, t', m')$ between the two phases.

Also, we use the KOSK model to compute the solution of the DL problem, namely the discrete logarithm of Y. Specifically, the reduction obtains two forgeries by rewinding and tries to extract the discrete log of Y by dividing those formulae satisfied such forgeries. But the terms related to cosigners become the obstacle to extract the solution because it does not know the cosigners' secret keys corresponding to the cosigners' public keys which are chosen freely by a forger. By using the KOSK model, the reduction can obtain the cosigners' secret key and compute the solution of the DL problem.

The following theorem states that PCAS is secure under the discrete logarithm assumption on the security model in Definition 4.1.2.

Theorem 4.2.1 *If there is a forger $\mathcal{F}$ that $(t, q_S, q_H, N, \varepsilon)$-breaks* PCAS *in the model of Definition 4.1.2, then there is an algorithm $\mathcal{B}$ that (t', ε')-breaks DL such that*

$$\varepsilon' \geq \frac{\varepsilon^2}{q_H + 1} - \frac{q_S(2q_H + q_S - 1)}{(q_H + 1)2^\kappa} - \frac{1}{q},$$
$$t' \leq 2t + 2q_S t_{exp} + O(q_H + q_S + 1),$$

where t_{exp} is the time for an exponentiation in G and we assume that $\kappa = \lambda$.

Proof 4.2.1 We first show the construction of the algorithm $\mathcal{B}$ which can solve the DL problem using the forger $\mathcal{F}$. $\mathcal{B}$ is given an instance of the DL problem Y and a parameter

(G, q, g).

To construct $\mathcal{B}$, let $\mathcal{A}$ be the algorithm as follows. On inputs (G, q, g, Y), $h_1, \ldots, h_{q_H+1} \in Z_q$, and a random tape ρ, $\mathcal{A}$ runs $\mathcal{F}$ on inputs (G, q, g) and Y as an honest signer's public key. It initializes counters $ctr_1 = 1, ctr_2 = 0$ and tables $T[\cdot]$, $L[\cdot]$ to be empty, where $T[\cdot]$ is a random oracle table and $L[\cdot]$ is a table that stores helper information of PreCom for signing queries. It responds to $\mathcal{F}$'s hash queries and signing queries as follows.

Hash Query $H(Q)$**.** A query Q is parsed as $Q = (\widetilde{R}, X, t, m)$. In the case where $X = Y$, $\mathcal{A}$ lets $T[Q] = h_{ctr_1}$ and $ctr_1 \leftarrow ctr_1 + 1$ if $T[Q]$ is undefined. In the case where $X \neq Y$, $\mathcal{A}$ lets $c \leftarrow\$\ Z_q, T[Q] \leftarrow c$ if $T[Q]$ is undefined. It returns $T[Q]$.

Signing Query. Firstly, when $\mathcal{A}$ receives the signal to start PreCom, it sets $ctr_2 \leftarrow ctr_2 + 1$, chooses $s', c' \leftarrow\$\ Z_q$, computes $R' \leftarrow g^{s'} Y^{-c'}$, and sends R' to $\mathcal{F}$. After that, when $\mathcal{A}$ is given $\widetilde{R}'$ from $\mathcal{F}$, $\mathcal{A}$ assigns $L[ctr_2] \leftarrow (s', c', \widetilde{R}')$.

When receiving a query (m', J), $\mathcal{A}$ sets $M' \leftarrow M' \cup \{m'\}$ and reads $L[J]$. It returns $\perp$ to $\mathcal{F}$ if $L[J]$ is empty. $\mathcal{A}$ chooses $t' \leftarrow\$\ \{0,1\}^\kappa$ and sets $Q' = (\widetilde{R}', Y, t', m')$. It sets $bad \leftarrow true$ and halts with output $\perp$ if $T[Q']$ is already defined. Otherwise it assigns $T[Q'] \leftarrow c'$, empties $L[J]$ and returns (s', t') to $\mathcal{F}$.

Finally, $\mathcal{F}$ outputs $\{X_i^*\}_{i=1}^{n^*}$ which is the set of public keys including Y, $\{x_i^*\}_{i \in [1,n^*] \setminus \{k\}}$ which is the set of secret keys corresponding to the public keys except X_k such that $Y = X_k$, the set of messages $\{m_i^*\}_{i=1}^{n^*}$, and a forgery $(\tilde{s}^*, \{t_i^*\}_{i=1}^{n^*}, \widetilde{R}^*)$. $\mathcal{A}$ checks whether $m_k^* \notin M'$ and $\mathbf{AggVer}(pp, \{(X_i^*, m_i)\}_{i=1}^{n^*}, (\tilde{s}^*, \{t_i^*\}_{i=1}^{n^*}, \widetilde{R}^*)) = 1$ holds, and it outputs $\perp$ if not. Otherwise $\mathcal{A}$ outputs $(I, \{(X_i, x_i)\}_{i \in [1,n] \setminus \{k\}}, (\tilde{s}^*, \{c_i^*\}_{i=1}^{n^*}, \widetilde{R}^*))$ where $c_i^* = T[\widetilde{R}^*, X_i, t_i^*, m_i^*]$ and I is the index such that $h_I = T[\widetilde{R}^*, Y, t_k^*, m_k^*]$.

$\mathcal{B}$ obtains the following two sequences by rewinding $\mathcal{A}$ according to the Bellare-Neven general forking lemma in Lemma 2.3.1.

$$
\begin{aligned}
&(I^{(1)}, \{(X_i^{(1)}, x_i^{(1)})\}_{i \in [1,n^{(1)}] \setminus \{k^{(1)}\}}, (\tilde{s}^{(1)}, \{c_i^{(1)}\}_{i=1}^{n^{(1)}}, \widetilde{R}^{(1)}))\\
&(I^{(2)}, \{(X_i^{(2)}, x_i^{(2)})\}_{i \in [1,n^{(2)}] \setminus \{k^{(2)}\}}, (\tilde{s}^{(2)}, \{c_i^{(2)}\}_{i=1}^{n^{(2)}}, \widetilde{R}^{(2)}))\\
&\quad \text{s.t. } \widetilde{R}^{(1)} = \widetilde{R}^{(2)} \wedge I^{(1)} = I^{(2)} \wedge c_{k^{(1)}}^{(1)} \neq c_{k^{(2)}}^{(2)}
\end{aligned}
$$

Since the above sequences satisfy the verification formula, we have

$$
\widetilde{R}^{(1)} = g^{\tilde{s}^{(1)}} \prod_{i=1}^{n^{(1)}} X_i^{(1)^{-c_i^{(1)}}} \text{ and } \widetilde{R}^{(2)} = g^{\tilde{s}^{(2)}} \prod_{i=1}^{n^{(2)}} X_i^{(2)^{-c_i^{(2)}}}.
$$

By $\widetilde{R}^{(1)} = \widetilde{R}^{(2)}$, dividing the above two equations gives

$$\begin{aligned} & Y^{c_k^{(1)} - c_k^{(2)}} \\ & = g^{\tilde{s}^{(1)} - \tilde{s}^{(2)}} \prod_{i \in [1, n^{(1)}] \setminus \{k^{(1)}\}} {X_i^{(1)}}^{-c_i^{(1)}} \prod_{i \in [1, n^{(2)}] \setminus \{k^{(2)}\}} {X_i^{(2)}}^{c_i^{(2)}}. \end{aligned}$$

Therefore, finally $\mathcal{B}$ outputs

$$\frac{\tilde{s}^{(1)} - \tilde{s}^{(2)} - \sum_{i \in [1, n^{(1)}] \setminus \{k^{(1)}\}} x_i^{(1)} c_i^{(1)} + \sum_{i \in [1, n^{(2)}] \setminus \{k^{(2)}\}} x_i^{(2)} c_i^{(2)}}{c_{k^{(1)}}^{(1)} - c_{k^{(2)}}^{(2)}} \bmod q \tag{4.1}$$

as the solution y to the instance Y of the DL problem.

$\mathcal{B}$ succeeds in outputting y if and only if it succeeds in forking $\mathcal{A}$. Let *frk* be the probability of succeeding in forking $\mathcal{A}$, and then the success probability ε' of $\mathcal{B}$ is equal to *frk*. Let *acc* be the probability that $\mathcal{A}$ outputs the sequence. We have

$$\begin{aligned} acc &= \Pr[bad \neq true \wedge \mathcal{F}succeed] \\ &\geq \Pr[\mathcal{F}succeed] - \Pr[bad = true]. \end{aligned}$$

The event $bad = true$ happens when $\mathcal{A}$ cannot set $H(\widetilde{R}', Y, t', m') \leftarrow c'$ in the random oracle table due to a predefined $H(\widetilde{R}', Y, t', m')$. $\mathcal{F}$ can cause this event by guessing t' which is the part of a signature that the signing oracle returns. How $\mathcal{F}$ maximizes the probability of causing this event is as follows. Firstly, for a hash query $Q = (\widetilde{R}', Y, t', m')$, $\mathcal{F}$ fixes $\widetilde{R}', Y$, and m' and queries q_H times with t' different from each other. After that, $\mathcal{F}$ makes q_S signing queries by using $\widetilde{R}'$ and m'. Let Hit_f be the event that $bad = true$ is happened in the fth time signing query. Note that one new row in the random oracle table is created every time $\mathcal{F}$ makes signing query. Then $\Pr[bad = true]$ is bounded as follows.

$$\begin{aligned} & \Pr[bad = true] \\ & = \Pr[\mathsf{Hit}_1 \vee \mathsf{Hit}_2 \vee \ldots \vee \mathsf{Hit}_{q_S}] \\ & \leq \Pr[\mathsf{Hit}_1] + \Pr[\mathsf{Hit}_2] + \ldots + \Pr[\mathsf{Hit}_{q_S}] \\ & \leq \frac{q_H}{2^\kappa} + \frac{q_H + 1}{2^\kappa} + \ldots + \frac{q_H + q_S - 1}{2^\kappa} \\ & = \frac{q_S(2q_H + q_S - 1)}{2^{\kappa+1}}. \end{aligned}$$

Thus we obtain

$$acc \geq \varepsilon - \frac{q_S(2q_H + q_S - 1)}{2^{\kappa+1}}.$$

By Lemma 2.3.1, we have

$$\begin{aligned}\varepsilon' = frk &\geq acc\left(\frac{acc}{q_H+1}-\frac{1}{q}\right) \geq \frac{acc^2}{q_H+1}-\frac{1}{q}\\ &= \frac{1}{q_H+1}\left(\varepsilon-\frac{q_S(2q_H+q_S-1)}{2^{\kappa+1}}\right)^2-\frac{1}{q}\\ &\geq \frac{\varepsilon^2}{q_H+1}-\frac{q_S(2q_H+q_S-1)}{(q_H+1)2^{\kappa}}-\frac{1}{q}.\end{aligned}$$

The running time t' of $\mathcal{B}$ is twice as the running time t of $\mathcal{F}$ plus $O(q_H+q_S+1)$ time needed to answer hash queries plus $2q_St_{exp}$ time because each signing query involves two exponentiation in G. □

The Restriction on the Public Keys. For PCAS, all signers' public keys need to be distinct from each other. The reason is as follows. In the security proof, if several cosigners are having the same public key as an honest signer, the denominator of Eq. (4.1) is $\sum_{i\in[1,n^{(1)}] \text{ s.t. } Y=X_i^{(1)}} c_i^{(1)} - \sum_{i\in[1,n^{(2)}] \text{ s.t. } Y=X_i^{(2)}} c_i^{(2)}$. In this situation, we cannot know whether this denominator is not equal to 0 only from condition $c_1^{(1)} \neq c_1^{(2)}$.

