Journal of Information Processing Vol.24 No.2 275–291 (Mar. 2016)

[DOI: 10.2197/ipsjjip.24.275]

Regular Paper

Fail-Stop Signatures for Multiple-Signers: Definitions, Constructions, and Their Extensions

Nobuaki Kitajima[1,2] Naoto Yanai[3,4] Takashi Nishide[1] Goichiro Hanaoka[4]
Eiji Okamoto[1]

Received: June 27, 2015, Accepted: December 7, 2015

Abstract: Fail-stop signatures (FSS) provide the security for a signer against a computationally unbounded adversary by enabling the signer to provide a proof of forgery. Conventional FSS schemes are for a single-signer setting, but in the real world, there is a case where a countersignature of multiple signers (e.g., a signature between a bank, a user, and a consumer) is required. In this work, we propose a framework of FSS capturing a multi-signer setting and call the primitive fail-stop multisignatures (FSMS). We propose a generic construction of FSMS via the bundling homomorphisms proposed by Pfitzmann and then propose a provably secure instantiation of the FSMS scheme from the factoring assumption. Our proposed schemes can be also extended to fail-stop aggregate signatures (FSAS).

Keywords: fail-stop signatures, fail-stop multisignatures, bundling homomorphisms, information-theoretic security

1. Introduction

1.1 Background

The security of many signature schemes is based on computational assumptions, and their security can be guaranteed as long as the assumptions hold. In recent years, developments in computer science are remarkable and there are many results in which several computational assumptions were broken [13], [17]. However, there now exist various documents that must not be forged, e.g., financial documents by banks or official documents by governments. These documents should be stored for the long term, and we are aware of the potential vulnerability of the computational security which may be broken in future. Hence, in order to protect the documents, we take into account the information-theoretic security, and it does not require any computational assumption for providing the security. In other words, a scheme based on the information-theoretic security is secure even against a computationally-unbounded adversary. Fail-stop signatures (FSS) [30] that we focus on are a known approach such as a cryptographic primitive. FSS have a valuable property such that even if a signature is forged, i.e., some computational assumption is broken, an honest signer can prove a forgery by virtue of the information-theoretic security. A main idea for the construction of FSS is that, whereas a signer can output only some signature on a message for a public key, there potentially exist many signatures which are accepted by a verification test with the message and the public key: more precisely, its algorithm outputs a public key such that there are a large amount of candidates corresponding to its secret key. This construction allows signers to introduce the choice problem of the secret key corresponding to the public key, and a space of signatures depends on a given secret key in general. We underline that, unless an adversary knows the true secret key, the honest signer can generate a signature as a *proof of forgery* that is different from one generated by the adversary. Thus, the signer can stop a dangerous situation showing the proof of forgery even if the adversary has unbounded computational power. On the other hand, via the computational security, a polynomially-bounded malicious signer cannot repudiate her valid signature as a forgery, and hence security for a verifier can be guaranteed.

Our main motivation for this work is to consider that most existing FSS schemes capture only a single signer setting and there are few applications of such a scenario in the real world. Generally speaking, applications of the FSS schemes, e.g., financial documents or official documents, require a countersignature between multiple entities, i.e., a bank and users or a president and ministers. We mention that a scenario for collecting individual signatures generated by each signer may be insufficient. In particular, for confirmation in decision making in the real world, there is often an interaction process between all persons associated with the documents. Namely, we consider that FSS should take into account a multi-signer setting with such an interaction process as a functionality of a cryptographic primitive. This is a cryptographically enhanced approach. Although there are several approaches which concern the multi-signer setting, they did not formalize its model and have slight gaps from the applications in the sense of a lack of the interaction process between the signers. Hence, in this work, we realize a framework of FSS for multiple signers and call this framework *fail-stop multisignatures (FSMS)*. One of potential applications of FSMS is *the digital ne-*

[1] Graduate School of Systems and Information Engineering, University of Tsukuba, Ibaraki 305–8573, Japan
[2] Tata Consultancy Services Japan, Ltd., Minato, Tokyo 105–8508, Japan
[3] Graduate School of Information Science and Technology, Osaka University, Suita, Osaka 565–0871, Japan
[4] Information Technology Research Institute, AIST, Koto, Tokyo 135–0064, Japan

gotiable certificate of deposit (NCD) system [10] as described in Section 1.4.

The preliminary version of this paper was presented at Asia Joint Conference on Information Security (AsiaJCIS) 2015 in May 2015, and this paper is the full version of it. The main difference between the preliminary version and this paper is that we propose an instantiation of fail-stop aggregate signatures (FSAS).

1.2 Our Contribution

In this work there are following three contributions: (1) We define a security model of an FSMS scheme. Our security model consists of two standpoints, i.e., signers and a verifier. Such a setting is based on the existing model of FSS for a single signer setting [20]. In our model, we furthermore allow an adversary, who has unlimited computational power, to collude with all of the signers except a target signer as a feature of the multi-signer setting. (2) We propose a generic construction of FSMS. We foresee that a homomorphic-like property for combining signatures is required in order to construct a multisignature scheme. From such an observation, we focus on the Pfitzmann's bundling homomorphisms [25]. The bundling homomorphisms are collision-resistant algebraic structures and we found the fact that the bundling homomorphisms potentially include both the choice problem of secret keys for FSS and the homomorphic-like property for multisignatures. (3) We propose an instantiation of the FSMS scheme whose signature size is independent from the number of signers. In this approach, we focus on the function for the bundling homomorphisms proposed by Mashatan and Ouafi [20] (the MO function for short) as an instantiation of the FSMS scheme. We consider that the size of signatures can be effectively compressed utilizing the MO function with a common modulus, and propose an efficient and provably secure FSMS scheme based on the factoring assumption. We furthermore extend the proposed FSMS schemes to an aggregate signature scheme [5].

1.3 Related Work

The existing approaches for FSS with the multi-signer setting are the scheme by Mambo and Okamoto [19] (MO94 scheme for short) and the scheme by Susilo et al. [29] (SSNP99 scheme for short). The MO94 scheme is a signature-chain based approach by multiple signers: more precisely, each signer signs each bit of a message and possesses a pair of the signature chain and the message. On the other hand, the SSNP99 scheme is a threshold signature scheme, and according to Susilo et al. signers can generate n-out-of-n signatures as multisignatures. We note that it is hard to add a new signer to the signing group, because these shares have to be recalculated when the new signer participates in the signing group. Namely, we consider that the SSNP99 scheme is slightly different from standard multisignature schemes. Moreover, there were no formalized security model and security proofs in both schemes. Thus, we consider that there is no framework and provably secure construction of FSMS. In the following, we recall the history of FSS and multisignatures.

The first FSS scheme was proposed by Waidner and Pfitzmann [30]. Later, van Heyst and Pedersen [14] proposed an FSS scheme based on the discrete logarithm problem (DLP). It is known as the most efficient scheme. Pedersen et al. [24] defined the security of FSS, and we briefly review it in Section 2.3. Mashatan and Ouafi [20], and Schmidt-Samoa [28] proposed FSS schemes based on the factoring assumption. Our instantiation is based on that of Mashatan and Ouafi [20]. As the latest work, Yamakawa et al. [31] have proposed a short FSS scheme with a variant of bundling homomorphisms.

The first multisignature scheme was proposed by Itakura and Nakamura [16]. Ohta and Okamoto [23] defined the model for an attack on the multisignature, called the adaptive-chosen-message attack and adaptive-insider attack (ACMA&AIA). Micali et al. [22] discussed the ACMA&AIA including the key-generation phase. These models are slightly intuitive in the sense that they do not consider the signing with an interaction between signers. The first formalized model was proposed by Boldyreva [4]. As a more advanced multisignature scheme Boneh et al. [5] proposed an aggregate signature scheme where each signer is allowed to sign an individual document and any party can aggregate these signatures into a single short signature. After the first aggregate signature scheme was proposed, many researchers have tried to propose an efficient aggregate signature scheme.

1.4 Application

We describe *the negotiable certificate of deposit* (*NCD*) [10] as a potential application of an FSMS scheme. In short, the NCD is a special type of fixed deposit which can be transfered to others. A main advantage of the NCD is that free and fair trading can be realized by virtue of a negotiation between an assignor and an assignee. However, generating a countersignature (composed of the signatures of the assignor, the assignee, a bank, a notary office, a payment and settlement authority, and other authorities) is required, and this can be an administrative burden since there are a large number of complicated trading transactions that involve imprinting and exchanging relevant documents. In addition, since the bank which floats the NCD and accepts the trading for the last time has to manage relevant documents for a long time, an optimization for the paper management is also required. We consider if our FSMS scheme is useful for the digital NCD system in the light of the efficiency and the feature. In our observation, each entity provides a signing data with an interaction process with each other. These data are concatenated by each entity and all the entities have a countersignature of the signing group to confirm an agreement of its members.

1.5 Paper Organization

In this section, we described backgrounds, contributions, related works and potential applications of this work. In Section 2, we describe several areas of knowledge for understanding this work and, in Section 3, we define the syntax and the security of FSMS. Then we propose a generic construction of FSMS in Section 4, and its instantiation from the factoring assumption in Section 5. We give only the constructions in these sections, and their security proofs are given in Appendix A.3 and Appendix A.4, respectively. We also describe how to extend these schemes to

aggregate signature schemes in Section 6 and show the full constructions in A.5. Finally, we describe the conclusion of this work in Section 7.

2. Preliminaries

2.1 Notation

We use the following notations throughout this paper. Let $y \leftarrow \mathcal{A}(x)$ denote that y is assigned an output of an algorithm $\mathcal{A}$ running on an input x. For any set X, we denote by $\|X\|$ its cardinality. Let $x \in_R X$ denote x chosen from X according to the uniform distribution. For any sets X and Y, we denote by $X \backslash Y$ the difference set $\{x \mid x \in X \ \wedge \ x \notin Y\}$. Let $\mathbb{Z}_n^*$ denote the set $\{x \mid 1 \leq x \leq n-1, \ \mathsf{gcd}(x,n) = 1\}$. Let $\varphi(\cdot)$ denote the Euler totient function, and in our scheme, $\varphi(n) = (p-1)(q-1)$ holds. Let $|x|$ denote the binary length of x. Finally, we denote by $\mathsf{negl}(\cdot)$ a negligible function with respect to a security parameter.

2.2 Algebraic Setting

We describe the following security assumption and algebraic structure that will be used throughout this paper. First, we define the *factorization assumption with a-strong primes* [27]. We call a prime number p a-strong, if $p = 2ap' + 1$ for a prime $p' > 2a$. For $a = 1$, we obtain the traditional definition of strong primes. We suppose a probabilistic polynomial time algorithm Gen that, on input a security parameter k, picks an odd integer a and generates a random a-strong prime $p = 2ap' + 1$ and a regular, i.e., not a-strong, prime number q, such that $|p| = |q|$. Gen outputs $n = pq$ and a along with p and q. When we use such a prime number p as a prime factor of n and the factorization of n is difficult, then we say that the factorization assumption with a-strong primes holds. The formal definition is given as follows.

Definition 1 (Factorization Assumption with a-Strong Primes) Let us consider a probabilistic polynomial time algorithm $\mathcal{A}_{\mathsf{FACT}}$ which takes as input a modulus $n = pq$ and an odd integer a, such that p, q, and $(p-1)/2a$ are all prime numbers, and outputs p and q. The advantage of $\mathcal{A}_{\mathsf{FACT}}$ can be defined as the probability

$$\epsilon := \Pr[(p,q) \leftarrow \mathcal{A}_{\mathsf{FACT}}(1^k, n, a) \mid (n, a, p, q) \leftarrow \mathsf{Gen}(1^k, a)],$$

and its execution time t. We say that the (t, ϵ)-factorization assumption with a-strong primes holds against $\mathcal{A}_{\mathsf{FACT}}$ with respect to Gen if $\mathcal{A}_{\mathsf{FACT}}$ cannot output (p, q) for a given $(1^k, n, a)$ within the execution time t and with a success probability greater than ϵ.