4.2.3 The Attack against PCAS with t in the Plain PK Model

Unfortunately, there is a sub-exponential time rogue-key attack against PCAS in the plain PK model. This forger is a universal forger under a key-only attack. In this sense, the KOSK model is essential for the security of PCAS.

We now show a forger $\mathcal{F}$ on input $pk = X_1$ as challenge key and a message m_1^* who succeeds in attacking PCAS in the plain PK model by using the k-sum algorithm. The goal of this forger is to output a forgery on m_1^* without making signing query. This will be done by making at most $k \cdot s_L$ hash queries, no signing query and running as follows.

1. $\mathcal{F}$ embeds X_1 in the public key pk_1. For $i \in [2,k]$, it chooses x_i such that $X_2 = X_1 g^{x_2}, \ldots, X_k = X_1 g^{x_k}$ which are distinct to each other, and it embeds X_i in the cosigner's public key pk_i. $\mathcal{F}$ chooses $\tilde{r} \leftarrow\$\ Z_q$ and compute $\widetilde{R} \leftarrow g^{\tilde{r}}$.
2. For each $i \in [1,k]$, $\mathcal{F}$ chooses m_i and s_L distinct nonces $\{t_{i,j}\}_{j=1}^{s_L}$, makes hash queries $(\widetilde{R}, X_i, t_{i,j}, m_i)$ for all $j \in [1,s_L]$, and obtains $\{c_{i,j}\}_{j=1}^{s_L}$ from the hash oracle. After that, it creates a list $L_i = \{c_{i,j}\}_{j=1}^{s_L}$.
3. $\mathcal{F}$ calls the k-sum algorithm with input $\{L_i\}_{i=1}^k$ and obtains $\{j_i\}_{i=1}^k$ such that $\sum_{i=1}^k c_{i,j_i} \equiv 0 \pmod q$.
4. $\mathcal{F}$ computes $\tilde{s} \leftarrow \tilde{r} + \sum_{i=2}^k c_{i,j_i} x_i \mod q$, then outputs the forgery $\sigma_a = (\tilde{s}, \widetilde{R}, \{t_{i,j_i}\}_{i=1}^k)$, the set of public keys $\{pk_i\}_{i=1}^k$, and the set of massages $\{m_i\}_{i=1}^k$.

For the forgery $\sigma_a = (\tilde{s}, \widetilde{R}, \{t_{i,j_i}\}_{i=1}^k)$ on $\{m_i\}_{i=1}^k$ where $c_{i,j_i} = H(\widetilde{R}, X_i, t_{i,j_i}, m_i)$, it holds that

$$\begin{aligned} g^{\tilde{s}} \prod_{i=1}^{k} X_i^{-c_{i,j_i}} &= g^{\tilde{r}+\sum_{i=2}^{k} c_{i,j_i} x_i} X_1^{-c_{1,j_1}} \prod_{i=2}^{k} (X_1 g^{x_i})^{-c_{i,j_i}} \\ &= g^{\tilde{r}+\sum_{i=2}^{k} c_{i,j_i} x_i} X_1^{-\sum_{i=1}^{k} c_{i,j_i}} g^{-\sum_{i=2}^{k} c_{i,j_i} x_i} \\ &= g^{\tilde{r}} = \widetilde{R}. \end{aligned}$$

The third equality holds because it holds that $\sum_{i=1}^{k} c_{i,j_i} \equiv 0 \pmod{q}$. Hence the above attack is valid for PCAS in the plain PK model. Note that, in the KOSK model, $\mathcal{F}$ must output the secret keys corresponding to $\{pk_i\}_{i=2}^k$, namely, it must extract the discrete logarithms of $\{X_1 g^{x_i}\}_{i=2}^k$. Because it is hard to do these extractions efficiently in general, this attack dose not work well in the KOSK model.

4.2.4 On the KOSK Model and Its Implementation

We used the KOSK model in the security proof for simplicity and necessity. In practice, a possible way to implement the KOSK model is to use a proof-of-possession (PoP). The security of this implementation depends on the security of PoP. For example, we may consider the case of using the Schnorr signature [Sch89] as PoP. More specifically, if a signer is required to include the PoP signed by his secret key in his public key, then, in the security game, a forger outputs the PoP signed by the secret keys behind the cosigners' public keys, not the secret keys. Since the set of the cosigners' secret keys is necessary for the proof of Theorem 4.2.1, proving the security of PCAS with this PoP is not trivial. A possible way to prove such a scheme secure is applying Bagherzandi-Cheon-Jarecki generalized forking lemma [BCJ08].

4.3 Performance Comparison among Aggregate Signature Scheme and Related Schemes

In this section, we compare the proposed scheme PCAS with the Zhao's aggregate signature scheme [Zha19], the Bellare-Neven interactive aggregate signature scheme BN-IAS [BN06] and the concatenation of the individual Schnorr signatures [Sch89].[2] These schemes are constructed based on the Schnorr signature scheme [Sch89]. Note that we

[2]This scheme has no communication between signers for signing, namely there is no open signing query. This means that Drijvers et al.'s impossibility cannot be applied to this scheme.

suppose the situation where these schemes are used for the same purpose of compressing signatures on different messages into a compact signature. Then, we focus on sharing messages, communication complexity, computational complexity, withdrawal, the key setup model, assumptions, and the size of the aggregate signature for the comparison. Table 4.1 summarizes this comparison.

Necessity of Sharing Messages. On the above purpose, BN-IAS is the multi-signature scheme used as an aggregate signature scheme. More detail, this scheme generates a combined signature on different messages by seeing a set of signers' messages as one message. Then, all players need to execute the interactive protocols before signing phase.

PCAS, Zhao's scheme, and the concatenation of Schnorr signatures need not share messages between all signers. Especially, PCAS requires interactive protocol as PreCom, however, this interaction is executed in Stage I. Thus, it achieves no sharing messages. Zhao's scheme and the concatenation of Schnorr signatures have no interaction protocol.

Communication Complexity. Firstly, let n be the number of signers, and $|M|$ be the size of a message M. Moreover, we consider the communication complexity including the cost of one-shot communication from signers to an aggregator for submitting a signature.

For BN-IAS, all signers need to share messages before the signing phase. Also, the signing protocol requires three-round communication between every two signers. Therefore, this scheme requires $n(n-1)/2$ channels, and the total communication complexity per channel is $2|M| + 2l_0 + 2|G| + 2|Z_q|$ where l_0 is the bit-length of the range of the hash function to produce the commitment to commitment element on the Schnorr signature scheme. The total communication complexity in aggregation protocol is $n(n-1)(|M| + l_0 + |G| + |Z_q|)$.

PCAS needs one bidirectional communication to share helper information between signers and the aggregator in the pre-communication phase. Hence this scheme requires n channels, and the total communication complexity per channel is $2|G| + |Z_q| + \kappa$. Then the total communication complexity is $n(2|G| + |Z_q| + \kappa)$.

Zhao's scheme has no interactive protocol, so there are only communications for submitting signatures. Then this scheme requires n channels, and the total communication complexity is $2n|Z_q|$.

In the case of the concatenation of Schnorr signatures, there are only communications for submitting individual signatures. Thus, it requires n channels, and the total communication complexity is $2n|Z_q|$ because a Schnorr signature consists of two elements in Z_q.

The Size of an Aggregate Signature. The size of a signature of BN-IAS is $|Z_q| + |G|$,

and hence it is independent of n. The signature size of PCAS is $|Z_q| + |G| + n\kappa$ and the signature size of Zhao's scheme is $|Z_q| + n|G|$. The size of the concatenation of Schnorr signatures is $2n|Z_q|$.

On Asymptotically Linear Signature Sizes of Ours and Zhao Scheme's. Both ours and Zhao scheme's signature sizes depend on n linearly. However, note that both schemes achieve smaller signature sizes than the size of the concatenation of Schnorr signatures. Moreover, for the parameter of PCAS, we can pick κ to be equal to the security parameter λ because we only needed to consider only target collisions for the hash function in the proof of Theorem 4.2.1. Also, we can have $\kappa \leq |G|$ because the order of G is about $2^{2\lambda}$ in general. Therefore PCAS can achieve a smaller signature size than Zhao's scheme.

Computational Complexity. Let t_{exp} be the time for an exponentiation in G and let be t_h be the time for the generation of one hash value.

Each signer of BN-IAS needs $t_{exp}+(n+1)t_h$ time for signing, and it takes $(n+1)t_{exp}+nt_h$ time to verify. For Zhao's scheme, each signer requires $t_{exp} + 2t_h$ time for signing, and a verifier requires $(2n + 1)t_{exp} + 2nt_h$ time for the verification. PCAS needs $t_{exp} + t_h$ time for signing per one signer and $(n + 1)t_{exp} + nt_h$ time for the verification. For the Schnorr signature scheme, a signer requires $t_{exp}+t_h$ time for signing, and a verifier requires $2t_{exp} + t_h$ time. Therefore, for the verification of the concatenation of Schnorr signatures, it requires $n(2t_{exp} + t_h)$ time.

Key Setup Model, Assumptions, and Acceptable Condition. We proved PCAS secure under the DL assumption in the KOSK model and the random oracle model. This scheme needs to use a PoP to prove the correct generation of a public key in practical due to the KOSK model. BN-IAS was proved secure under the DL assumption in the random oracle model and the plain PK model. Zhao's scheme was proved secure under the NMDL assumption in the random oracle model and the plain PK model. Zhao also showed the hardness of the NMDL problem in the generic group model [Mau05] and the random oracle model. The Schnorr signature scheme was proved secure under the DL assumption in the random oracle model. Also, rogue-key attacks are ineffective for the concatenation of Schnorr signatures because the verification of it executes individual verification for all individual signatures.

BN-IAS has no restrictions on public keys and messages, namely, we may include duplicate public keys and messages in aggregation. In Zhao's scheme, an aggregate signature is not accepted when all signers' pairs of a public key and a message are not distinct to each other. In PCAS, the aggregate signature can be accepted at least every public keys should be distinct.

Withdrawal. BN-IAS must halt and restart a signing protocol when some signers disappear in the signing phase. Also, PCAS must halt and restart a signing protocol when some signers fail to participate in an aggregation phase. On the other hand, Zhao's scheme and the concatenation of Schnorr signatures can continue the process in such a situation because each signer generates a signature without any communication.

Table 4.1: Performance Comparison among Aggregate Signature Scheme and Related Schemes

Scheme	BN-IAS [BN06]	Zhao [Zha19]	PCAS	Schnorr [Sch89]
Type	IAS	standard AS	AS with PreCom	concatenation of individual signatures
No sharing Messages	No	Yes	Yes	Yes
Communication Complexity	$(\|M\| + l_0 + \|G\| + \|Z_q\|) \times n(n-1)$	$2n\|Z_q\|$	$n(2\|G\| + \|Z_q\| + \kappa)$	$2n\|Z_q\|$
Signature Size	$\|Z_q\| + \|G\|$	$\|Z_q\| + n\|G\|$	$\|Z_q\| + \|G\| + n\kappa$	$2n\|Z_q\|$
Computational Complexity for Signing	$t_{exp} + (n+1)t_h$	$t_{exp} + 2t_h$	$t_{exp} + t_h$	$t_{exp} + t_h$
Computational Complexity for Verifying	$(n+1)t_{exp} + nt_h$	$(2n+1)t_{exp} + 2nt_h$	$(n+1)t_{exp} + nt_h$	$n(2t_{exp} + t_h)$
Assumption	DL	NMDL	DL	DL
Key Setup	plain PK	plain PK	KOSK	plain PK
Restriction in Aggregation	No Restriction	Distinct (pk,m)	Distinct pk	No Restriction
Withdrawal	No	Yes	No	Yes

* The row 1 and 2 indicate pairing-free aggregate signature (AS) scheme and related schemes. In row 4 and 5, $|M|$, $|Z_q|$ and $|G|$ indicate the size of a element in $|M|$, Z_q, and G. Also. n denotes the number of signers, and ℓ_0 and κ are specific parameters on each scheme. Especially, the bit-length of κ is as same as the security parameter in general. The row 6 and 7 show the computational complexity for signing and verifying focused on the time for the exponential in G and the calculation of hash function. The row 8 and 9 show that the assumption and the key-setup model (cf., the notion of models in Section 2.4.1) in which each scheme is proved secure, where DL and NMDL indicate the discrete logarithm assumption and non-malleable DL assumption [Zha19]. The row 10 shows the restriction of all signers' public keys and/or messages to be accepted in the verification. The final row shows the possibility of a continuation of the procedure in the case where signers disappear before the aggregation phase.

Trade-offs among the Four Schemes. We conclude this comparison by saying that there are trade-offs among the four schemes. BN-IAS achieves the security based on the DL assumption in the plain PK model and the constant signature size. Zhao's scheme achieves non-interactive aggregation and is proved secure in the plain PK model. PCAS achieves the security is based on the DL assumption and a smaller communication complexity than BN-IAS. The concatenation of the Schnorr signatures achieves the security based on the DL assumption, and in this case, rogue-key attacks are not effective. As described above, each scheme has different strong and weak points. Thus it is significant for us to choose

a suitable scheme for an application, and our scheme PCAS provides a new candidate. For example, for applications where we want no interaction or withdrawal, we should use Zhao's scheme or the Schnorr signature scheme. For applications where we can allow communications and want the security on the well-established assumption, we can use BN-IAS or PCAS. Especially, BN-IAS and PCAS are suitable if we attach importance to efficiency in terms of the signature size and the communication complexity, respectively.

4.4 Circumventing of Drijvers et al.'s Impossibility

In this section, we discuss how we avoided Drijvers et al.'s attacks and impossibility results [DEF+19]. The key to avoiding their attacks and impossibility, though the two-round protocol is implemented as PCAS, is the use of random value t in the input to the hash function. We explain this from two viewpoints. The one is that if t is omitted, we *have* an attack by adopting Benhamouda et al.'s attack against CoSi [BLOR20], which is an improvement of Drijvers et al.'s attack against the same scheme, but the existence of t prevents such an attack (cf., Theorem 4.2.1 in Section 4.2.2). The other is that we cannot extend the meta-reduction arguments to PCAS.

Review on Drijvers et al.'s and Benhamouda et al.'s Attacks. Here we review Drijvers et al.'s and Benhamouda et al.'s attacks. While Drijvers et al.'s attacks use a k-sum solver, Benhamouda et al.'s attack uses an ROS solver. A k-sum solver runs in sub-exponential time, and their ROS solver runs in polynomial time. This improvement in time complexity is obtained by allowing more degrees of freedom in the solution of the problem. Given an integer c^* and a list of integers, a k-sum solver can find integers $\{c_i\}_{i=0}^{\ell-1}$ from the given list satisfying

$$\sum_{i=0}^{\ell-1} c_i = c^*.$$

In contrast, given a list of integers, their ROS solver can find integers $\{a_i\}_{i=0}^{\ell-1}$ which enables us, given an integer c^*, to find integers $\{c_i\}_{i=0}^{\ell-1}$ from the given list satisfying

$$\sum_{i=0}^{\ell-1} a_i c_i = c^*.$$

Here, their ROS solver chooses $\{a_i\}_{i=0}^{\ell-1}$ even *from the outside of the set*. This freedom in the choice of $\{a_i\}_{i=0}^{\ell-1}$ enables us to construct a polynomial-time ROS solver.

Note that in the ROS problem, c^* is determined after $\{a_i\}_{i=0}^{\ell-1}$ are chosen. This dependence between variables complicates the construction of the solver. While this dependence

among variables is one of the most important techniques of Benhamouda et al.'s ROS solver [BLOR20], however, their attacks were presented in a monolithic manner. Contrasting, for the ease of understanding, we modularize their ROS solver and present our attack against PCAS without t on top of this modularized ROS solver. This modularized presentation can be of independent interest.

4.4.1 How to Break PCAS *without* t in the KOSK Model

We have already explained the necessity of the random value t in terms of the security proof in Section 4.2.2. Here, we show it again in terms of an attack by constructing a universal forger who succeeds in attacking PCAS *without* t in the KOSK model.

Recently, Benhamouda et al. presented an algorithm which can solve the ROS (Random inhomogeneities in a Overdetermined Solvable system of linear equations) problem [Sch01] in polynomial time *for large enough dimensions*, and they proposed polynomial-time forgers against Schnorr blind signature scheme, a multi-signature scheme CoSi [STV+16], and so on [BLOR20]. The construction of PCAS without t is very similar to CoSi. Therefore, it is natural for us to apply the ROS attack to PCAS without t, and we will confirm that it is possible. Below, we will present a concrete procedure of the polynomial-time forgery in the case where the number of parallel signing sessions is larger than $\log q$. This number of sessions is essential for the attack.

Our attack employs as a subroutine the attack against the ROS assumption by Benhamouda et al. In addition, our attack is in the KOSK model, and corrupts one cosigner. These are similar to the attack against CoSi.

To motivate the necessity of the ROS attack, we first explain an overview of our attack against PCAS without t.

We first specify the goal of this attack. Let $\mathcal{F}$ be a forger with input X_0 as a challenge key, and let X_1 be the cosigner's public key chosen by $\mathcal{F}$. $\mathcal{F}$ tries to output $(\widetilde{R}^*, \widetilde{s}^*)$, (m_0^*, m_1^*), and x_1 s.t. $X_1 = g^{x_1}$ and $\widetilde{R}^* = g^{\widetilde{s}^*} X_0^{-c_0^*} X_1^{-c_1^*}$, which is the verification equation, where $c_0^* = H(\widetilde{R}^*, X_0, m_0^*)$ and $c_1^* = H(\widetilde{R}^*, X_1, m_1^*)$. Because $\mathcal{F}$ must open the cosigner's secret key x_1 due to the KOSK model, it generates X_1 honestly. Then, notice that, if $\mathcal{F}$ can obtain $(\hat{s}, \widetilde{R}^*, m_0^*)$ s.t.

$$\hat{s} \equiv \log_g \widetilde{R}^* + c_0^* \log_g X_0 \pmod{q}, \tag{4.2}$$

it can compute $\widetilde{s}^* \leftarrow \hat{s} + c_1^* x_1$ and succeed in forgery. Thus, $\mathcal{F}$ aims at obtaining $(\hat{s}, \widetilde{R}^*, m_0^*)$ s.t. Eq. (4.2).