We note that, due to a release of an integer a such that $a|(p-1)(q-1)$, a and a modulus n have to be chosen as $|a| < |n|/4$ in order to prevent the attack by Coppersmith [7]. Throughout this paper, we set up $|a| = 80$ and $|n|/4 = 512$ to achieve 80-bit security in our scheme, and so there is no problem with respect to the attack. We refer readers who have more interests in this consideration to the results by Groth [12].

Second, we recall the definition of a *family of bundling homomorphisms* [25]. This notion is a kind of collision-resistant algebraic structure. By utilizing the bundling homomorphisms, we can construct a signature scheme where the number of secret keys corresponding to a public key is at least 2^τ with respect to a security parameter τ. The formal definition is given as follows.

Definition 2 (Family of Bundling Homomorphisms) Let $\eta \in I$ be an index for a family of triples (h_η, G_η, H_η) such that for all possible $\eta \in I$, $(G_\eta, +, 0)$ and $(H_\eta, \times, 1)$ are Abelian groups. Let $h_\eta : G_\eta \rightarrow H_\eta$. The triple (h_η, G_η, H_η) is called a family of bundling homomorphisms with degree level 2^τ and collision-resistance security of level k if it satisfies:

(1) For every $\eta \in I$, h_η is a homomorphism.
(2) There exist polynomial time algorithms for sampling from I, computing h_η and the group operations in G_η and H_η, for every $\eta \in I$.
(3) For any output $\mu \in H_\eta$, $\|\{x \in G_\eta \,|\, \mu = h_\eta(x)\}\| \geq 2^\tau$. We call 2^τ the bundling degree of the homomorphisms.
(4) It is computationally infeasible for any probabilistic polynomial time algorithm $\tilde{\mathcal{A}}$ to find collisions, i.e., the probability that $\tilde{\mathcal{A}}$ outputs x_1 and x_2 such that $h_\eta(x_1) = h_\eta(x_2)$ and $x_1 \neq x_2$ hold is $\mathsf{negl}(k)$, where the probability is taken among random choice of η and random coins of $\tilde{\mathcal{A}}$.

2.3 Fail-Stop Signatures

We briefly review FSS. When a forgery does not happen, FSS run an ordinary signing process. However, once a forgery happens, a signer executes a prove-forgery algorithm which is another signing algorithm to prove that the signature is a forgery, i.e., a cryptographic assumption has been broken. As described above, a public key is generated via the choice problem of secret keys, and so the signer is able to output only one signature for a pair of the public key and the true secret key. If any forged signature is different from a valid signature via the true secret key, there are two acceptable signatures on the same message. This statement means that the forgery happened.

There are two concepts for the security of FSS, i.e., the *signer's security* and the *verifier's security*. An adversary who has unbounded computational power can always compute a set of signatures accepted via the verification test. The signer's security against such an adversary is achieved by the information-theoretic security, and the security-level is determined by a security parameter $\sigma \in \mathbb{N}$. The adversary cannot generate a valid signature which is exactly the same as one by the signer except with a probability $2^{-\sigma}$. On the other hand, a verifier should be able to resist a repudiation of malicious signers. By computing a proof of forgery, the malicious signers can repudiate signatures generated by themselves as forgeries. The security in such a situation is known as the verifier's security and is based on only the computational security; more precisely, the probability that signers can repudiate signatures is $\mathsf{negl}(k)$ with respect to a security parameter k.

We mention that in its basic form FSS is one-time signatures. In order to be able to sign multiple messages, each signer has to generate at least $(N+1)$ k-bit strings as secret information where N is the number of messages. This is an unavoidable fact since the adversary who has unbounded computational power can deduce information of secret keys and reduce its entropy [15]. In order to apply FSS to the efficient on-line digital payment systems, we assume the use of collision-free accumulators [1] where they accumulate many one-time public keys into a single short public key. Barić and Pfitzmann have proved the security of the

FSS scheme with accumulators in Ref. [1].

Theorem 1 (Barić and Pfitzmann [1]) If an underlying one-time FSS scheme is secure, then an FSS scheme with the accumulators is secure.

2.4 General Construction of Fail-Stop Signatures

In this section, we briefly explain the Pfitzmann's general construction of FSS [25]. The basic idea of this construction is to use a high bundling degree of the bundling homomorphisms and to hide a true secret key of a signer. The Pfitzmann's construction consists of the following algorithms. Here, we assume that the key generation algorithm is executed by a signer in conjunction with a single verifier (or a trusted center). More precisely, the verifier (or the center) generates a prekey described below, and then the signer generates a pair of a secret key and a public key under the prekey. This setting is necessary in order to guarantee the verifier's security [*1].

- $\mathsf{PrekeyGen}(1^k, 1^\sigma)$: As a prekey, it picks a random index K for a bundling homomorphism h_η, and sets $h := h_K$, $G := G_K$ and $H := H_K$.
- $\mathsf{KeyGen}(K)$: For verification of the prekey, check if h is a group homomorphism with bundling degree 2^τ (e.g., using zero-knowledge proof) [*2]. If not, it regenerates a random index K to obtain such a bundling homomorphism. Once it is convinced, it outputs a secret key as $sk = (sk_1, sk_2) \in_R G^2$, and computes a corresponding public key as $pk = (pk_1, pk_2) = (h(sk_1), h(sk_2)) \in H^2$.
- $\mathsf{Sign}(sk, m)$: A message space $\mathcal{M}$ is given as a subset of $\mathbb{Z}$. To sign a message $m \in \mathcal{M}$, it computes $s = sk_1 + msk_2$, where msk_2 is m additions of sk_2 over G.
- $\mathsf{Verify}(pk, m, s)$: It outputs 1 if $h(s) = pk_1 pk_2^m$ holds; otherwise it outputs 0.
- $\mathsf{ProveForgery}(sk, m^\star, s^\star)$: Given a forgery $s^\star$ on a message $m^\star$ accepted by the verification test, it computes $pr = sk_1 + m^\star sk_2$ and outputs $(m^\star, s^\star, pr)$.
- $\mathsf{VerifyProof}(pk, m^\star, s^\star, pr)$: It outputs 1 if $s^\star \neq pr$ and $h(s^\star) = h(pr)$ hold; otherwise it outputs 0.

We say that a signature s is *acceptable* on a message m if we have $\mathsf{Verify}(pk, m, s) \to 1$. Moreover, we say that a proof pr is *valid* on a message $m^\star$ if we have $\mathsf{VerifyProof}(pk, m^\star, s^\star, pr) \to 1$.

Next, we briefly explain the security of this construction. One might think that there are at least $2^{2\tau}$ possible secret keys corresponding to the public key (pk_1, pk_2). However, as the equation $sk_1 = s - msk_2$ must hold in G, then the number of the possible secret keys reduces to 2^τ. The signer can provide a valid proof of forgery as long as the forged signature $s^\star$ on a message $m^\star$ differs from a proof pr, i.e., a signature generated via the true secret key, on $m^\star$. In order to measure the adversary's probability of generating an unprovable forgery, we have to estimate the number of these possible keys which produce $s^\star$ on $m^\star$. This number is obtained from the size of set

$$T = \{d \in G : h(d) = 1 \wedge (m^\star - m)d = 0\}.$$

In order to provide an upper-bound of the adversary's success probability, we consider the worst case. Namely, we must find the maximum number taken over all possible messages $m^\star \neq m$. Thus, we can obtain the bound

$$T_{\max} = \max_{m' \in \mathcal{M} \setminus \{0\}} |\{d \in G : h(d) = 1 \wedge m'd = 0\}|.$$

A more detailed proof and analysis of this construction can be found in Ref. [25].

Theorem 2 (Security of the Pfitzmann's Construction [25]) Let (k, σ) be security parameters and let a fail-stop signature scheme follow the general construction described above. Then we have

(1) The scheme provides a level of security k for the verifier.
(2) 2^τ is chosen such that $T_{\max}/2^\tau \leq 2^{-\sigma}$, then the scheme provides a level of security σ for the signer.

In this paper, we paraphrase (1): let us consider a probabilistic polynomial time algorithm $\tilde{\mathcal{U}}_{pol}$ which takes as input K and outputs $(pk, m^\star, s^\star, pr) \leftarrow \tilde{\mathcal{U}}_{pol}(K)$ where $K \leftarrow \mathsf{PrekeyGen}(1^k, 1^\sigma)$. The advantage of $\tilde{\mathcal{U}}_{pol}$ can be defined as the probability $\epsilon'' := \Pr[\mathsf{VerifyProof}(pk, m^\star, s^\star, pr) = 1]$ and its execution time t''. We say that the scheme is (t'', ϵ'')-secure for the verifier with respect to $\mathsf{PrekeyGen}$ if there is no $\tilde{\mathcal{U}}_{pol}$ which can output $(pk, m^\star, s^\star, pr)$ such that $\mathsf{VerifyProof}(pk, m^\star, s^\star, pr) = 1$ within the execution time t'' and with a success probability greater than ϵ''.

2.5 Multisignatures

A multisignature scheme is a cryptographic primitive to generate signatures by multiple signers on a common message. In conventional schemes, its definition is that the total size of signatures in a multisignature scheme is less than i times of the size in a signature scheme with a single signer where i is the number of signers, and so there are several schemes whose signature size is $O(i)$. In this paper, we define a multisignature scheme as a signature scheme whose signature size is $O(1)$ regardless of the number of signers. Constructions of a multisignature scheme are various, and these are classified according to methods by which signers transfer signatures. A construction that we discuss in this paper is a scheme with an interactive protocol: more precisely, each signer has a signing function in its local environment and generates a multisignature via an interaction process with co-signers. For instance, in a multisignature scheme, each signer generates an individual partial signature via its own signing function taking a list of co-signers to generate a multisignature, and broadcasts the generated partial signature to other co-signers. Then, the signer receives partial signatures from the other co-signers as response. Finally, each signer can generate the multisignature by combining the partial signatures. Note that all the co-signers obtain the same final multisignature. Meanwhile, the multisignature is verifiable by one verification test with all public keys of the signers and the multisignature. The main technical difficulty for multisignatures is to achieve both an efficient scheme and provable

[*1] An FSS scheme can be made much more efficient under a single-verifier setting. This is a common requirement in applications such as digital payment systems, where a bank is the single verifier.
[*2] We note that this is provable via a general zero-knowledge proof for the NP-language.

security. A multisignature scheme needs an algebraic structure to combine individually generated signatures into a single short signature, while such an algebraic structure often makes a forgery by an adversary easy. Namely, research on multisignatures can be seen as finding an algebraic structure to support both the efficiency and the security.

The security of multisignatures is to prevent a forgery against insider attackers in a group of signers. In particular, in the security model in Ref. [4], an adversary can register signers with his/her chosen secret and public keys as insider attackers, and can obtain multisignatures involving target signer's signature via interactions between the target signer and the insider attackers. Finally, the adversary tries to output a signature on a message which has never been queried to the target signer. The model is a natural extension of an ordinary signature scheme for a single signer except for adaptively registering the insider attackers.

3. Definitions of Security

In this section, we define the security of the FSMS scheme. We firstly discuss some trivial attack on the FSMS scheme and its restriction to resist the attack in Section 3.1. Then we describe a syntax of the FSMS scheme and the security model utilizing an adaptive-chosen-message attack and fail-stop-multisignatures' adaptive-insider attack (ACMA&FSMS-AIA). Our security model is a variant of the security model of FSS and captures two notions, i.e., the signer's security and the verifier's security. Note that the security model in this section is for the use of multiple messages.