Here, we review the process of the signing query. In the pre-communication phase, first

$\mathcal{F}$ receives R' from the signing oracle Σ and sends helper information $\widetilde{R}'$ to Σ. In the signing phase, $\mathcal{F}$ sends m' as a query to Σ and finally obtains a valid signature s' from Σ s.t.

$$s' \equiv \log_g R' + c' \log_g X_0 \pmod{q} \tag{4.3}$$

where $c' = H(\widetilde{R}', X_0, m')$. Note that, after receiving R' from Σ, $\mathcal{F}$ can know a value c' by making a hash query $(\widetilde{R}', X_0, m')$ before sending $\widetilde{R}'$ to Σ in the pre-communication phase, and it can *force* Σ to use c' of $\mathcal{F}$'s choice through m' and $\widetilde{R}'$.[3]

In this attack, $\mathcal{F}$ executes ℓ signing queries in parallel. Let m'_i be a message queried in ith message signing queries, and let s'_i and R'_i be responses of ith signing query. For all $i \in [0, \ell-1]$, these satisfy

$$s'_i \equiv \log_g R'_i + c'_i \log_g X_0 \pmod{q} \tag{4.4}$$

where $c'_i = H(\widetilde{R}'_i, X_0, m'_i)$. From this equation, if $\mathcal{F}$ can find $\widetilde{R}^*$ and m_0^* that have some $a_0, \ldots, a_{\ell-1}$ satisfying

$$(i) \quad \widetilde{R}^* = \prod_{i=0}^{\ell-1} R_i'^{a_i} \quad \text{and}$$

$$(ii) \quad H(\widetilde{R}^*, X_0, m_0^*) = \sum_{i=0}^{\ell-1} a_i H(\widetilde{R}'_i, X_0, m'_i),$$

then by computing $\hat{s} \leftarrow \sum_{i=0}^{\ell-1} a_i s'_i \mod q$, $\mathcal{F}$ can make Eq. (4.2) hold. Thus, to obtain $(\hat{s}, \widetilde{R}^*, m_0^*)$ satisfying Eq. (4.2), it is sufficient for $\mathcal{F}$ to make both (i) and (ii) hold.

First, $\mathcal{F}$ makes the set of pairs of two distinct elements $\{(c_i'^{(0)}, c_i'^{(1)})\}_{i=0}^{\ell-1}$ by making hash queries $c_i'^{(0)} = H(\widetilde{R}_i'^{(0)}, X_0, m'_i)$ and $c_i'^{(1)} = H(\widetilde{R}_i'^{(1)}, X_0, m'_i)$ for each $i \in [0, \ell-1]$. Second, it generates $\{a_i\}_{i=0}^{\ell-1}$ satisfying the following property (P):

(P): for all $c^* \in Z_q$, one can efficiently find a set of bits $\{b_i\}_{i=0}^{\ell-1}$ such that

$$\sum_{\ell=0}^{\ell-1} a_i c_i^{(b_i)} \equiv c^* \pmod{q}. \tag{4.5}$$

$\mathcal{F}$ initiates ℓ signing queries and receives $\{R'_i\}_{i=0}^{\ell-1}$ from Σ in the pre-communication phase. Third, $\mathcal{F}$ can makes (i) hold by computing $\widetilde{R}^* \leftarrow \prod_{i=0}^{\ell-1} R_i'^{a_i}$. $\mathcal{F}$ obtains $c_0^* = H(\widetilde{R}^*, X_0, m_0^*)$ by querying the hash oracle. Exploiting the property (P), it obtains $\{b_i\}_{i=0}^{\ell-1}$ such that $c_0^* \equiv \sum_{i=0}^{\ell-1} a_i c_i'^{(b_i)} \pmod{q}$, and then (ii) holds. Finally, $\mathcal{F}$ forces Σ to use $c_i'^{(b_i)}$

[3]Here, we exploit open signing queries.

by sending $\widetilde{R}_i'^{(b_i)}$, and obtains $\{s_i'\}_{i=0}^{\ell-1}$ such that $s_i' \equiv \log_g R_i' + c_i'^{(b_i)} \log_g X_0 \pmod q$ for $i \in [0, \ell-1]$. The result shows that $\mathcal{F}$ obtains $\hat{s} \leftarrow \sum_{i=0}^{\ell-1} a_i s_i' \mod q$ satisfying Eq. (4.2). Therefore, $\mathcal{F}$ succeeds in forgery in the KOSK model.

To find $\{a_i\}_{i=0}^{\ell-1}$ satisfying the property (P), we exploit the technique of the ROS attack as follows. For $\ell > \log q$, when Benhamouda et al.'s ROS solver is given the set of pairs of two distinct elements in Z_q $\{(c_i^{(0)}, c_i^{(1)})\}_{i=0}^{\ell-1}$, it can obtain $\{a_i\}_{i=0}^{\ell-1}$ satisfying the property (P).

Below, we show the concrete procedure of $\mathcal{F}$ on input a challenge key X_0, making ℓ signing queries in parallel and $2(\ell+1)$ hash queries.

1. $\mathcal{F}$ chooses $\{m_i'\}_{i=0}^{\ell-1}$ used for message signing queries.
2. For each $i \in [0, \ell-1]$, $\mathcal{F}$ chooses distinct $\widetilde{R}_i'^{(0)}$ and $\widetilde{R}_i'^{(1)}$, makes hash queries $(\widetilde{R}_i'^{(0)}, X_0, m_i')$ and $(\widetilde{R}_i'^{(1)}, X_0, m_i')$, and obtains $c_i'^{(0)}$ and $c_i'^{(1)}$. If $c_i'^{(0)} = c_i'^{(1)}$ holds, it starts over from choosing $\widetilde{R}_i'^{(0)}$ and $\widetilde{R}_i'^{(1)}$.
3. $\mathcal{F}$ executes the subroutine $\mathsf{ROS}_1(\{(c_i'^{(0)}, c_i'^{(1)})\}_{i=0}^{\ell-1})$ and obtains $\{a_i\}_{i=0}^{\ell-1}$.
4. $\mathcal{F}$ initiates ℓ signing queries *in parallel*. Then it receives $\{R_i'\}_{i=0}^{\ell-1}$ from the oracle Σ and sets $\widetilde{R}^* \leftarrow \prod_{i=0}^{\ell-1} R_i'^{a_i}$. Note that Eq. (i) holds.
5. $\mathcal{F}$ chooses any m_0^* used as a forged message and obtains c_0^* by making a hash query $(\widetilde{R}^*, X_0, m_0^*)$. Note that $\mathcal{F}$ is a universal forger. It executes the subroutine $\mathsf{ROS}_2(\{(c_i'^{(0)}, c_i'^{(1)})\}_{i=0}^{\ell-1}, \{a_i\}_{i=0}^{\ell-1}, c_0^*)$ and obtains $\{b_i\}_{i=0}^{\ell-1}$. From Eq. (4.5), $\mathcal{F}$ can obtains $\{\widetilde{R}_i'^{(b_i)}\}_{i=0}^{\ell-1}$ which makes Eq. (ii) hold.
6. For each $i \in [0, \ell-1]$, $\mathcal{F}$ gives $\widetilde{R}_i'^{(b_i)}$ and m_i' to Σ and obtains the valid signature s_i' satisfying $s_i' \equiv \log_g R_i' + c_i'^{(b_i)} \log_g X_0 \pmod q$. Then, it sets $\hat{s} \leftarrow \sum_{i=0}^{\ell-1} a_i s_i' \mod q$ and obtains $\hat{s}$ satisfying Eq. (4.2).
7. $\mathcal{F}$ chooses cosigner's secret key x_1 and computes $X_1 \leftarrow g^{x_1}$. $\mathcal{F}$ chooses a message m_1^* and obtains c_1^* by making the hash query $(\widetilde{R}^*, X_1, m_1^*)$. It computes $s^* \leftarrow \hat{s} + c_1^* x_1 \mod q$, and then outputs the forgery $\sigma_a = (s^*, \widetilde{R}^*)$, the set of keys $\{(X_0, \perp), (X_1, x_1)\}$, and the set of massages $\{m_0^*, m_1^*\}$.

The ROS solver by Benhamouda et al. [BLOR20] is consisted of the two subroutines ROS_1 and ROS_2 with the review based on our modular treatments as follows.

ROS_1 :

Input $\{(c_i^{(0)}, c_i^{(1)})\}_{i=0}^{\ell-1}$ s.t. $c_i^{(0)}, c_i^{(1)} \in Z_q$, $c_i^{(0)} \neq c_i^{(1)}$ for $i \in [0, \ell-1]$.

Step 1 For $i \in [0, \ell-1]$, compute $a_i \leftarrow \frac{2^i}{c_i^{(1)} - c_i^{(0)}} \mod q$.

Step 2 Output $\{a_i\}_{i=0}^{\ell-1}$.

ROS_2 :

Input $\{(c_i^{(0)}, c_i^{(1)})\}_{i=0}^{\ell-1}$ s.t. $c_i^{(0)}, c_i^{(1)} \in Z_q$, $c_i^{(0)} \neq c_i^{(1)}$ for $i \in [0, \ell-1]$, $\{a_i\}_{i=0}^{\ell-1}$, and c^*.

Step 1 Compute $c \leftarrow c^* - \sum_{i=0}^{\ell-1} a_i c_i^{(0)} \mod q$.

Step 2 Write c in binary as $c = \sum_{i=0}^{\ell-1} 2^i b_i$.

Step 3 Output $\{b_i\}_{i=0}^{\ell-1}$.

Correctness The output $\{b_i\}_{i=0}^{\ell-1}$ of the subroutine ROS_2 satisfies the following equation because of Step 2 of ROS_2:

$$\sum_{i=0}^{\ell-1} 2^i b_i = c.$$

Note that $\frac{c_i^{(b_i)} - c_i^{(0)}}{c_i^{(1)} - c_i^{(0)}}$ is equal to 0 or 1 if b_i is equal to 0 or 1, respectively. By apply this to the above equation and since $c \equiv c^* - \sum_{i=0}^{\ell-1} a_i c_i^{(0)} \pmod q$, we obtain the following equations:

$$\sum_{i=0}^{\ell-1} 2^i \frac{c_i^{(b_i)} - c_i^{(0)}}{c_i^{(1)} - c_i^{(0)}} \equiv c^* - \sum_{i=0}^{\ell-1} a_i c_i^{(0)} \pmod q.$$

By the definition of a_i, we obtain the following equation:

$$\sum_{i=0}^{\ell-1} a_i c_i^{(b_i)} - \sum_{i=0}^{\ell-1} a_i c_i^{(0)} \equiv c^* - \sum_{i=0}^{\ell-1} a_i c_i^{(0)} \pmod q.$$

By deleting the same term in both sides, we obtain

$$\sum_{i=0}^{\ell-1} a_i c_i^{(b_i)} \equiv c^* \pmod q.$$

Note that PCAS with t is secure against the above forger F. Let us consider applying $\mathcal{F}$ to PCAS. In signing queries of this case, Σ returns (s'_i, t'_i) satisfying $s'_i \equiv \log_g R'_i + c_i'^{(b_i)} \log_g X_0 \pmod q$ where $c_i'^{(b_i)} = H(\widetilde{R}_i'^{(b_i)}, X_0, t'_i, m'_i)$. Thus, $\mathcal{F}$ needs to make the hash queries $(\widetilde{R}_i'^{(b_i)}, X_0, t'_i, m'_i)$ in Step 2 of the attack procedure. However, $\mathcal{F}$ cannot do this Step because it obtains t'_i in Step 6.

4.4.2 Why Meta-Reduction Does Not Work?

In this section, we will explain simply why PCAS can avoid the Drijvers et al. ' s impossibility from the term of the meta-reduction arguments. For more details, see Appendix A.

Meta-reduction Arguments of Drijvers et al.'s Impossibility Result

Drijvers et al. proved that two-round MS schemes, e.g., CoSi [STV+16], BCJ [BCJ08], MWLD [MWLD10] and MuSig [MPSW18], cannot be proved secure under the OMDL assumption [PV05] even in the KOSK model. Specifically, they proved that there are the meta-reductions who can solve the OMDL problem if there are a reductions who proves the above schemes to be unforgeable in the KOSK model. In this proof, the meta-reduction needs to solve the OMDL problem by running a black-box reduction and simulate a forger in order to run such a reduction. The meta-reductions achieves it by exploiting the open signing queries.

More concretely, the meta-reductions basically simulate a forger by using the DL oracle. However, there are situations where a reduction force a forger to generate different forgeries on the same message and the same commitment element of Σ-protocol [Cra96], e.g., the situation where a reduction rewinds a forger. In such situations, if the meta-reduction accesses the DL oracle for simulating a forger, it cannot solve the OMDL problem because the number of times to access to the DL oracle is a deficiency. To avoid this, the meta-reduction exploits the open signing queries to extract a secret key corresponding to a challenge key given to a forger.

Here, for simplicity, we suppose a black-box reduction $\mathcal{R}$ who proves PCAS without t secure under the OMDL assumption in the KOSK model. Also, we assume that such situations are caused by $\mathcal{R}$ who rewinds a forger $\mathcal{F}$. Then, the meta-reduction $\mathcal{M}$ designs $\mathcal{F}$ on input a challenge key X who executes signing queries as follows.

1. $\mathcal{F}$ initiates the pre-communication protocol by sending the signal to start PreCom. $\mathcal{F}$ acts as a malicious aggregator and receives R from signing oracle simulated by $\mathcal{R}$.
2. $\mathcal{F}$ makes two random oracle queries $H(\widetilde{R}_0, X, m)$ and $H(\widetilde{R}_1, X, m)$ where $\widetilde{R}_0 \neq \widetilde{R}_1$ and m is a message used as a message signing query. It checks $H(\widetilde{R}_0, X, m) = H(\widetilde{R}_0, X, m)$. If it holds, it redoes this step.
3. $\mathcal{F}$ makes a random oracle query $H(\widetilde{R}*, X, m^*)$ where $\widetilde{R}^*$ is a part of a forgery and m^* is a message corresponding to a forgery.
4. $\mathcal{F}$ chooses $b \leftarrow \{0, 1\}$ and sends $\widetilde{R}_b$ as the helper information.
5. $\mathcal{F}$ sends m as the message signing query and receives an individual signature s where the tuple $(R, H(\widetilde{R}_b, X, m), s)$ forms a valid transcript of the Schnorr protocol.

In this case, the rewind point is Step 3. Then, $\mathcal{M}$ can obtain two different transcripts $(R, H(\widetilde{R}_0, X, m), s_0)$ and $(R, H(\widetilde{R}_1, X, m), s_1)$ on the same commitment element R by choosing different b before and after rewinding. From these transcripts, $\mathcal{M}$ can extract

the secret key corresponding to the challenge key by the special soundness property.

Inapplicability of Meta-reduction Arguments to PCAS

Now we simply explain why the meta-reduction argument is inapplicable to PCAS. For PCAS, the different point from the case of PCAS without t is that $\mathcal{R}$ returns (s,t) s.t. $R = g^s X^{-H(\widetilde{R}_b,X,t,m)}$. Then, to extract the secret key corresponding to the challenge key, in Step 2, $\mathcal{M}$ needs to guess the random value t chosen by $\mathcal{R}$ in Step 5. However, it can no longer guess t. Therefore, the meta-reduction argument mentioned in the previous section cannot be applied to PCAS. In terms of the reduction, the reduction can generate only one transcript related to $b = 0$ or $b = 1$ in the case of PCAS without t. But the reduction of PCAS can submit the answer to signing queries regardless of the helper information $\widetilde{R}_0$ or $\widetilde{R}_1$ because of the random value t. Specifically, the reduction generates a tuple (R, c, s), where c is the value assigned in the random oracle table, by using the honest-verifier zero-knowledge property in Step 1. It needs to assign c to $H(\widetilde{R}_b, X, t, m)$ in the random oracle table in Step 5. $H(\widetilde{R}_b, X, t, m)$ is undefined at the time of Step 5 with the overwhelming probability because it is hard for the forger simulated by meta-reduction to guess t. Therefore, by generating t freshly in Step 5, the reduction can assign the same c before and after rewinding.

Chapter 5

One-Time Aggregate Signature

Bellare and Shoup proposed a one-time signature scheme [BS07]. This signature scheme is a variant of the Schnorr signature scheme. More concretely, a commitment R is sent to a verifier by including it to the public key.

In this chapter, we propose a one-time aggregate signature (OTAS) scheme. In the protocol of the OTAS, all signers are required generating new key pairs each time they generate a new aggregate signature. Our proposed one-time aggregate signature scheme OTAS is an extension of the Bellare-Shoup one-time signature scheme. Our scheme achieves non-interactive aggregation. We prove this scheme secure based on the one-more discrete logarithm (OMDL) assumption [PV05] and the existence of a collision-resistant hash function in the KOSK model and the standard model, which does not require the random oracle.

Moreover, the Bellare-Shoup one-time signature [BS07] is proposed as two-tier signatures. The keys of two-tier signatures consist of a primary key and a secondary key. A signer can reuse a primary key, but it must regenerate a secondary key every time it generates a signature. OTAS can be executed like two-tier signatures. In that case we call it two-tier aggregate signatures. For simplicity, we propose OTAS as one-time aggregate signatures.

Recently, Boneh and Kim proposed an OTAS scheme [BK20]. This scheme is built from the Bellare-Shoup one-time signature scheme and achieves the security based on the DL assumption in the plain PK model and the random oracle model. However, there is a very large reduction loss, e.g., $1/|M|$ where $|M|$ is the size of the set of messages. In contrast, our proposed scheme achieves the tight security though this is in the KOSK model.

5.1 Proposed Scheme **OTAS** and Security Proof

The definition is described in Section 2.4.2. Notably, the algorithms constructed OTAS is as same as conventional aggregate signatures. Below we show proposed OTAS scheme.

5.1.1 The Algorithms of OTAS

OTAS consists of the following algorithms.

Setup$(1^\lambda) \to pp$**.** It chooses (G, q, g) and a hash key $K \leftarrow\$ \{0,1\}^\lambda$, then outputs $pp = (G, q, g, K)$.

KeyGen$(pp) \to (pk, sk)$**.** It chooses $x \leftarrow\$ Z_q$ and $r \leftarrow\$ Z_q$, computes $X \leftarrow g^x$ and $R \leftarrow g^r$, and outputs the public key $pk = (X, R)$, then the secret key $sk = (x, r)$.

Sign$(pp, pk, sk, m) \to \sigma$**.** It computes $c \leftarrow H(K, R, X, m)$ and $s \leftarrow cx + r \mod q$, then outputs $\sigma = s$ as a signature.

Agg$(pp, \{(pk_i, m_i, \sigma_i)\}_{i=1}^n, j) \to \sigma_a$**.** It computes $\tilde{s}_a \leftarrow \sum_{i=1}^n s_i \mod q$, then outputs $\sigma_a = \tilde{s}_a$ as an aggregate signature.

AggVer$(pp, \{(pk_i, m_i)\}_{i=1}^n, \sigma_a) \to \{0,1\}$**.** If $\{pk_i\}$ are not distinct to each other, it outputs 0. For $i = 1, \ldots, n$, it computes $c_i \leftarrow H(K, R_i, X_i, m_i)$. If $\prod_{i=1}^n R_i = g^{\tilde{s}_a} \prod_{i=1}^n X_i^{-c_i}$ holds, it outputs 1. Otherwise, it outputs 0.

5.1.2 The Security of OTAS

We explain the security model of OTAS in Section 2.4.2. Note that a forger can make only one signing query in that model. OTAS is unforgeable under the OMDL assumption and the existence of a collision-resistant hash function in the KOSK model.

Theorem 5.1.1 *If there is a forger $\mathcal{F}$ that (t, N, ε)-breaks* OTAS*, then there are an algorithm $\mathcal{B}$ that (t', ε')-breaks 2-OMDL and an algorithm $\mathcal{C}$ that breaks the collision resistance of H such that*

$$\varepsilon \leq \varepsilon' + \mathrm{Adv}_H(\mathcal{C}), \qquad t' \geq t + t_{exp} + O(1), \qquad t_{\mathcal{C}} \geq t + 3t_{exp} + O(1),$$

where $t_{\mathcal{C}}$ is the running time of $\mathcal{C}$ and t_{exp} is the computation time of a single exponentiation in G.

Here we use a collision-resistant hash function $H : \{0,1\}^\lambda \times \{0,1\}^* \to Z_q$ where the hash key is $K \leftarrow\$ \{0,1\}^\lambda$, instead of the random oracle model.