3.1 Limitation on the Number of Signers

In most FSS schemes, if a collision of signatures, i.e., two acceptable signatures, can be found, then the computational security can be broken. In other words, the most important point is whether a same public key (generated from different secret keys) is found or not. For FSMS, we must pay attention to a limitation on the number of signers. If there are malicious signers who have the same public keys in the signing group, they can repudiate their signature even if they have only polynomially-bounded computational power. This fact means that given multisignatures are doubtful and a verifier may be damaged by these signatures. In order to provide a rigorous security for the verifier, we introduce a new parameter $\omega_{\max}$ as a limitation on the number of signers who can participate in the signature generation. When the probability that two or more signers have the same public keys is less than $2^{-\sigma}$, we can obtain the inequality $1 - e^{-\omega_{\max}(\omega_{\max}-1)/2n} \leq 2^{-\sigma}$ by using the birthday paradox, where n is the range of values of the public key and $n = 2^k$ holds. By using the Taylor expansion and rearranging, we can obtain $\omega_{\max} \leq \lfloor \sqrt{2^{k-\sigma+1}} \rfloor$ as the limitation of $\omega_{\max}$. In addition, by choosing an appropriate tuple $(k, \sigma, \omega_{\max})$, we can show that all signer's 2^σ candidates of the secret keys can be uniformly distributed in the space with respect to k. This setting also allows FSMS to provide the signer's security at the same time. For instance, if k is $2,048$ and σ is 80 for 80-bit security, this is good enough. We show how to derive the limitation of $\omega_{\max}$ in Appendix A.1.

3.2 Model of FSMS Scheme

We define the model of an FSMS scheme.

Definition 3 (Fail-Stop Multisignatures) A fail-stop multisignature scheme is a tuple of algorithms (*Setup, KeyGen, MSign, MVerify, ProveForgery, VerifyProof*) such that:

$\mathsf{Setup}(1^k, 1^\sigma, 1^{\omega_{\max}}) \to \mathsf{pp}$: This is a probabilistic algorithm that, on input of security parameters $(k, \sigma, \omega_{\max})$, outputs a public parameter pp.

$\mathsf{KeyGen}(\mathsf{pp}, 1^{N_{\max}}) \to (sk_j, pk_j)$: This is a probabilistic algorithm that, on input of pp and an integer $N_{\max}$, outputs a secret/public key pair (sk_j, pk_j) which can be used for signing $N_{\max}$ times. For the sake of simplicity, the input $1^{N_{\max}}$ is omitted if the scheme is used as a one-time signature scheme.

$\mathsf{MSign}(\mathsf{pp}, sk_j, m, \ell) \to \{(m, S, L), \perp\}$: This is an (probabilistic) interactive algorithm that, on input of pp, sk_j, a message m, and a counter ℓ incremented at each invocation of this algorithm, outputs a tuple (m, S, L) where S is a multisignature and L is a set $\{pk_j\}_{j=1}^{i}$ of i signers. If $i > \omega_{\max}$, the output is $\perp$.

$\mathsf{MVerify}(\mathsf{pp}, m, S, L) \to \{0, 1\}$: This is a deterministic (possibly probabilistic) algorithm that, on input of pp, m, S, and L, outputs 0 or 1.

$\mathsf{ProveForgery}(\mathsf{pp}, sk_j, m^\star, S^\star, L^\star) \to \{pr, \perp\}$: This is an (probabilistic) algorithm for generating a proof of forgery. Given pp, sk_j, $m^\star$, $S^\star$, and $L^\star$, it outputs a bit string pr as a proof of forgery or $\perp$ in a case of failure.

$\mathsf{VerifyProof}(\mathsf{pp}, m^\star, S^\star, L^\star, pr) \to \{0, 1\}$: This is a deterministic (possibly probabilistic) algorithm that, on input of pp, $m^\star$, $S^\star$, $L^\star$, and pr, outputs 0 or 1.

We say that a multisignature S is *acceptable* on a message m if we have $\mathsf{MVerify}(\mathsf{pp}, m, S, L) \to 1$. Moreover, we say that a proof pr is *valid* on a message $m^\star$ if we have $\mathsf{VerifyProof}(\mathsf{pp}, m^\star, S^\star, L^\star, pr) \to 1$. MSign and ProveForgery are interactive algorithms and each signer runs these algorithms with the inputs described above. Similarly as the conventional model of multisignatures [2], [4], we assume that the signers are connected to each other via point-to-point links over which they can send a message. A different point between these algorithms is to take the set L or not. Whereas MSign algorithm is an ordinary signing algorithm that is almost the same as that of a normal (non-FSS) multisignature scheme, ProveForgery algorithm is a proof protocol against a forgery. Hence, ProveForgery algorithm should firstly check the validity of the given multisignature and clearly show the list of problematic signers in order to generate a corresponding proof. The algorithm takes the set L toward these capability.

We require for an FSMS scheme to satisfy the following conditions as correctness.

Definition 4 (Correctness of Fail-Stop Multisignatures)

(1) Every honestly generated multisignature is acceptable, i.e., for any $k, \sigma, \omega_{\max}, N_{\max} \in \mathbb{N}$ and message $m \in \mathcal{M}$, we have

$$\Pr[\mathsf{MVerify}(\mathsf{pp}, m, S, L) = 1] = 1,$$

where $\mathsf{pp} \leftarrow \mathsf{Setup}(1^k, 1^\sigma, 1^{\omega_{\max}})$, $(sk_j, pk_j) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, 1^{N_{\max}})$, and $(m, S, L) \leftarrow \mathsf{MSign}(\mathsf{pp}, sk_j, m, \ell)$.

(2) Every honestly generated proof of forgery is valid, i.e., for any $k, \sigma, \omega_{\max}, N_{\max} \in \mathbb{N}$, message $m^\star \in \mathcal{M}$ and tuple

$(S^\star, L^\star)$, we have

$$\Pr[\mathsf{VerifyProof}(\mathsf{pp}, m^\star, S^\star, L^\star, pr) = 1 \mid \mathsf{MVerify}(\mathsf{pp}, m^\star, S^\star, L^\star) = 1, pr \neq \perp] = 1,$$

where $\mathsf{pp} \leftarrow \mathsf{Setup}(1^k, 1^\sigma, 1^{\omega_{\max}})$, $(sk_j, pk_j) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, 1^{N_{\max}})$, and $pr \leftarrow \mathsf{ProveForgery}(\mathsf{pp}, sk_j, m^\star, S^\star, L^\star)$.

3.3 ACMA&FSMS-AIA

We define the adaptive-chosen-message attack and fail-stop-multisignatures' adaptive-insider attack, i.e., ACMA&FSMS-AIA. The basic difference between ACMA&FSMS-AIA and the conventional security model of multisignatures is whether there is a computationally unbounded adversary or not. Moreover, the adversary can collude with the other signers against a target signer in comparison with the conventional model of FSS. Under this setting, we have to consider if the signer's security described in Section 2.3 is satisfied. In addition, we have to consider if the verifier's security is satisfied at the same time. In order to capture these scenarios, we define the following two games, the FSMS Limited Repudiation (FSMS.LR) game and the FSMS Full Forging (FSMS.FF) game. These are described as interactions between an adversary and a challenger in the ACMA&FSMS-AIA scenario. The FSMS.LR game is related to the verifier's security. The goal of malicious signers is to repudiate their valid signature by generating a proof of forgery, i.e., an acceptable signature. On the other hand, the FSMS.FF game is related to the signer's security. The goal of the adversary who has unbounded computational power is to forge a multisignature that includes a partial signature of the target signer. In both games, we assume the knowledge-of-secret-key (KOSK) assumption [4] where an adversary has to provide not only a public key but also a secret key in the registration phase of signers. The assumption is often necessary for the security model of multisignatures in order to prevent malicious generations of public keys [22], and it can be implemented via a cryptographic identification scheme to prove a knowledge of a secret key in the real world. In general, it is common that there is an interaction for the registration between a user and a service provider at the start of the use of an application. In such a situation, it is not difficult to implement a cryptographic identification scheme between the user, who generates signatures afterward, and the provider as the proof of a knowledge of secret information. Therefore, the KOSK assumption is reasonable. In the following games, we denote by a symbol $*$ the target signer's information.

3.3.1 FSMS.LR Game

We define the FSMS.LR game. In this game, there exists a group of polynomially-bounded malicious signers $\tilde{\mathcal{U}}_{pol}$ and a challenger C corresponding to a verifier. The goal of $\tilde{\mathcal{U}}_{pol}$ is to repudiate their signature by computing a valid proof of forgery. Note that the signing query does not need to be defined because the adversary in this game is a group of malicious signers. Thus there is no signing oracle. We also note that $\tilde{\mathcal{U}}_{pol}$ has to give both a secret key and a public key to C in the registration of signers due to the KOSK assumption described above. In such a situation, the goal of $\tilde{\mathcal{U}}_{pol}$ is to output a valid proof of forgery involving a target signer who is designated in the setup phase. The procedure of the FSMS.LR game is as follows:

- Setup : C executes $\mathsf{Setup}(1^k, 1^\sigma, 1^{\omega_{\max}})$ to output pp. Then, $\tilde{\mathcal{U}}_{pol}$ generates $(sk^*, pk^*) = (sk_1, pk_1)$, and gives pk^* to C.
- Create User query $(2 \le j \le q_c + 1)$: $\tilde{\mathcal{U}}_{pol}$ generates (sk_j, pk_j) and then gives (sk_j, pk_j) to C, where $q_c \le \omega_{\max} - 1$. C checks if pk_j is the public key corresponding to sk_j. If so, we say that pk_j is registered; otherwise C outputs an error symbol $\perp$.
- Output : $\tilde{\mathcal{U}}_{pol}$ outputs $(m^\star, S^\star, L^\star, pr^\star)$, where $L^\star = \{pk_j\}_{j=1}^{q_c+1}$ and $pk_1 = pk^*$. C checks if all the following winning conditions hold: $\mathsf{MVerify}(\mathsf{pp}, m^\star, S^\star, L^\star)$ outputs 1; all of the public keys except pk^* in $L^\star$ are registered; $\mathsf{VerifyProof}(\mathsf{pp}, m^\star, S^\star, L^\star, pr^\star)$ outputs 1; the number of signers is not over $\omega_{\max}$.

$\tilde{\mathcal{U}}_{pol}$ wins the game if all the winning conditions hold. The following definition corresponds to the *verifier's security*.

Definition 5 We say that a fail-stop multisignature scheme is (t, q_c, ϵ)-secure for the verifier if there is no group of polynomially-bounded malicious signers $\tilde{\mathcal{U}}_{pol}$ that wins the FSMS.LR game within an execution time t, generating at most q_c ($\le \omega_{\max} - 1$) Create User queries, and with a success probability greater than ϵ.

3.3.2 FSMS.FF Game

We define the FSMS.FF game. In this game, there exists a computationally unbounded adversary $\tilde{\mathcal{F}}_{exp}$ who colludes with a group of signers, and a challenger C corresponds to a target signer. The goal of $\tilde{\mathcal{F}}_{exp}$ is to forge a multisignature for which C cannot generate a proof of forgery. The procedure of the FSMS.FF game is as follows:

- Setup : C executes $\mathsf{Setup}(1^k, 1^\sigma, 1^{\omega_{\max}})$ to obtain pp. Then C executes KeyGen to generate $(sk^*, pk^*) = (sk_1, pk_1)$, and then gives pp and pk^* to $\tilde{\mathcal{F}}_{exp}$.
- Create User query $(2 \le j \le q_c + 1)$: $\tilde{\mathcal{F}}_{exp}$ generates (sk_j, pk_j) and then gives (sk_j, pk_j) to C, where $q_c \le \omega_{\max} - 1$. C checks if pk_j is the public key corresponding to sk_j. If so, we say that pk_j is registered; otherwise C outputs an error symbol $\perp$.
- Sign query $(1 \le d \le N_{\max})$: $\tilde{\mathcal{F}}_{exp}$ gives a query m_d to C. C executes MSign with $\tilde{\mathcal{F}}_{exp}$ and outputs a multisignature (m_d, S_d, L_d).
- Output : $\tilde{\mathcal{F}}_{exp}$ outputs $(m^\star, S^\star, L^\star)$, where $L^\star = \{pk_j\}_{j=1}^{q_c+1}$ and $pk_1 = pk^*$. C checks if the following winning conditions hold: $\mathsf{MVerify}(\mathsf{pp}, m^\star, S^\star, L^\star)$ outputs 1; all of the public keys except pk^* in $L^\star$ are registered; $\mathsf{ProveForgery}(\mathsf{pp}, sk^*, m^\star, S^\star, L^\star)$ outputs $\perp$; $m^\star \notin \{m_d\}_{d=1}^{N_{\max}}$; sk^* is used less than $N_{\max}$ times; the number of signers is not over $\omega_{\max}$.