Proof 5.1.1 We first show the construction of the algorithm $\mathcal{B}$ which can solve the 2-OMDL problem using the forger $\mathcal{F}$. $\mathcal{B}$ is given a instances of the OMDL problem Y_0 and Y_1, chooses $K \in \{0,1\}^\lambda$ at uniformly random, and gives K and $(X, R) = (Y_0, Y_1)$ to $\mathcal{F}$ as a hash key and a public key, respectively.

$\mathcal{B}$ responds to $\mathcal{F}$'s signing query as follows.

Signing Query. When receiving a query m', $\mathcal{B}$ computes $c' \leftarrow H(K, Y_1||Y_0||m')$. $\mathcal{B}$ sends $Y_1 Y_0^{c'}$ to the oracle DL$(\cdot)$, and after that, it receives s' from the oracle. Finally, $\mathcal{B}$ returns s' to $\mathcal{F}$.

When $\mathcal{F}$ terminates with a valid forgery, $\mathcal{B}$ obtains the set of public keys $\{(X_i^*, R_i^*)\}_{i=1}^n$, the set of secret keys $\{(x_i, r_i)\}_{i\in[1,n]\setminus\{k\}}$, and the set of messages $(\{m_i^*\}_{i=1}^n, \tilde{s}^*)$ where $(Y_0, Y_1) \in \{(X_i^*, R_i^*)\}_{i=1}^n$, k is the index such that $(Y_0, Y_1) = (X_k^*, R_k^*)$, $m_k^* \neq m'$, and sk_k is $(\perp, \perp)$. $\mathcal{B}$ computes $c_i^* \leftarrow H(K, R_i||X_i||m_i^*)$, and it halts if $c_k^* = c'$. Let HC be the event that $c_k^* = c'$ holds. When $\mathcal{F}$ succeeds in forging a signature and HC does not happen, $\mathcal{B}$ computes the discrete logarithm of Y_0 as

$$y_0 \leftarrow \frac{\tilde{s}^* - s'_{J^*} - \sum_{i\in[1,n]\setminus\{k\}} .(x_i c_i^* + r_i)}{(c_k^* - c')} \mod q.$$

Finally, $\mathcal{B}$ outputs $(y_0, s'_1 - y_0 c'_1)$ as the solution to the instance of the 2 OMDL problem. Let $\mathsf{Succ}_\mathcal{F}$ be the event that $\mathcal{F}$ succeeds in forging a signature. Then the probability ε' that $\mathcal{B}$ succeeds is bounded as follows.

$$\varepsilon' \geq \Pr[\mathsf{Succ}_\mathcal{F} \wedge \overline{\mathsf{HC}}].$$

Secondly, we show the construction of the algorithm $\mathcal{C}$ which can find a collision of H using a forger $\mathcal{F}$. $\mathcal{C}$ is given a hash key K, generates (pk, sk) by **KeyGen** and runs $\mathcal{F}$ on input the public key $pk = (X, R)$ and a hash key K. $\mathcal{C}$ responds to $\mathcal{F}$'s signing query by using sk and the signing algorithm. When $\mathcal{F}$ terminates with a valid forgery, $\mathcal{C}$ obtains a set of public keys $\{(X_i^*, R_i^*)\}_{i=1}^n$, a set of secret keys $\{(x_i, r_i)\}_{i\in[1,n]\setminus\{k\}}$, and the set of messages $(\{m_i^*\}_{i=1}^n, \tilde{s}^*)$ where $(Y_0, Y_1) \in \{(X_i^*, R_i^*)\}_{i=1}^n$, k is the index such that $(Y_0, Y_1) = (X_k^*, R_k^*)$, $m_k^* \neq m'$ and sk_k is $(\perp, \perp)$. If $\mathsf{Succ}_\mathcal{F} \wedge \mathsf{HC}$ happens, it holds that $H(K, Y_1||Y_0||m_k^*) = H(K, Y_1||Y_0||m')$ and $Y_1||Y_0||m_k^* \neq Y_1||Y_0||m'$. Hence $\mathcal{C}$ outputs $(Y_1||Y_0||m_k^*, Y_1||Y_0||m')$ as collision of H when $\mathsf{Succ}_\mathcal{F} \wedge \mathsf{HC}$ happens. Otherwise, $\mathcal{C}$ halts. The advantage $\mathrm{Adv}_H(\mathcal{C})$ is bounded as

$$\mathrm{Adv}_H(\mathcal{C}) \geq \Pr[\mathsf{Succ}_\mathcal{F} \wedge \mathsf{HC}].$$

Putting the above two bounds together, we have

$$\varepsilon = \Pr[\mathsf{Succ}_{\mathcal{F}}] = \Pr[\mathsf{Succ}_{\mathcal{F}} \wedge \overline{\mathsf{HC}}] + \Pr[\mathsf{Succ}_{\mathcal{F}} \wedge \mathsf{HC}] \leq \varepsilon' + \mathrm{Adv}_H(\mathcal{C}).$$

The running time t' of $\mathcal{B}$ is the running time t of $\mathcal{F}$ plus t_{exp} time for an exponentiation in G in a signing query, plus $O(1)$ time for some setup. The running time $t_{\mathcal{C}}$ of $\mathcal{C}$ is the running time t of $\mathcal{F}$ plus $3t_{exp}$ time for three exponentiations in a signing query and generating keys, plus $O(1)$ time for some setup. □

5.2 The Rogue-Key Attack against OTAS

In above section, we proved OTAS secure under the OMDL assumption and the existence of a collision-resistant hash function in the KOSK model. The KOSK model is essential for OTAS because there is an effective rogue-key attack. Zhao mentioned a similar as an ephemeral rogue-key attack in [Zha19]. The procedure of this attack is as follows. The forger $\mathcal{F}$ is given a challenge key $pk_0 = (X_0, R_0)$ and a hash key K.

1. $\mathcal{F}$ chooses a message m_0^* and computes $c_0^* \leftarrow H(K, R_0||X_0||m_0^*)$.
2. $\mathcal{F}$ chooses a cosigner's key $pk_1 = (X_1, R_1) = (g^{x_1}, (R_0 X_0^{c_0^*})^{-1})$ and a message m_1^*, and computes $c_1^* \leftarrow H(K, R_1||X_1||m_1^*)$.
3. $\mathcal{F}$ computes $s_a \leftarrow x_1 c_1^* \mod q$, then outputs the set of the public keys $\{(X_0, R_0), (X_1, R_1)\}$, the set of messages $\{m_0^*, m_1^*\}$, and the forgery s_a.

The above output of $\mathcal{F}$ is a valid forgery because it holds that $c_0^* = H(K, R_0||X_0||m_0^*)$, $c_1^* = H(K, R_1||X_1||m_1^*)$, and

$$g^{s_a} X_0^{-c_0^*} X_1^{-c_1^*} = g^{x_1 c_1^*} X_0^{-c_0^*} X_1^{-c_1^*} R_0 R_0^{-1} = R_0 R_1.$$

This rogue-key attack is carried out by generating R dishonestly, which is a part of the public key. In the KOSK model, $\mathcal{F}$ cannot apply this attack because it is hard for $\mathcal{F}$ to output the discrete logarithm of $R_1 = (R_0 X_0^{c_0^*})^{-1}$.

5.2.1 On the KOSK Model and Its Implementation

The public key of OTAS consists of two elements in G. Also, we used the KOSK model to prove OTAS secure. In the implementation, the efficiency of the key-setup reduces than PCAS because the proof of possession (PoP) is required for each element in the public key. Moreover, since the security of OTAS is based on the OMDL assumption,

we cannot rewind a forger in a security proof. Therefore, unfortunately, the security of OTAS which uses the Schnorr signature as PoP is not clear because it is not necessarily possible to extract all cosigners' secret keys from PoP by the Bagherzandi-Cheon-Jarecki generalized forking lemma [BCJ08] like PCAS. Furthermore, when a signature scheme used as PoP is secure in the random oracle model, the advantage of proving the security with the collision-resistant hash function fades out.

5.2.2 Variant Schemes

We can consider some variant schemes for OTAS by changing the input of the hash function.

There is a rogue-key attack against OTAS by generating R dishonestly (we describe to the detail in Section 5.2). OTAS can become robust against such an attack by changing the input R of the hash function to $\widetilde{R}$ which is the product of all signers' R.

Unfortunately, for the above variant scheme, there is an attack in the plain PK model by using a k-sum algorithm, which is similar to the attack in Section 4.4.1. To defend against this attack, we can come up with one solution to add a random value to the input of the hash function like PCAS. In this case, we should use the random oracle model, not a collision-resistant hash function. This variant scheme is almost as same as PCAS, and the difference is only the way to share $\widetilde{R}$. Because sharing $\widetilde{R}$ is required for every time we generate an aggregate signature, signers need to regenerate R_i as a public key in each signing. Then this variant is regarded as two-tier aggregate signatures. Also, we can prove such a scheme secure under the DL assumption without extracting the secret keys corresponding to all cosigners' R though R is a part of a public key.

Chapter 6

Conclusion

First, we have shown a sub-exponential time universal forger under a key-only attack in the KOSK model against a standard aggregate signature scheme [Zha19]. Also we theoretically analyzed the computational complexity of the proposed forger and explain the inapplicability in the Bitcoin system.

Next, we have proposed a new paradigm *pre-communication* and the PCAS scheme which is constructed based on this new paradigm and proved secure under the standard DL assumption and the KOSK model. By presenting the concrete rogue-key attack, we stated that the KOSK model is essential for PCAS. Moreover, we explained that we avoided Drijvers et al.'s attacks and impossibility results.

Finally, we also have proposed a one-time aggregate signature scheme that was built from the Bellare-Shoup one-time signature scheme. This scheme was proved secure under the OMDL assumption and the existence of a collision-resistant hash function in the KOSK model.

Chapter 7

Future Works

Up to this date, there is still no secure standard aggregate signature scheme from general elliptic curve groups. Constructing such schemes is an important open problem. In [BK20], Boneh and Kim mentioned that the techniques of their proposed scheme can be applied to the partial aggregate signature, which can partially combine individual signatures like Zhao's scheme. However, they did not specify the construction of such a scheme and did not prove the security. Proving the security of such a partial aggregate signature scheme is also worthwhile future work.