$\tilde{\mathcal{F}}_{exp}$ wins the game if all the winning conditions hold. The following definition corresponds to the *signer's security*.

Definition 6 We say that a fail-stop multisignature scheme is secure for the signers with $2^{-\sigma}$ if there is no computationally unbounded adversary $\tilde{\mathcal{F}}_{exp}$ who wins the FSMS.FF game with a probability greater than $2^{-\sigma}$, where σ is a security parameter.

According to Pfitzmann [25], in a security analysis of an FSS scheme we do not need to consider the game against

a polynomially-bounded adversary. Suppose that there is a polynomially-bounded adversary $\tilde{\mathcal{F}}_{pol}$ who can forge multisignatures with non-negligible probability. Then we can construct a polynomial time algorithm that breaks the verifier's security with the non-negligible probability, and it contradicts the verifier's security. In Appendix A.2, we give a security model in FSMS, called the FSMS Limited Forging (FSMS.LF) Game, against $\tilde{\mathcal{F}}_{pol}$, and show the details of the proof.

4. General Construction of Fail-Stop Multisignatures

In this section, we propose a general construction of FSMS. Our construction is provably secure against the ACMA&FSMS-AIA. We first describe a technical problem for constructing an FSMS scheme, and then propose the general construction.

4.1 Our Approach

A problem for constructing an FSMS scheme is that there is some contradiction between a property of FSS and a property of multisignatures. An essential approach of FSS is to construct a public key cryptosystem such that a large amount of candidates of secret keys exist, but such a construction seems to need a special and complicated algebraic structure. On the other hand, a multisignature scheme is required to efficiently compress signatures, and so we need a property to support an aggregation of the signatures. In general, such a property is given via a simple algebraic structure such as the BLS short signature scheme [6] for the BGLS03 aggregate signature scheme [5]. In fact, the MO94 scheme brings on a degradation of the efficiency due to the Lamport signature scheme [18] and the SSNP99 scheme brings on a restriction in participation of a new signer due to a threshold scheme.

We focus on Pfitzmann's approach [25] in order to overcome the problem described above. Pfitzmann constructed FSS via the bundling homomorphisms. The bundling homomorphisms are collision-resistant algebraic structures, and they both have a property that for any output there are many candidate inputs and a homomorphic property. We found the fact that the former property contributes to the security of an FSS scheme while the latter one supports the aggregation of signatures in a multisignature scheme efficiently. Moreover, the security of an FSMS scheme can be proven by a reduction to the Pfitzmann's construction in the manner of extracting a target signer's signature from a forgery in the FSMS scheme. Thus, we can propose a generic construction of FSMS via the bundling homomorphisms.

4.2 The Construction

The general construction of FSMS consists of the following algorithms:

$\mathsf{Setup}(1^k, 1^\sigma, 1^{\omega_{\max}})$: As a prekey, it picks random index K for a bundling homomorphism h_η and sets $h := h_K, G := G_K$ and $H := H_K$. It outputs $\mathsf{pp} := (K, h, G, H, \omega_{\max})$.

$\mathsf{KeyGen}(\mathsf{pp})$ [*3]: Firstly, it checks if h is a group homomorphism with bundling degree 2^τ (e.g., using zero-knowledge proof) [*4]. If not, it executes Setup algorithm to obtain such a bundling homomorphism. Once it is convinced, it outputs a secret key as $sk_j = (sk_{j,1}, sk_{j,2}) \in_R G^2$ and its corresponding public key as $pk_j = (pk_{j,1}, pk_{j,2}) = (h(sk_{j,1}), h(sk_{j,2})) \in H^2$.

$\mathsf{MSign}(\mathsf{pp}, sk_j, m)$ [*5]: The message space $\mathcal{M}$ is given as a subset of $\mathbb{Z}$. It computes and broadcasts a partial signature $s_j \in G$ as $s_j = sk_{j,1} + msk_{j,2}$ and receives partial signatures of other signers. Given $\{s_j\}_{j=1}^i$, then it can obtain a multisignature $S \in G$ as $S = \sum_{j=1}^i s_j$ with respect to the public keys on $L = \{pk_j\}_{j=1}^i$. If $i > \omega_{\max}$, then it outputs $\perp$. Otherwise, it outputs (m, S, L).

$\mathsf{MVerify}(\mathsf{pp}, m, S, L)$: It outputs 1 if $h(S) = \prod_{j=1}^i (pk_{j,1} \times pk_{j,2}^m)$ holds. Otherwise, it outputs 0.

$\mathsf{ProveForgery}(\mathsf{pp}, sk_j, m^\star, S^\star, L^\star)$: By invoking MSign, it computes a partial signature $s_j \in G$ as $s_j = sk_{j,1} + m^\star sk_{j,2}$, and then obtains $pr = \sum_{j=1}^i s_j$ via partial signatures of other signers. If either $h(S^\star) \neq \prod_{j=1}^i (pk_{j,1} \times pk_{j,2}^{m^\star})$ or $S^\star = pr$ holds, then it outputs $\perp$. Otherwise, it outputs pr as a proof of forgery.

$\mathsf{VerifyProof}(\mathsf{pp}, m^\star, S^\star, L^\star, pr)$: It outputs 1 if $S^\star \neq pr$ and $h(S^\star) = h(pr)$ hold. Otherwise, it outputs 0.

Remark: One might think that an adversary who obtains two signatures can recover secret keys and forge a multisignature as a behavior of any signer, but we mention that the construction described above is a one-time signature scheme similar to the existing FSS schemes. Namely, a situation where the adversary can obtain multiple signatures of the same secret keys will never occur. We underline that we can extend the proposed scheme for multiple messages by a single public key utilizing approaches in Refs. [1], [14], [24]. We review the *collision-free accumulators* [1] in Appendix A.7, and then show a general construction of FSMS with them in Appendix A.8.

We show the following theorems, which assert the security of our general construction, and then give their proofs in Appendix A.3.

Theorem 3 The proposed general construction is (t', q_c, ϵ')-secure for the verifier if the Pfitzmann's construction of FSS is (t'', ϵ'')-secure, where $t'' = t' + O(q_c)$, $q_c \leq \omega_{\max} - 1$ and $\epsilon'' = \epsilon'$ hold.

Theorem 4 The proposed general construction is secure for the signer with a probability $T_{\max}/2^\tau$.

5. Instantiation of Fail-Stop Multisignatures

In this section, we propose an efficient instantiation of FSMS. Our construction is based on, as an instantiation, the MO function $h(x) = x^a \bmod n$ [20] whose computational complexity is equivalent to the problem of factoring an integer n. We set up a such that it is *not coprime* to the order of the group $\mathbb{Z}_n^*$, and it provides a property that every $x^a \in \mathbb{Z}_n^*$ has multiple a-th roots.

We emphasize that the original MO11 scheme has some potential vulnerability where an adversary can forge a new signature

[*3] Note that the input $1^{N_{\max}}$ is omitted since the scheme described above is used as a one-time signature scheme.

[*4] We note that this is provable via a general zero-knowledge proof for the NP-language.

[*5] Note that, similarly as mentioned on KeyGen, a counter ℓ is omitted of the input since the scheme we describe is used as a one-time signature scheme.

on a message $m + ax$ for any integer x from a prekey a and a signature on m generated by an honest signer. As the details described in Appendix A.6, the reason of the vulnerability is due to an incorrect estimation of the probability for the signer's security. We also suggest the use of a prime number as a and $\mathbb{Z}_a$ as the message space. Intuitively, the adversary cannot choose $m + ax$ as messages since the message space is smaller than a. Meanwhile, by adopting the prime number as a, the estimation of the probability in the proof of the MO11 scheme can be fixed.

5.1 The Construction

Our concrete construction consists of the following algorithms:

$\mathsf{Setup}(1^k, 1^\sigma, 1^{\omega_{\max}})$: It invokes Gen (in Section 2.2), and Gen chooses a σ-bit prime integer a and a prime p' such that $p' > 2a$. Gen computes a prime p as $p = 2ap' + 1$ and then chooses a prime q such that $|p| = |q|$, where $\gcd(q-1, a) = 1$ holds. Then, Gen computes $n = pq$ and outputs (n, a, p, q). At this point, the setup algorithm defines a group homomorphism h_n as $h_n : x \mapsto x^a \bmod n$ for $x \in \mathbb{Z}_n^*$, and finally outputs $\mathsf{pp} := (n, a, h_n, \mathbb{Z}_n^*, \omega_{\max})$.

$\mathsf{KeyGen}(\mathsf{pp})$ [*6]: To verify that the prekey is correctly generated, the signer utilizes a zero-knowledge proof that a indeed divides $\varphi(n)$ (such a proof can be constructed from general zero-knowledge proofs [11]). If not, it executes Setup to obtain such a pp. Otherwise, it outputs a secret key as $sk_j = (sk_{j,1}, sk_{j,2}) \in_R \mathbb{Z}_n^{*2}$ and its corresponding public key as $pk_j = (pk_{j,1}, pk_{j,2}) = (sk_{j,1}^a \bmod n, sk_{j,2}^a \bmod n)$.

$\mathsf{MSign}(\mathsf{pp}, sk_j, m)$ [*7]: The message space $\mathcal{M}$ is defined as $\mathcal{M} := \mathbb{Z}_a$. It computes and broadcasts a partial signature $s_j \in \mathbb{Z}_n^*$ as $s_j = sk_{j,1} sk_{j,2}^m \bmod n$ and receives partial signatures of other signers. Given $\{s_j\}_{j=1}^i$, then it computes a multisignature $S \in \mathbb{Z}_n^*$ as $S = \prod_{j=1}^i s_j \bmod n$ with respect to the public keys on $L = \{pk_j\}_{j=1}^i$. If $i > \omega_{\max}$, then it outputs $\perp$. Otherwise, it outputs (m, S, L).

$\mathsf{MVerify}(\mathsf{pp}, m, S, L)$: It outputs 1 if $S^a = \prod_{j=1}^i (pk_{j,1} \times pk_{j,2}^m) \bmod n$ holds. Otherwise, it outputs 0.

$\mathsf{ProveForgery}(\mathsf{pp}, sk_j, m^\star, S^\star, L^\star)$: It invokes MSign and computes a partial signature $s_j \in \mathbb{Z}_n^*$ as $s_j = sk_{j,1} sk_{j,2}^{m^\star} \bmod n$. By using partial signatures of other signers, it computes $pr = \prod_{j=1}^i s_j \bmod n \in \mathbb{Z}_n^*$. If either $(S^\star)^a \neq \prod_{j=1}^i (pk_{j,1} \times pk_{j,2}^{m^\star}) \bmod n$ or $S^\star = pr$ holds, then it outputs $\perp$. Otherwise, it outputs pr as a proof of forgery.

$\mathsf{VerifyProof}(\mathsf{pp}, m^\star, S^\star, L^\star, pr)$: It outputs 1 if $S^\star \neq pr$ and $S^{\star a} = (pr)^a \bmod n$ hold; otherwise it outputs 0.

Remark: The scheme described above is a one-time signature scheme, and so an adversary cannot forge multisignatures in a trivial way as described in Section 4.2. In addition, we can also extend the proposed scheme for multiple messages by a single public key utilizing approaches in Refs. [1], [14], [24] as described in Section 4.2. Moreover, unlike an RSA signature scheme, our FSMS scheme is secure although we apply a common modulus n. Shortly, $\tilde{\mathcal{U}}_{pol}$ cannot compute $\varphi(n)$ since $(sk_{j,1}, sk_{j,2}) \in \mathbb{Z}_n^{*2}$ and $(pk_{j,1}, pk_{j,2}) \in \mathbb{Z}_n^{*2}$ hold. One might think that, in terms of our security model (i.e., ACMA&FSMS-AIA) and its arguments (existence of malicious signers), there is the following problem; a couple of malicious signers U_1 and U_2 can collude and choose their secret keys $(sk_{j,1}, sk_{j,2}) \in \mathbb{Z}_n^{*2}$ $(j = 1, 2)$ in the following manner: $sk_{1,1} = sk_{2,1}^{-1} \bmod n$ and $sk_{1,2} = sk_{2,2}^{-1} \bmod n$ hold. Therefore, U_1's public key and partial signatures are the inverse of those of U_2's. In the current construction, these malicious signers may be able to hide and reveal their partial signatures *for different verifiers* due to the validation formula $S^a = \prod_{j=1}^i (pk_{j,1} \times pk_{j,2}^m) \bmod n$. Unless the verifier is aware that U_1 and U_2 are included in the signing group, the multisignature will be verified in either case. However, note that such a case cannot happen in our proposed FSMS scheme. As mentioned in Section 3, we assume that a single verifier (or a center trusted by verifiers) exists. In the single-verifier setting, this problem will never occur. Although we need the trusted center in the multiple-verifier setting, even in this case the security can be guaranteed as long as the center correctly and securely keeps the list L of public keys.

We show the following theorems, which assert the security of our proposed scheme. The proofs are given in Appendix A.4.

Theorem 5 The proposed scheme is (t, q_c, ϵ)-secure for the verifier if (t', ϵ')-factorization assumption with a-strong primes holds against a probabilistic polynomial time algorithm $\mathcal{A}_{\mathsf{FACT}}$ with respect to Gen, where $t' = t + 2t_e(1 + q_c)$, $q_c (\leq \omega_{\max} - 1)$, $\epsilon' = \epsilon$ and t_e is a computational time for one exponentiation.

Theorem 6 The proposed scheme is secure for the signer with a probability $2^{-\sigma}$.

5.2 Efficiency Comparison

Here, we compare the performance of our scheme with the MO94 scheme [19], and present the results in **Table 1**. For a performance evaluation of our proposed scheme, suppose that k is 2,048 bits, and σ is 80 bits for 80-bit security. The size of multisignatures in the MO94 scheme increases in proportion to the number of signers. Moreover, since the MO94 scheme is based on the Lamport signature scheme [18], each bit of a message must be signed, and every value increases in proportion to the message length $|m|$. We also note that the scheme [19] did not provide the formal model and its security proof.

We describe the MO94 scheme and our scheme as one-time signature schemes in which one secret key can be used for signing only a single message similar to conventional FSS schemes. However, note that we can extend our scheme to work for multiple messages by using one of several standard ways [1], [14], [24].

6. Application to Fail-Stop Aggregate Signatures

An aggregate signature scheme [5] is a generalized multisignature scheme in which each signer can sign an individual message and then anyone can aggregate these individual signatures into a single short signature. We underline that the proposed FSMS

[*6] Note that the input $1^{N_{\max}}$ is omitted since the scheme we describe is used as a one-time signature scheme.

[*7] Note that, similarly as mentioned on KeyGen, a counter ℓ is omitted of the input since the scheme we describe is used as a one-time signature scheme.

Table 1 Comparison of the existing FSMS schemes: For better readability and easier notations, we utilize the following parameters. Let $\tilde{k}$ and k be the length of the moduli of each scheme. We denote by $|m|$ the binary length of message m in the MO94 scheme, σ the message length in the other schemes, and by i the number of signers, respectively. Let Sign(#mult.) and Verify(#mult.) denote the number of modular multiplications required for signing and verification, respectively. We discuss these costs in terms of generating a partial signature for each signer. Finally, we denote by $<$ an upper-bound on the number of modular multiplications.

	Size of Secret Key (bits)	Size of Public Key (bits)	Size of Signature for i Signers (bits)	Sign (#mult.)	Verify (#mult.)	Security Proof
MO94	$2\tilde{k}\|m\|$	$2\tilde{k}\|m\|$	$i\tilde{k}\|m\|$	$< 4\tilde{k}\|m\|$	$< 2\tilde{k}\|m\|$	No
Ours	$2k$	$2k$	k	σ	$< 2\sigma$	Yes

scheme can be turned into an aggregate signature scheme. By introducing an individual message m_j for each signer and dividing MSign into two individual algorithms, i.e., a signing algorithm for generating a partial signature and an aggregation algorithm for compressing the signatures, in the proposed scheme, the fail-stop aggregate signature (FSAS) scheme can be constructed.

We note that, from the standpoint of security proofs, there is no difference between an interactive multisignature scheme and an aggregate signature scheme. The main difference between the two constructions is how to compress signatures: Whereas the multisignature scheme requires an interaction of signers in order to generate a compressed signature, the aggregate signature scheme allows anyone to arbitrarily compress individual signatures generated by each signer. The view of an adversary in the former model is to give a target signer messages and to receive partial signatures of the target signer. The view corresponds to that of an adversary in the aggregate signature where the adversary receives individual signatures generated by the target signer. Thus, the technicalities in these models are exactly the same and the security of the FSAS scheme is provable similarly to that of our FSMS scheme. We give the details in Appendix A.5.

7. Conclusion

Fail-stop signatures (FSS) are digital signatures which can guarantee information-theoretic unforgeability for a signer and computational undeniability for a verifier, and existing schemes have captured only single-signer setting. In this work, we defined fail-stop multisignatures (FSMS) which can be seen as a variant of FSS for multi-signer setting, and proposed a generic construction with bundling homomorphisms and its instantiation. The bundling homomorphisms provide three capabilities, i.e., the group homomorphism, the collision resistant, and many-to-one function, and these are used for constructing efficient and secure FSMS schemes. More specifically, the group homomorphism is necessary for combining individual signatures of multisignatures. Then, the collision resistance is utilized for the undeniability while the many-to-one function is utilized for the unforgeability. Under these observations, we proposed the generic construction, and proved its security by a reduction to the Pfitzmann's generic construction of FSS. We also proposed an efficient instantiation from the MO11 function. Our constructions can be extended into fail-stop aggregate signatures (FSAS) where they are more generalized multi-signer settings in the sense that each signer can deal with an individual message. For its purpose, we also defined their models. Our future work is to prove the security under stronger security models, such as the key registration model or the plain public key model.

Acknowledgments We would like to thank anonymous reviewers for their valuable comments. We would also like to appreciate the anonymous reviewers of AsiaJCIS 2015 and members of the study group "Shin-Akarui-Ango-Benkyo-Kai" for their helpful comments. This work was partially supported by JSPS KAKENHI Grant Numbers 26330151 and 26880012, Kurata Grant from The Kurata Memorial Hitachi Science and Technology Foundation, and JSPS A3 Foresight Program.

References

[1] Barić, N. and Pfitzmann, B.: Collisioin-Free Accumulators and Fail-Stop Signature Schemes Without Trees, *EUROCRYPT 1997*, *LNCS 1233*, pp.480–494 (1997).
[2] Bellare, M. and Neven, G.: Multi-Signatures in the Plain Public-Key Model and a General Forking Lemma, *ACM CCS 2006*, pp.390–399, ACM Press (2006).
[3] Benaloh, J. and de Mare, M.: One-Way Accumulators: A Decentralized Alternative to Digital Signatures (Extended Abstract), *EUROCRYPT 1993*, *LNCS 765*, pp.274–285 (1994).
[4] Boldyreva, A.: Threshold Signature, Multisignature and Blind Signature Schemes Based on the Gap-Diffie-Hellman-Group Signature Scheme, *PKC 2003*, *LNCS 2567*, pp.31–46 (2003).
[5] Boneh, D., Gentry, C., Lynn, B. and Shacham, H.: Aggregate and Verifiably Encrypted Signatures from Bilinear Maps, *EUROCRYPT 2003*, *LNCS 2656*, pp.416–432 (2003).
[6] Boneh, D., Lynn, B. and Shacham, H.: Short Signatures from the Weil Pairing, *ASIACRYPT 2001*, *LNCS 2248*, pp.514–532 (2001).
[7] Coppersmith, D.: Finding a Small Root of a Bivariate Integer Equation; Factoring with High Bits Known, *EUROCRYPT 1996*, *LNCS 1070*, pp.178–189 (1996).
[8] Fazio, N. and Nicolosi, A.: Cryptographic Accumulators: Definitions, Constructions and Applications, available from ⟨http://www-cs.ccny.cuny.edu/~fazio/pubs/FaNi03.pdf⟩.
[9] Frobenius, G.: *Über einen Fundamentalsatz der Gruppentheorie*, *II*, Sitzungs-Berichte der Preussischen Akademie Weissenstein (1907).
[10] Fujitsu Limited: The System of Electronic Negotiable Certificates of Deposit, available from ⟨http://www.ipa.go.jp/NBP/10-3ecabst/pdf/0-075-01.pdf#search='The+System+of+Electronic+Negotiable+Certificates+of+Deposit'⟩ (Japanese).
[11] Goldreich, O., Micali, S. and Wigderson, A.: Proofs That Yield Nothing but Their Validity or All Languages in NP Have Zero-Knowledge Proof Systems, *J. ACM*, Vol.38, No.3, pp.690–728 (1991).
[12] Groth, J.: Cryptography in Subgroups of $\mathbb{Z}_n^*$, *TCC 2005*, *LNCS 3378*, pp.50–65 (2005).
[13] Hayashi, T., Shimoyama, T., Shinohara, N. and Takagi, T.: Breaking pairing-based cryptosystems using η_T pairing over $GF(3^{97})$, Cryptology ePrint Archieve (2012), available from ⟨http://eprint.iacr.org/2012/042⟩.
[14] van Heyst, E. and Pedersen, T.P.: How to Make Efficient Fail-stop Signatures, *EUROCRYPT 1992*, *LNCS 658*, pp.366–377 (1992).
[15] van Heyst, E., Pedersen, T.P. and Pfitzmann, B.: New Constructions of Fail-Stop Signatures and Lower Bounds (Extended Abstract), *CRYPTO 1992*, *LNCS 740*, pp.15–30 (1992).
[16] Itakura, K. and Nakamura, K.: A Public-key Cryptosystem Suitable for Digital Multi-signatures, *Trans. IPSJ*, Vol.24, No.4, pp.474–480 (1983).
[17] Kleinjung, T., Aoki, K., Franke, J., Lenstra, A.K., Thome, E., Bos, J.W., Gaudry, P., Kruppa, A., Montgomery, P.L., Osvik, D.A., te R.H., Timofeev, A. and Zimmermann, P.: Factorization of a 768-bit RSA modulus, *CRYPTO 2010*, *LNCS 6223*, pp.333–350 (2010).

[18] Leslie Lamport: Constructing Digital Signatures from a One Way Function, *SRI International*, *CSL-98* (1979).
[19] Mambo, M. and Okamoto, E.: Fail-Stop Multisignatures, *IEICE Spring Conference* (1994) (Japanese).
[20] Mashatan, A. and Ouafi, K.: Efficient Fail-Stop Signatures from the Factoring Assumption, *ISC 2011*, *LNCS 7001*, pp.372–385 (2011).
[21] Mashatan, A. and Ouafi, K.: Forgery-Resilience for Digital Signature Schemes, *AsiaCCS 2012*, pp.24–25 (2012).
[22] Micali, S., Ohta, K. and Reyzin, L.: Accountable-Subgroup Multisignatures, *ACM CCS 2001*, pp.245–54, ACM Press (2001).
[23] Ohta, K. and Okamoto, T.: Multi-Signature Schemes Secure against Active Insider Attacks, *IEICE Trans. Fundamentals*, Vol.E82-A, No.1, pp.22–31 (1999).
[24] Pedersen, T.P. and Pfitzmann, B.: Fail-Stop Signatures, *SIAM Journal on Computing*, Vol.26, No.2 pp.291–330 (1997).
[25] Pfitzmann, B.: Digital Signature Schemes, General Framework and Fail-Stop Signatures, *LNCS 1100* (1996).
[26] Pfitzmann, B. and Waidner, M.: Fail-Stop Signatures and Their Application, *SECURICOM 1991*, pp.145–160 (1991).
[27] Rivest, R. and Silverman, R.: Are 'Strong' Primes Needed for RSA?, *Cryptology ePrint Archieve* (2001), available from ⟨http://eprint.iacr.org/2001/007⟩.
[28] Schmidt-Samoa, K.: Factorization-Based Fail-Stop Signatures Revisited, *ICICS 2004*, *LNCS 3269*, pp.118–131 (2004).
[29] Susilo, W., Safavi-Naini, R. and Pieprzyk, J.: Fail-Stop Threshold Signature Schemes Based on Elliptic Curves, *ACISP 1999*, *LNCS 1587*, pp.103–116 (1999).
[30] Waidner, M. and Pfitzmann, B.: The Dining Cryptographers in the Disco: under Conditional Sender and Recipient Untraceability with Computationally Secure Serviceability (Abstract), *EUROCRYPT 1989*, *LNCS 434*, p.690 (1990).
[31] Yamakawa, T., Kitajima, N., Nishide, T., Hanaoka, G. and Okamoto, E.: A Short Fail-Stop Signature Scheme from Factoring, *ProvSec 2014*, *LNCS 8782*, pp.309–316 (2014).