In practice, PCAS and OTAS need the proof-of-possession (PoP) because of their security in the KOSK model. Therefore to analyze the security of schemes equipped with a concrete PoP is also an important research direction.

Reference

[AGH10] Jae Hyun Ahn, Matthew Green, and Susan Hohenberger. Synchronized aggregate signatures: new definitions, constructions and applications. In *CCS 2010*, pages 473–484, 2010.

[BCJ08] Ali Bagherzandi, Jung Hee Cheon, and Stanislaw Jarecki. Multisignatures secure under the discrete logarithm assumption and a generalized forking lemma. In *CCS 2008*, pages 449–458, 2008.

[BDN18] Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. In *ASIACRYPT 2018*, pages 435–464, 2018.

[BGLS03] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *EUROCRYPT 2003*, pages 416–432, 2003.

[BGOY07] Alexandra Boldyreva, Craig Gentry, Adam O'Neill, and Dae Hyun Yum. Ordered multisignatures and identity-based sequential aggregate signatures, with applications to secure routing. In *CCS 2007*, pages 276–285, 2007.

[BGR12] Kyle Brogle, Sharon Goldberg, and Leonid Reyzin. Sequential aggregate signatures with lazy verification from trapdoor permutations - (extended abstract). In *ASIACRYPT 2012*, pages 644–662, 2012.

[BJ] Ali Bagherzandi and Stanislaw Jarecki. Identity-based aggregate and multi-signature schemes based on RSA. In *Public Key Cryptography - PKC 2010*, volume 6056 of *Lecture Notes in Computer Science*, pages 480–498.

[BK20] Dan Boneh and Sam Kim. One-time and interactive aggregate signatures from lattices. 2020.

[BLOR20] Fabrice Benhamouda, Tancrède Lepoint, Michele Orrù, and Mariana Raykova. On the (in)security of ROS. *IACR Cryptol. ePrint Arch.*, page 945, 2020.

[BMP16] Rachid El Bansarkhani, Mohamed Saied Emam Mohamed, and Albrecht Petzoldt. MQSAS - A multivariate sequential aggregate signature scheme. In *ISC 2016*, pages 426–439, 2016.

[BN06] Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In *CCS 2006*, pages 390–399, 2006.

[BNN07] Mihir Bellare, Chanathip Namprempre, and Gregory Neven. Unrestricted aggregate signatures. In *ICALP 2007*, pages 411–422, 2007.

[Bol03] Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In *PKC 2003*, pages 31–46, 2003.

[BS07] Mihir Bellare and Sarah Shoup. Two-tier signatures, strongly unforgeable signatures, and fiat-shamir without random oracles. In *PKC 2007*, pages 201–216, 2007.

[CDG18] Sandro Coretti, Yevgeniy Dodis, and Siyao Guo. Non-uniform bounds in the random-permutation, ideal-cipher, and generic-group models. In *CRYPTO 2018*, pages 693–721, 2018.

[Cra96] R. Cramer. *Modular design of secure, yet practical cryptographic protocols*. PhD thesis, University of Amsterdam, 1996.

[DEF+19] Manu Drijvers, Kasra Edalatnejad, Bryan Ford, Eike Kiltz, Julian Loss, Gregory Neven, and Igors Stepanovs. On the security of two-round multi-signatures. In *IEEE S&P 2019*, pages 1084–1101, 2019.

[FLS12] Marc Fischlin, Anja Lehmann, and Dominique Schröder. History-free sequential aggregate signatures. In *SCN 2012*, pages 113–130, 2012.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, 1988.

[GR06] Craig Gentry and Zulfikar Ramzan. Identity-based aggregate signatures. In *PKC 2006*, pages 257–273, 2006.

[Gui20] Aurore Guillevic. A short-list of pairing-friendly curves resistant to special TNFS at the 128-bit security level. In *PKC 2020*, pages 535–564, 2020.

[HKW15] Susan Hohenberger, Venkata Koppula, and Brent Waters. Universal signature aggregators. In *EUROCRYPT 2015*, pages 3–34, 2015.

[HOS+20] Goichiro Hanaoka, Kazuo Ohta, Yusuke Sakai, Bagus Santoso, Kaoru Takemure, and Yunlei Zhao. Cryptanalysis of aggregate γ-signature and practical countermeasures in application to bitcoin. Cryptology ePrint Archive, Report 2020/1484, 2020.

[HSW13] Susan Hohenberger, Amit Sahai, and Brent Waters. Full domain hash from (leveled) multilinear maps and identity-based aggregate signatures. In *CRYPTO 2013*, pages 494–512, 2013.

[HW18] Susan Hohenberger and Brent Waters. Synchronized aggregate signatures from the RSA assumption. In *EUROCRYPT 2018*, pages 197–229, 2018.

[JMV01] Don Johnson, Alfred Menezes, and Scott A. Vanstone. The elliptic curve digital signature algorithm (ECDSA). *Int. J. Inf. Sec.*, 1(1):36–63, 2001.

[KB16] Taechan Kim and Razvan Barbulescu. Extended tower number field sieve: A new complexity for the medium prime case. In *CRYPTO 2016*, pages 543–571, 2016.

[KL08] Jonathan Katz and Andrew Y. Lindell. Aggregate message authentication codes. In *CT-RSA 2008*, volume 4964 of *Lecture Notes in Computer Science*, pages 155–169, 2008.

[LLY13] Kwangsu Lee, Dong Hoon Lee, and Moti Yung. Sequential aggregate signatures made shorter. In *ACNS 2013*, pages 202–217, 2013.

[LMRS04] Anna Lysyanskaya, Silvio Micali, Leonid Reyzin, and Hovav Shacham. Sequential aggregate signatures from trapdoor permutations. In *EUROCRYPT 2004*, pages 74–90, 2004.

[LOS+06] Steve Lu, Rafail Ostrovsky, Amit Sahai, Hovav Shacham, and Brent Waters. Sequential aggregate signatures and multisignatures without random oracles. In *EUROCRYPT 2006*, pages 465–485, 2006.

[LSP82] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[Mau05] Ueli M. Maurer. Abstract models of computation in cryptography. In *IMA 2005*, pages 1–12, 2005.

[MOR01] Silvio Micali, Kazuo Ohta, and Leonid Reyzin. Accountable-subgroup multisignatures: extended abstract. In *CCS 2001*, pages 245–254, 2001.

[MPSW18] Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. Simple schnorr multi-signatures with applications to bitcoin. *IACR Cryptol. ePrint Arch.*, 2018:68, 2018.

[MT07] Di Ma and Gene Tsudik. Extended abstract: Forward-secure sequential aggregate authentication. In *S&P 2007*, pages 86–91, 2007.

[MWLD10] Changshe Ma, Jian Weng, Yingjiu Li, and Robert H. Deng. Efficient discrete logarithm based multi-signature scheme in the plain public key model. *Des. Codes Cryptogr.*, 54(2):121–133, 2010.

[Nak08] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*, 2008. `https://bitcoin.org/bitcoin.pdf`.

[Nev08] Gregory Neven. Efficient sequential aggregate signed data. In *EUROCRYPT 2008*, pages 52–69, 2008.

[Pol78] John M. Pollard. Monte Carlo methods for index computation mod p. *Mathematics of Computation*, 32:918–924, 1978.

[PV05] Pascal Paillier and Damien Vergnaud. Discrete-log-based signatures may not be equivalent to discrete log. In *ASIACRYPT 2005*, pages 1–20, 2005.

[Res10] Certicom Research. *SEC 2: Recommended Elliptic Curve Domain Parameters*, 2010. `http://www.secg.org/sec2-v2.pdf`.

[RY07] Thomas Ristenpart and Scott Yilek. The power of proofs-of-possession: Securing multiparty signatures against rogue-key attacks. In *EUROCRYPT 2007*, pages 228–245, 2007.

[Sch89] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *CRYPTO '89*, pages 239–252, 1989.

[Sch01] Claus-Peter Schnorr. Security of blind discrete log signatures against interactive attacks. In *ICICS 2001,*, pages 1–12, 2001.

[STV+16] Ewa Syta, Iulia Tamas, Dylan Visher, David Isaac Wolinsky, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ismail Khoffi, and Bryan Ford. Keeping authorities "honest or bust" with decentralized witness cosigning. In *S&P 2016*, pages 526–545, 2016.

[Wag02] David A. Wagner. A generalized birthday problem. In *CRYPTO 2002*, pages 288–303, 2002.

[XZF05] Jing Xu, Zhenfeng Zhang, and Dengguo Feng. ID-based aggregate signatures from bilinear pairings. In *CANS 2005*, pages 110–119, 2005.

[YZ13] Andrew Chi-Chih Yao and Yunlei Zhao. Online/offline signatures for low-power devices. *IEEE Trans. Information Forensics and Security*, 8(2):283–294, 2013.

[Zha19] Yunlei Zhao. Practical aggregate signature from general elliptic curves, and applications to blockchain. In *AsiaCCS 2019*, pages 529–538, 2019.

Acknowledgement

I would like to thank my main advisor Assoc. Prof. Bagus Santoso for your fruitful discussions, creative comments, and for giving me a chance to engage in research as a research assistant of National Institute of Advanced Industrial Science and Technology (AIST). I could not write this thesis without his guidance and advice about research. I would also like to thank Yasutada Oohama, Tsutom Kawabata, and Hideki Yagi for their guidance.

I would like to thank members of Santoso, Oohama Kawabata Laboratory for their encouragement, friendship, and pleasant private time. Thank you, Takahiro Arai, Yuji Hatta, Takumi Katayama, and Yusuke Hashimoto.

I would like to heartily thank Goichiro Hanaoka for your support, and acceptance of me as a research assistant in AIST. I am deeply grateful to the members of AIST. I especially thank to Yusuke Sakai, and Kazuo Ohta for frequent and fruitful discussion, insightful suggestions. Without their guidance and cooperation, this thesis could not be done.

Finally, I would like to thank my family for their financial and mental supports.

Appendix A

Detail of Circumventing the Drijvers et al.'s Impossibility Results

Here we explain that, fortunately, Drijvers et al.'s impossibility results [DEF+19] fails to apply to our PCAS scheme (with t).