Appendix

A.1 How to Derive the Limitation of $\omega_{\max}$

In this section, we show how to derive the limitation of $\omega_{\max}$. Firstly, we can obtain the following inequality as we explain in Section 3.1:

$$1 - e^{-\omega_{\max}(\omega_{\max}-1)/2n} \leq 2^{-\sigma}.$$

We approximate and rearrange this inequality. We can obtain new inequality as follows:

$$e^{-\omega_{\max}^2/2n} \geq \frac{2^\sigma - 1}{2^\sigma}.$$

We take natural logarithm for both sides. Here, $n = 2^k$ holds:

$$\omega_{\max}^2 \leq \log\left(\frac{2^\sigma - 1}{2^\sigma}\right)^{-1} \cdot 2n = \log\left(\frac{2^\sigma}{2^\sigma - 1}\right) \cdot 2^{k+1}.$$

The following inequality holds since $\omega_{\max}$ is a positive number:

$$\omega_{\max} \leq \sqrt{2^{k+1}\log\left(1 + \frac{1}{2^\sigma - 1}\right)}.$$

By using the Taylor expansion for $\log\left(1 + \frac{1}{2^\sigma - 1}\right)$, we can obtain an approximate value $\frac{1}{2^\sigma - 1}$. Therefore, the following inequality holds:

$$\omega_{\max} \leq \sqrt{\frac{2^{k+1}}{2^\sigma - 1}} \approx \sqrt{2^{k-\sigma+1}}.$$

Finally, we obtain the upper-bound of $\omega_{\max}$:

$$\omega_{\max} \leq \lfloor \sqrt{2^{k-\sigma+1}} \rfloor.$$

$\tilde{\mathcal{U}}_{\sqrt{\updownarrow}}$ or $\tilde{\mathcal{F}}_{\urcorner\S\lrcorner}$ can execute Create User query at most $\lfloor \sqrt{2^{k-\sigma+1}} \rfloor - 1$ times. In this case, suppose that there are at least 2^σ possible secret keys for each signer, and $k = 2{,}048$ and $\sigma = 80$ hold, obviously we can see that $2^k - 2^\sigma \cdot \lfloor \sqrt{2^{k-\sigma+1}} \rfloor$ is a positive number. Therefore, all signers' 2^σ possible secret keys can be uniformly distributed in the space generated by k.

A.2 Security Against Polynomially-Bounded Adversary

In this section, we define the security against a polynomially-bounded adversary.

A.2.1 Definition of FSMS.LF Game

We define the FSMS.LF game. In this game, there exist a computationally bounded adversary $\tilde{\mathcal{F}}_{pol}$ who colludes with a group of signers and a challenger C corresponding to a target signer. The goal of $\tilde{\mathcal{F}}_{pol}$ is to forge a multisignature. The procedure of the FSMS.LF game is as follows:

Setup : C executes Setup$(1^k, 1^\sigma, 1^{\omega_{\max}})$ to obtain pp. Then, C executes KeyGen to generate $(sk^*, pk^*) = (sk_1, pk_1)$, and then gives pp and pk^* to $\tilde{\mathcal{F}}_{pol}$.

Create User query$(2 \leq j \leq q_c + 1)$: $\tilde{\mathcal{F}}_{pol}$ generates (sk_j, pk_j) and then gives (sk_j, pk_j) to C, where $q_c \leq \omega_{\max} - 1$. C checks if pk_j is the public key corresponding to sk_j. If so, we say that pk_j is registered; otherwise C outputs an error symbol $\perp$.

Sign query$(1 \leq d \leq N_{\max})$: $\tilde{\mathcal{F}}_{pol}$ gives a query m_d to C. C executes MSign with $\tilde{\mathcal{F}}_{pol}$ and returns a multisignature (m_d, S_d, L_d) to $\tilde{\mathcal{F}}_{pol}$.

Output : $\tilde{\mathcal{F}}_{pol}$ outputs $(m^\star, S^\star, L^\star)$, where $L^\star = \{pk_j\}_{j=1}^{q_c+1}$ and $pk_1 = pk^*$. C checks if the following winning conditions hold: MVerify$(\mathsf{pp}, m^\star, S^\star, L^\star)$ outputs 1; all of the public keys except pk^* in $L^\star$ are registered; $m^\star \notin \{m_d\}_{d=1}^{N_{\max}}$; sk^* is used less than $N_{\max}$ times; the number of signers is not over $\omega_{\max}$.

$\tilde{\mathcal{F}}_{pol}$ wins the game if all the winning conditions hold.

Definition 7 We say that a fail-stop multisignature scheme is $(t, q_c, N_{\max}, \epsilon)$-secure if there is no polynomially-bounded adversary $\tilde{\mathcal{F}}_{pol}$ that wins the FSMS.LF game within an execution time t, generating at most q_c ($\leq \omega_{\max} - 1$) Create User queries and $N_{\max}$ Sign queries, and with a success probability greater than ϵ.

A.2.2 Security Proof of FSMS.LF Game

Theorem 7 Any fail-stop multisignature scheme is $(t, q_c, N_{\max}, \epsilon)$-secure if the scheme is (t', q_c', ϵ')-secure for the verifier and secure for the signers with $2^{-\sigma}$, where $t' = t + O(q_c + N_{\max})$, $q_c' = q_c$ and $\epsilon' = \epsilon(1 - 2^{-\sigma})$ hold.

Proof Let $\tilde{\mathcal{F}}_{pol}$ be a polynomially-bounded adversary who can break an FSMS scheme with $(t, q_c, N_{\max}, \epsilon)$. We show how to construct a polynomial time algorithm $\tilde{\mathcal{B}}_{pol}$ which wins the FSMS.LR game with a challenger C. $\tilde{\mathcal{B}}_{pol}$ is given a public parameter pp and generates (sk^*, pk^*) as a pair of keys of a target signer. Then, $\tilde{\mathcal{B}}_{pol}$ sends pk^* to C and runs $\tilde{\mathcal{F}}_{pol}$ with pp and pk^* as input.

Setup : $\tilde{\mathcal{B}}_{pol}$ is given a public parameter pp and generates

(sk^*, pk^*) as a pair of keys of a target signer. Then, $\mathcal{B}_{pol}$ sends pk^* to C and gives pp and pk^* to $\tilde{\mathcal{F}}_{pol}$.

Create User query : $\tilde{\mathcal{F}}_{pol}$ executes the query q_c ($\leq \omega_{\max} - 1$) times and generates secret keys sk_j and their corresponding public keys pk_j where $2 \leq j \leq q_c + 1$. $\tilde{\mathcal{F}}_{pol}$ gives them to $\tilde{\mathcal{B}}_{pol}$. $\tilde{\mathcal{B}}_{pol}$ checks if pk_j is a public key corresponding to sk_j and then registers pk_j; otherwise $\tilde{\mathcal{B}}_{pol}$ outputs an error symbol $\perp$.

Sign Query : $\tilde{\mathcal{F}}_{pol}$ generates a query m_d where $1 \leq d \leq N_{\max}$, and gives it to $\tilde{\mathcal{B}}_{pol}$. $\tilde{\mathcal{B}}_{pol}$ executes $\mathsf{MSign}(\mathsf{pp}, sk^*, m_j, \ell)$ with $\tilde{\mathcal{F}}_{pol}$. $\tilde{\mathcal{B}}_{pol}$ knows sk^* and so can generate it. $\tilde{\mathcal{B}}_{pol}$ outputs S.

Output : $\tilde{\mathcal{F}}_{pol}$ halts, having a signature $(m^\star, S^\star, L^\star)$, and gives $(m^\star, S^\star, L^\star)$ to $\tilde{\mathcal{B}}_{pol}$. $\tilde{\mathcal{B}}_{pol}$ executes $\mathsf{ProveForgery}(\mathsf{pp}, sk^*, m^\star, S^\star, L^\star)$ where $\tilde{\mathcal{B}}_{pol}$ knows all the secret keys in $L^\star$ and so can execute the algorithm as a distribution of the group of the signers. If an output of $\mathsf{ProveForgery}(\mathsf{pp}, sk^*, m^\star, S^\star, L^\star)$ is $\perp$, then $\tilde{\mathcal{B}}_{pol}$ aborts. Otherwise, $\tilde{\mathcal{B}}_{pol}$ obtains $pr^\star$ and sends $(m^\star, S^\star, L^\star, pr^\star)$ to C. Then $\tilde{\mathcal{B}}_{pol}$ can always win the game with C from the following reasons: (1) $\mathsf{MVerify}(\mathsf{pp}, m^\star, S^\star, L^\star)$ outputs 1 from the definition of $\tilde{\mathcal{F}}_{pol}$; (2) $\mathsf{VerifyProof}(\mathsf{pp}, m^\star, S^\star, L^\star, pr^\star)$ outputs 1 since $pr^\star$ is generated via an identical distribution to that of a group of honest signers.

Finally, we evaluate the success probability ϵ' and an execution time t' of $\tilde{\mathcal{B}}_{pol}$. An event that $\tilde{\mathcal{B}}_{pol}$ aborts is in a case in which $\mathsf{ProveForgery}(\mathsf{pp}, sk^*, m^\star, S^\star, L^\star)$ outputs $\perp$. Thus, ϵ' can be obtained as $\epsilon' = \epsilon(1 - 2^{-\sigma})$. Here, σ is a positive number and ϵ' becomes a non-negligible value. Its running time is that of $\tilde{\mathcal{F}}_{pol}$, plus the time required for Setup and Output (both $O(1)$), and to handle $\tilde{\mathcal{F}}_{pol}$'s Create User query ($O(q_c)$) for at most q_c queries and Sign query ($O(N_{\max})$). Thus $t' = t + O(q_c + N_{\max})$ holds. The number of Create User query q'_c of $\tilde{\mathcal{B}}_{pol}$ to C is exactly the same as that of $\tilde{\mathcal{F}}_{pol}$, and thus $q'_c = q_c$ holds. □

A.3 Security of the General Construction of FSMS

We give the security proof of the general construction of FSMS.

A.3.1 Proof of Theorem 3 (Verifier's Security)

First, we prove Theorem 3 corresponding to the *verifier's security*.

Proof Let $\tilde{\mathcal{U}}_{pol}$ be a group of polynomially-bounded malicious signers who can break the general construction of FSMS with (t', q_c, ϵ'). We show how to construct a polynomial time algorithm $\mathcal{B}_{pol}$ which breaks the Pfitzmann's construction with (t'', ϵ''). $\mathcal{B}_{pol}$ is given $(K, h, G, H, \omega_{\max})$ and sets $\mathsf{pp} := (K, h, G, H, \omega_{\max})$. It interacts with $\tilde{\mathcal{U}}_{pol}$ as follows.

Setup : $\mathcal{B}_{pol}$ gives pp to $\tilde{\mathcal{U}}_{\sqrt{\ }\updownarrow}$. $\tilde{\mathcal{U}}_{\sqrt{\ }\updownarrow}$ generates a challenge secret key $sk^* := (sk_{1,1}, sk_{1,2})$ and its corresponding public key $pk^* := (pk_{1,1}, pk_{1,2})$. $\tilde{\mathcal{U}}_{\sqrt{\ }\updownarrow}$ gives pk^* to $\mathcal{B}_{pol}$.

Create User query : $\tilde{\mathcal{U}}_{pol}$ executes the query q_c ($\leq \omega_{\max} - 1$) times and generates secret keys $(sk_{j,1}, sk_{j,2})$ and their corresponding public keys $(pk_{j,1}, pk_{j,2})$ where $2 \leq j \leq q_c + 1$. $\tilde{\mathcal{U}}_{\sqrt{\ }\updownarrow}$ gives them to $\mathcal{B}_{pol}$. $\mathcal{B}_{pol}$ checks that $(pk_{j,1}, pk_{j,2}) = (h(sk_{j,1}), h(sk_{j,2}))$ holds; otherwise $\mathcal{B}_{pol}$ outputs the error symbol $\perp$ [*8].

Output : $\tilde{\mathcal{U}}_{pol}$ halts, having output two signatures $(S^\star, pr^\star)$ such that $S^\star \neq pr^\star$ and $h(S^\star) = h(pr^\star)$ hold on a message $m^\star$, and gives $(S^\star, pr^\star)$ to $\mathcal{B}_{pol}$. $\mathcal{B}_{pol}$ sets $s^\star_{1,1} \leftarrow S^\star - \sum_{j=2}^{q_c+1} s_j$ and $s^\star_{1,2} \leftarrow pr^\star - \sum_{j=2}^{q_c+1} s_j$, where s_j is a partial signature generated from the secret key $(sk_{j,1}, sk_{j,2})$ and $\mathcal{B}_{pol}$ can know these values via Create User query. Then we have

$$
\begin{aligned}
h(s^\star_{1,1}) &= h\left(S^\star - \sum_{j=2}^{q_c+1} s_j\right) \\
&= \prod_{j=1}^{q_c+1}\left(pk_{j,1} \times pk_{j,2}^{m^\star}\right)\prod_{j=2}^{q_c+1}\left(pk_{j,1} \times pk_{j,2}^{m^\star}\right)^{-1} \\
&= pk_{1,1} \times pk_{1,2}^{m^\star}. \\
h(s^\star_{1,2}) &= h\left(pr^\star - \sum_{j=2}^{q_c+1} s_j\right) \\
&= \prod_{j=1}^{q_c+1}\left(pk_{j,1} \times pk_{j,2}^{m^\star}\right)\prod_{j=2}^{q_c+1}\left(pk_{j,1} \times pk_{j,2}^{m^\star}\right)^{-1} \\
&= pk_{1,1} \times pk_{1,2}^{m^\star}.
\end{aligned}
$$

Thus $s^\star_{1,1}$ and $s^\star_{1,2}$ ($s^\star_{1,1} \neq s^\star_{1,2}$) are acceptable signatures under the challenge public key pk^*. $\mathcal{B}_{pol}$ halts, having output $s^\star_{1,1}$ as a forged signature and $s^\star_{1,2}$ as a proof of forgery.

$\mathcal{B}_{pol}$ succeeds whenever $\tilde{\mathcal{U}}_{pol}$ does, and thus $\epsilon'' = \epsilon'$ holds. Its running time is that of $\tilde{\mathcal{U}}_{pol}$, plus the time required for Setup and Output (both $O(1)$), and to handle $\tilde{\mathcal{U}}_{pol}$'s Create User query ($O(q_c)$) for at most q_c queries. Thus $t'' = t' + O(q_c)$ and $\epsilon'' = \epsilon'$ hold. □

A.3.2 Proof of Theorem 4 (Signer's Security)

Next, we prove Theorem 4 corresponding to the *signer's security*.

Proof Let $\tilde{\mathcal{F}}_{exp}$ be an adversary who can output an unprovable forgery in FSMS with a probability greater than $T_{\max}/2^\tau$. $\tilde{\mathcal{F}}_{exp}$ outputs a forged multisignature $S^\star \in G$ on some message $m^\star$, and sets $s^\star_1 \leftarrow S^\star - \sum_{j=2}^{q_c+1} s_j$, where $s_j \in G$ is the partial signature generated from the secret key $(sk_{j,1}, sk_{j,2})$ and $\tilde{\mathcal{F}}_{exp}$ can know these values via Create User query. Then we have

$$
\begin{aligned}
h(s^\star_1) &= h\left(S^\star - \sum_{j=2}^{q_c+1} s_j\right) \\
&= \prod_{j=1}^{q_c+1}\left(pk_{j,1} \times pk_{j,2}^{m^\star}\right)\prod_{j=2}^{q_c+1}\left(pk_{j,1} \times pk_{j,2}^{m^\star}\right)^{-1} \\
&= pk_{1,1} \times pk_{1,2}^{m^\star}.
\end{aligned}
$$

Thus $\tilde{\mathcal{F}}_{exp}$ can output an unprovable forgery in the Pfitzmann's construction with a probability greater than $T_{\max}/2^\tau$. □

A.4 Security of the Proposed FSMS Scheme

In this section, we give the security proof of the proposed FSMS scheme.

[*8] Note that, as mentioned in Section 3.3.1, the signing query is not used in the proof since the adversary is not allowed to use the signing query.

A.4.1 Proof of Theorem 5 (Verifier's Security)

First, we prove Theorem 5 corresponding to the *verifier's security*.

Proof Let $\tilde{\mathcal{U}}_{pol}$ be a group of polynomially-bounded malicious signers who can break the proposed FSMS scheme with (t, q_c, ϵ). We show how to construct a polynomial time algorithm $\mathcal{A}_{\mathsf{FACT}}$ which outputs (p, q) for a given $(1^k, n, a)$ with (t', ϵ'). $\mathcal{A}_{\mathsf{FACT}}$ is given $(n, a, h_n, \mathbb{Z}_n^*, \omega_{\max})$ and sets $\mathsf{pp} := (n, a, h_n, \mathbb{Z}_n^*, \omega_{\max})$. It interacts with $\tilde{\mathcal{U}}_{pol}$ as follows.

Setup : $\mathcal{A}_{\mathsf{FACT}}$ gives pp to $\tilde{\mathcal{U}}_{pol}$. $\tilde{\mathcal{U}}_{pol}$ generates a challenge secret key $sk^* := (sk_{1,1}, sk_{1,2})$ and its corresponding challenge public key $pk^* := (pk_{1,1}, pk_{1,2})$. $\tilde{\mathcal{U}}_{pol}$ gives pk^* to $\mathcal{A}_{\mathsf{FACT}}$.

Create User query : $\tilde{\mathcal{U}}_{pol}$ executes the query $q_c (\leq \omega_{\max} - 1)$ times and generates secret keys $(sk_{j,1}, sk_{j,2})$ and their corresponding public keys $(pk_{j,1}, pk_{j,2})$ where $2 \leq j \leq q_c + 1$. $\tilde{\mathcal{U}}_{pol}$ gives them to $\mathcal{A}_{\mathsf{FACT}}$. $\mathcal{A}_{\mathsf{FACT}}$ checks that $(pk_{j,1}, pk_{j,2}) = (sk_{j,1}^a \bmod n, sk_{j,2}^a \bmod n)$ holds; otherwise $\mathcal{A}_{\mathsf{FACT}}$ outputs the error symbol $\perp$.

Output : $\tilde{\mathcal{U}}_{pol}$ halts, having output two signatures $(S^\star, pr^\star)$ such that $S^\star \neq pr^\star$ and $S^{\star a} = (pr^\star)^a \bmod n$ hold on a message $m^\star$, and gives $(S^\star, pr^\star)$ to $\mathcal{A}_{\mathsf{FACT}}$. Since $|p| = |q|$ holds, it must be $q > a$. Hence, $a \bmod q - 1 = a$ holds. Therefore, $S^{\star a} = (pr^\star)^a \bmod q$ can be obtained. Since $\mathsf{gcd}(a, q - 1) = 1$ holds, a is invertible modulo $q - 1$, and it results that $pr^\star = S^\star + \psi \cdot q$ holds for $1 \leq \psi < p$. Hence, we conclude $\mathsf{gcd}(pr^\star - S^\star, q) = q$. Therefore, it is sufficient for $\mathcal{A}_{\mathsf{FACT}}$ to compute $q = \mathsf{gcd}(pr^\star - S^\star, n)$, and then deduce $p = n/q$.

$\mathcal{A}_{\mathsf{FACT}}$ succeeds whenever $\tilde{\mathcal{U}}_{pol}$ does, and thus $\epsilon' = \epsilon$ holds. Its running time is that of $\tilde{\mathcal{U}}_{pol}$, plus the time required for two exponentiations $2t_e$ during Setup phase, and to handle $\tilde{\mathcal{U}}_{pol}$'s Create User query: two exponentiations for at most q_c queries, i.e., $2q_c t_e$. Thus $t' = t + 2t_e(1 + q_c)$ and $\epsilon' = \epsilon$ hold. □

A.4.2 Proof of Theorem 6 (Signer's Security)

Next, we prove Theorem 6 corresponding to the *signer's security*.

Proof To prove this theorem, we prove that, under the factorization assumption with a-strong primes, the construction above is a bundling homomorphism with bundling degree 2^σ firstly. Obviously we can see that $h_n : x \mapsto x^a \bmod n$ is a group homomorphism for $x \in \mathbb{Z}_n^*$. Let $\mathsf{CRT} : \mathbb{Z}_n^* \to \mathbb{Z}_p^* \times \mathbb{Z}_q^*$ denotes the isomorphism induced by the Chinese remainder theorem. By applying CRT, if S is a solution to the equation $S^a = y \bmod n$, then $(S_p, S_q) = (S \bmod p, S \bmod q)$ is a solution of the following equations: $(S_p)^a = y \bmod p \cdots (1)$ and $(S_q)^a = y \bmod q \cdots (2)$. In order to prove that $S^a \in \mathbb{Z}_n^*$ has the multiple a-th roots, we recall the following theorem which is proved by Frobenius.

Theorem 8 (Frobenius [9]) If a divides the order of a group, then the number of elements in the group whose order divides a is a multiple of a. If the group is cyclic, then this number is exactly a.

On one hand, since $\mathbb{Z}_p^*$ is cyclic and a divides the order of $\mathbb{Z}_p^*$ (i.e., $\varphi(p) = 2ap'$), Eq. (1) has exactly a solutions. On the other hand, since $\mathsf{gcd}(a, q - 1) = 1$ holds, there must exist a'_q such that $a \cdot a'_q \equiv 1 \bmod q - 1$ holds. Thus Eq. (2) can be rewritten as $S_q = y^{a'_q} \bmod q$ and admits one unique solution. As there exist a tuples of the form (S_p, S_q) that satisfy Eq. (1) and Eq. (2), by applying CRT^{-1}, we deduce that the number of a-th roots of y in $\mathbb{Z}_n^*$ is exactly a. Note that every $x^a \in \mathbb{Z}_n^*$ has a solutions under this setting. Setting $y = 1$ trivially leads to the kernel of the homomorphism. The kernel is thus of size a which is lower-bounded by 2^σ.