A.1 Counterintuitiveness on Security of PCAS

The security of PCAS may be counterintuitive for the readers who know the Drijvers et al.'s impossibility results because of the simplicity and round-efficiency of PCAS. Drijvers et al. used a meta-reduction technique and argued that it is impossible to prove that many of the known *two-round* multi-signature schemes [STV+16, MPSW18, MWLD10, BCJ08] are secure. The range of the targets of their arguments seems broad enough to cover PCAS, and thus one may wonder if it is impossible to prove PCAS secure. Moreover, while Drijvers et al. provided a secure two-round multi-signature scheme named mBCJ, it is extremely carefully designed to avoid every difficulty posed by their impossibility results. In contrast to mBCJ, PCAS looks too simple to be secure at first glance. To make matters even worse, PCAS is an aggregate signature scheme but not a multi-signature scheme, which makes the situation more complicated. However, despite this impression, PCAS is secure and avoids the impossibility results.

A.2 Resolving Counterintuitiveness

To resolve this counterintuitiveness, we will explain that PCAS without t *is* a target of the impossibility results, and, contrary to this, PCAS itself *is not* a target. Such an explanation, given later in Section A.2.2, not only resolves the counterintuitiveness but

also provides a deeper understanding of the essential reason why our reduction works correctly. One can argue that if we are convinced of the correctness of the security proof of PCAS, it is obvious that the meta-reduction technique fails somewhere. However, this superficial argument does not provide a better understanding of the security of PCAS. We deepen our understanding by trying to apply the meta-reduction technique to PCAS and pinpointing where the meta-reduction fails.

Before continuing this discussion, we remind readers of the common strategy of meta-reduction arguments and how such an argument can be dismissed.

A.2.1 Reviewing Meta-reduction Argument

To observe the impossibility of a reduction-based proof, basically, it is sufficient to find a carefully designed adversary[1] whose queries cannot be responded to correctly by *any* reduction. Concretely, if we can observe that simulation against such an adversary is as hard as solving some assumed-to-be-hard problem, we can conclude that a reduction-based security proof is impossible. To make this observation precise, a meta-reduction argument first assumes an *arbitrary* reduction being able to simulate the oracle even against the above adversary. Then it constructs a concrete algorithm which solves the assumed-to-be-hard problem by exploiting the *arbitrary* reduction's ability to simulating the oracle. The point is that the reduction is fixed *arbitrarily*. By this *arbitrariness*, *any* reduction is excluded from proving the security of the cryptographic scheme in question. By this means, we can conclude that there is no reduction-based proof of the security of the scheme.

A.2.2 Dismissing Meta-reduction Argument

Instead, if we want to dismiss the applicability of such an argument, we can provide the following counterargument. Firstly, we construct a concrete reduction which can simulate the oracle even against the above adversary (and can prove the security of the cryptographic scheme in question). Then we confirm that the meta-reduction fails to do that, if the meta-reduction tries to solving the assumed-to-be-hard problem by exploiting *this* reduction's ability to simulate the oracle. Furthermore, for a deeper understanding, when we confirm that, it is important to pinpoint where the meta-reduction fails in its procedure. Typically, it happens because *this* reduction is carefully designed not only to correctly simulate the oracle, that is usual for reduction-based security proofs

[1] Remember the forger described in Section 4.4.1.

but also to prevent the meta-reduction from exploiting the reduction's ability to simulate oracle. Then, the meta-reduction fails to be applied to *this* reduction, in other words, this reduction constitutes a counterexample to the *generality* of the meta-reduction. In this way, the impossibility argument is dismissed. We remark that for this counterargument, it is sufficient to construct a *certain* reduction. This is in contrast to the situation of arguing the impossibility of a reduction-based proof, where we need to focus on an *arbitrary* reduction.

A.3 Case Studies Using PCAS

Below we provide case studies of the above general discussion in Sections A.2.1 and A.2.2.

A.3.1 Security of PCAS without t Is Unprovable

For providing a case study, let us go back to the discussion on the applicability of the meta-reduction arguments to PCAS. The explanation below follows the above strategy mentioned in Section A.2.1.

Towards this end, we remind readers that the following basic (soundness) property of the Schnorr protocol:

> For a fixed commitment R, one not knowing the witness can obtain only a single challenge c (with a valid response s) where (R, c, s) constitutes a valid transcript.

Usually, this property is discussed for malicious provers (or forgers) to argue the soundness of the protocol (or the unforgeability of its Fiat-Shamir transformation). However, in our context, it is important to observe that this property is also true for *reductions*. This is true for reductions because reductions do not know the witness, but can only simulate a transcript of the Schnorr protocol by the honest-verifier zero-knowledge property.

For PCAS without t, the above property prevents *any* reduction from simulating the signing oracle when the reduction interacts with the following carefully designed forger $\mathcal{F}$. We remind readers that to argue the impossibility, we focus on an arbitrary reduction which tries to simulate the oracle and to prove the security. Let X be the public key of the honest signer, and the carefully designed forger $\mathcal{F}$ works as follows.

1. The forger $\mathcal{F}$ initiates a pre-communication session, and as a malicious aggregator it receives R from the honest signer;

2. it queries $(\widetilde{R}_0, X, m)$ and $(\widetilde{R}_1, X, m)$ to the random oracle with $\widetilde{R}_0 \neq \widetilde{R}_1$ and $m \in \{0,1\}^*$, and confirms that $H(\widetilde{R}_0, X, m) \neq H(\widetilde{R}_1, X, m)$, which holds with overwhelming probability;
3. it chooses $b \leftarrow \{0,1\}$, and sends $\widetilde{R}_b$ to the honest signer as the helper information;
4. it issues a signing query m (with respect to the above helper information $\widetilde{R}_b$) to the honest signer and obtains a response s where $(R, H(\widetilde{R}_b, X, m), s)$ forms a valid transcript of the Schnorr protocol.

Notice that due to the above basic property of the Schnorr protocol, a reduction cannot responds to this forger's signing query at least with $b = 0$ or with $b = 1$. We remind that we are considering a reduction which is arbitrarily given and emphasize that this is the case regardless of how cleverly the reduction computes R submitted by the reduction in Step 1. This is because otherwise the reduction can compute the discrete logarithm of X without any help of the forger.

To construct a meta-reduction which solves an assumed-to-be-hard problem, this $\mathcal{F}$ is essential. We make use of $\mathcal{F}$ and construct a concrete meta-reduction. The fundamental observation for constructing the meta-reduction is as follows. If a reduction were able to respond to the signing query from $\mathcal{F}$ (with high probability), the reduction would know that the discrete logarithm of X; for the meta-reduction to exploit this observation, the meta-reduction internally runs the reduction multiple times and obtains two transcripts $(R, H(\widetilde{R}_0, X, m), s_0)$ and $(R, H(\widetilde{R}_1, X, m), s_1)$, from which the meta-reduction can extract the discrete logarithm of X, the assumed-to-be-hard problem.

A.3.2 Meta-reduction Arguments Are Inapplicable to PCAS

Finally, we discuss the construction of a reduction for PCAS. We already constructed such a reduction in the proof of Theorem 4.2.1, Besides, we also pinpoint where the attempt to apply the meta-reduction arguments to PCAS fails, which corresponds to the explanation in Section A.2.2.

To see why the meta-reduction fails, firstly, from the reduction's perspective, let us describe how our concrete reduction (in Theorem 4.2.1) works against the above $\mathcal{F}$:

1. When $\mathcal{F}$ initiates a pre-communication session, the reduction simulates a transcript (R, c, s) of the Schnorr protocol;
2. then $\mathcal{F}$ issues two random oracle queries $(\widetilde{R}_0, X, m)$ and $(\widetilde{R}_1, X, m)$; the reduction responds to them by fresh hash values;
3. $\mathcal{F}$ submits a helper information $\widetilde{R}_b$ where $b \leftarrow \{0,1\}$;

4. $\mathcal{F}$ issues a signing query m with respect to the helper information $\widetilde{R}_b$; the reduction responds to it by setting $H(\widetilde{R}_b, X, t, m) \leftarrow c$ where $t \leftarrow \{0,1\}^\kappa$ and sending (t, s) as the honest signer's signature on m, where $(R, H(\widetilde{R}_b, X, t, m), s)$ forms a valid transcript of the Schnorr protocol.

This strategy can be seen as *deferring* the programming of the random oracle to the point when $\mathcal{F}$ issues a signing query, namely, the point of Step 4. We say this strategy "defers" the programming because without t in the hashed message, the reduction needs to decide to program the random oracle as either $H(\widetilde{R}_0, X, m) \leftarrow c$ or $H(\widetilde{R}_1, X, m) \leftarrow c$ at the point of the random oracle queries, namely, the point of Step 2. Contrary to this, with t, the random oracle queries in Step 2. differs from $(\widetilde{R}_b, X, t, m)$ (with overwhelming probability), and thus the reduction can program the random oracle at Step 4. Another important observation is that regardless of the helper information $\widetilde{R}_0$ or $\widetilde{R}_1$ that $\mathcal{F}$ uses, the reduction uses the unique c that it knows for simulating the response to the signing query.

From the meta-reduction perspective, the meta-reduction is unable to exploit this reduction for solving the assumed-to-be-hard problem. To see this, let us remind readers of the above-mentioned meta-reduction's strategy and pinpoint where this strategy fails. In this strategy, the meta-reduction somehow "rewinds" the reduction to obtain two transcripts $(R, H(\widetilde{R}_0, X, t, m), s)$ and $(R, H(\widetilde{R}_1, X, t', m), s')$. Using these transcripts, the meta-reduction extracts the discrete logarithm of X, that is, it solves the assumed-to-be-hard problem. This strategy in fact fails. It can be true that the meta-reduction obtains two transcripts for both cases of $b = 0$ and $b = 1$, namely, $(R, H(\widetilde{R}_0, X, t, m), s)$ and $(R, H(\widetilde{R}_1, X, t', m), s')$. However, even if it obtains, these transcripts have the same challenge, which are not useful for solving the assumed-to-be-hard problem, This is the point what we want to pinpoint. Notice that this equality happens because regardless of whether $b = 0$ or $b = 1$, the reduction sets $H(\widetilde{R}_b, X, t, m) \leftarrow c$, where c is uniquely c known to the reduction, and sends (t, s) in which (R, c, s) is a valid transcript as a signature on m.