Next, we show that taking bundling homomorphism with degree 2^σ is sufficient to achieve the security against $\tilde{\mathcal{F}}_{exp}$. To do so, we must analyse the size of $T_{\max}$ and show that $T_{\max}/2^\sigma \leq 2^{-\sigma}$ holds. This policy will not change even in the proposed FSMS scheme since there are 2^σ possible secret keys for a targeted signer, and he can provide valid proof of forgery S if the forged signature $s_1^\star$ on a message $m^\star$ differs from his own signature s_1 on $m^\star$. Since $\mathsf{gcd}(m', a) = 1$ holds, we have

$$
\begin{aligned}
T_{\max} &= \max_{m' \in \mathcal{M} \setminus \{0\}} |\{d \in \mathbb{Z}_n^* : h_n(d) = 1 \wedge d^{m'} = 1\}| \\
&= \max_{m' \in \mathcal{M} \setminus \{0\}} |\{d \in \mathbb{Z}_n^* : d^a = 1 \wedge d^{m'} = 1\}| \\
&= \max_{m' \in \mathcal{M} \setminus \{0\}} |\{d \in \mathbb{Z}_n^* : (\mathsf{ord}_{\mathbb{Z}_n^*}(d) | a) \wedge (\mathsf{ord}_{\mathbb{Z}_n^*}(d) | m')\}| \\
&= \max_{m' \in \mathcal{M} \setminus \{0\}} |\{d \in \mathbb{Z}_n^* : \mathsf{ord}_{\mathbb{Z}_n^*}(d) | \mathsf{gcd}(a, m')\}| \\
&= 1.
\end{aligned}
$$

Thus, we can conclude that the proposed FSMS scheme is secure for the signer with a probability $2^{-\sigma}$. □

A.5 Fail-Stop Aggregate Signatures

In this section, we define the model and the security of the FSAS scheme. Moreover, we give a general construction and an instantiation of FSAS.

A.5.1 Model of FSAS Scheme

We define the model of an FSAS scheme.

Definition 8 (Fail-Stop Aggregate Signatures). A fail-stop aggregate signature scheme is a tuple of algorithms (*Setup, KeyGen, Sign, Aggregation, AggVerify, ProveForgery, VerifyProof*) such that:

$\mathsf{Setup}(1^k, 1^\sigma, 1^{\omega_{\max}}) \to \mathsf{pp}$: This is a probabilistic algorithm that, on input of security parameters $(k, \sigma, \omega_{\max})$, outputs a public parameter pp.

$\mathsf{KeyGen}(\mathsf{pp}, 1^{N_{\max}}) \to (sk_j, pk_j)$: This is a probabilistic algorithm that, on input of pp and an integer $N_{\max}$, outputs a secret/public key pair (sk_j, pk_j) which can be used for signing $N_{\max}$ times. For the sake of simplicity, the input $1^{N_{\max}}$ is omitted when the scheme is used as a one-time signature scheme.

$\mathsf{Sign}(\mathsf{pp}, sk_j, m_j, \ell) \to (m_j, s_j)$: This is an (probabilistic) algorithm that, on input of pp, sk_j, a message m, and a counter ℓ incremented at each invocation of this algorithm, outputs a tuple (m_j, s_j) where s_j is a signature.

$\mathsf{Aggregation}(\mathsf{pp}, \{m_j, s_j, pk_j\}_{j=1}^i) \to \{(\{m_j\}_{j=1}^i, S, L), \perp\}$: This is a deterministic algorithm that, on input of pp, i tuples (m_j, s_j, pk_j) of messages, signatures and public keys, outputs a tuple $(\{m_j\}_{j=1}^i, S, L)$ where S is an aggregate signature and L is a set $\{pk_j\}_{j=1}^i$ of i signers. If $i > \omega_{\max}$, the output is $\perp$.

$\mathsf{AggVerify}(\mathsf{pp}, \{m_j\}_{j=1}^i, S, L) \to \{0, 1\}$: This is a deterministic (possibly probabilistic) algorithm that, on input of pp, $\{m_j\}_{j=1}^i$, S, and L, outputs 0 or 1.

ProveForgery(pp, sk_j, $\{m_j^\star\}_{j=1}^i$, $S^\star$, $L^\star$) $\rightarrow$ $\{pr, \perp\}$: This is an (probabilistic) interactive algorithm for generating a proof of forgery. Given pp, sk_j, $\{m_j^\star\}_{j=1}^i$, $S^\star$, and $L^\star$, it outputs a bit string pr as a proof of forgery or $\perp$ in a case of failure.

VerifyProof(pp, $\{m_j^\star\}_{j=1}^i$, $S^\star$, $L^\star$, pr) $\rightarrow$ $\{0, 1\}$: This is a deterministic (possibly probabilistic) algorithm that, on input of pp, $\{m_j^\star\}_{j=1}^i$, $S^\star$, $L^\star$, and pr, outputs 0 or 1.

We say that an aggregate signature S is *acceptable* on messages $\{m_j\}_{j=1}^i$ if we have AggVerify(pp, $\{m_j\}_{j=1}^i$, S, L) $\rightarrow$ 1. Moreover, we say that a proof pr is *valid* on messages $\{m_j^\star\}_{j=1}^i$ if we have VerifyProof(pp, $\{m_j^\star\}_{j=1}^i$, $S^\star$, $L^\star$, pr) $\rightarrow$ 1. Here, ProveForgery is an interactive algorithm and each signer runs these algorithms with the inputs described above. Similarly to the model in Section 3.2, we assume that the signers are connected to each other via point-to-point links over which they can send a message, and the algorithm takes the set L in order to generate a proof of forgery.

We require for an FSAS scheme to satisfy the following conditions as correctness.

Definition 9 (Correctness of Fail-Stop Aggregate Signatures)

(1) Every honestly generated aggregate signature is acceptable, i.e., for any $k, \sigma, \omega_{\max}, N_{\max} \in \mathbb{N}$ and messages $\{m_j\}_{j=1}^i \in \mathcal{M}$, we have

$$\Pr[\mathsf{AggVerify}(\mathsf{pp}, \{m_j\}_{j=1}^i, S, L) = 1] = 1,$$

where pp $\leftarrow$ Setup($1^k, 1^\sigma, 1^{\omega_{\max}}$), $(sk_j, pk_j) \leftarrow$ KeyGen(pp, $1^{N_{\max}}$), $(m_j, s_j) \leftarrow$ Sign(pp, sk_j, m_j, ℓ), and $(\{m_j\}_{j=1}^i, S, L) \leftarrow$ Aggregation(pp, $\{m_j, s_j, pk_j\}_{j=1}^i$).

(2) Every honestly generated proof of forgery is valid, i.e., for any $k, \sigma, \omega_{\max}, N_{\max} \in \mathbb{N}$, messages $\{m_j^\star\}_{j=1}^i \in \mathcal{M}$ and tuple $(S^\star, L^\star)$, we have

$$\Pr[\mathsf{VerifyProof}(\mathsf{pp}, \{m_j^\star\}_{j=1}^i, S^\star, L^\star, pr) = 1 \mid \mathsf{AggVerify}(\mathsf{pp}, \{m_j^\star\}_{j=1}^i, S^\star, L^\star) = 1, pr \neq \perp] = 1,$$

where pp $\leftarrow$ Setup($1^k, 1^\sigma, 1^{\omega_{\max}}$), $(sk_j, pk_j) \leftarrow$ KeyGen(pp, $1^{N_{\max}}$), and $pr \leftarrow$ ProveForgery(pp, sk_j, $\{m_j^\star\}_{j=1}^i$, $S^\star$, $L^\star$).

A.5.2 ACMA&FSAS-AIA

We define the adaptive-chosen-message attack and fail-stop-aggregate-signatures' adaptive-insider attack (ACMA&FSAS-AIA). Its construction is almost the same as that in Section 3.3, and we give the details below. In the following games, we represent the target signer's information by using a symbol $*$.

A.5.2.1 FSAS.LR Game

We define the FSAS.LR game. In this game, there exists a group of polynomially-bounded malicious signers $\tilde{\mathcal{U}}_{pol}$ and a challenger C corresponds to a verifier. The goal of $\tilde{\mathcal{U}}_{pol}$ is to repudiate their signature by computing a valid proof of forgery. Note that the signing query does not need to be defined since the adversary in this game is a group of malicious signers. Thus there is no signing oracle. The procedure of the FSAS.LR game is as follows:

Setup : C executes Setup($1^k, 1^\sigma, 1^{\omega_{\max}}$) to output pp. Then, $\tilde{\mathcal{U}}_{pol}$ generates $(sk^*, pk^*) = (sk_1, pk_1)$, and gives pk^* to C.

Create User query : $(2 \leq j \leq q_c+1)$. $\tilde{\mathcal{U}}_{pol}$ generates (sk_j, pk_j) and then gives (sk_j, pk_j) to C, where $q_c \leq \omega_{\max} - 1$. C checks if pk_j is the public key corresponding to sk_j. If so, we say that pk_j is registered; otherwise C outputs an error symbol $\perp$.

Output : $\tilde{\mathcal{U}}_{pol}$ outputs $(\{m_j^\star\}_{j=1}^i, S^\star, L^\star, pr^\star)$, where $L^\star = \{pk_j\}_{j=1}^{q_c+1}$ and $pk_1 = pk^*$. C checks if all the following winning conditions hold: AggVerify(pp, $\{m_j^\star\}_{j=1}^i$, $S^\star$, $L^\star$) outputs 1; all of the public keys except pk^* in $L^\star$ are registered; VerifyProof(pp, $\{m_j^\star\}_{j=1}^i$, $S^\star$, $L^\star$, $pr^\star$) outputs 1; the number of signers is not over $\omega_{\max}$.

$\tilde{\mathcal{U}}_{pol}$ wins the game if all the winning conditions hold. The following definition corresponds to the *verifier's security*.

Definition 10 We say that a fail-stop aggregate signature scheme is (t, q_c, ϵ)-secure for the verifier if there is no group of polynomially-bounded malicious signers $\tilde{\mathcal{U}}_{pol}$ that wins the FSAS.LR game within an execution time t, generating at most q_c ($\leq \omega_{\max} - 1$) Create User queries, and with a success probability greater than ϵ.

A.5.2.2 FSAS.FF Game

We define the FSAS.FF game. In this game, there exists a computationally unbounded adversary $\tilde{\mathcal{F}}_{exp}$ who colludes with a group of signers, and a challenger C corresponds to a target signer. The goal of $\tilde{\mathcal{F}}_{exp}$ is to forge an aggregate signature for which C cannot generate a proof of forgery. The procedure of the FSAS.FF game is as follows:

Setup : C executes Setup($1^k, 1^\sigma, 1^{\omega_{\max}}$) to obtain pp. Then C executes KeyGen to generate $(sk^*, pk^*) = (sk_1, pk_1)$, and then gives pp and pk^* to $\tilde{\mathcal{F}}_{exp}$.

Create User query : $(2 \leq j \leq q_c + 1)$. $\tilde{\mathcal{F}}_{exp}$ generates (sk_j, pk_j) and then gives (sk_j, pk_j) to C, where $q_c \leq \omega_{\max} - 1$. C checks if pk_j is the public key corresponding to sk_j. If so, we say that pk_j is registered; otherwise C outputs an error symbol $\perp$.

Sign query : $(1 \leq d \leq N_{\max})$. $\tilde{\mathcal{F}}_{exp}$ gives a query m_d to C. C executes Sign and outputs a signature (m_d, s_d, L_d).

Output : $\tilde{\mathcal{F}}_{exp}$ outputs $(\{m_j^\star\}_{j=1}^i, S^\star, L^\star)$, where $L^\star = \{pk_j\}_{j=1}^{q_c+1}$ and $pk_1 = pk^*$. C checks if the following winning conditions hold: AggVerify(pp, $\{m_j^\star\}_{j=1}^i$, $S^\star$, $L^\star$) outputs 1; all of the public keys except pk^* in $L^\star$ are registered; ProveForgery(pp, sk^*, $\{m_j^\star\}_{j=1}^i$, $S^\star$, $L^\star$) outputs $\perp$; $m_1^\star \notin \{m_d\}_{d=1}^{N_{\max}}$; sk^* is used less than $N_{\max}$ times; the number of signers is not over $\omega_{\max}$.

$\tilde{\mathcal{F}}_{exp}$ wins the game if all the winning conditions hold. The following definition corresponds to the *signer's security*.

Definition 11 We say that a fail-stop aggregate signature scheme is secure for the signers with $2^{-\sigma}$ if there is no computationally unbounded adversary $\tilde{\mathcal{F}}_{exp}$ who wins the FSAS.FF game with a probability greater than $2^{-\sigma}$, where σ is a security parameter.

A.5.3 General Construction of FSAS

The general construction of FSAS consists of the following algorithms:

Setup$(1^k, 1^\sigma, 1^{\omega_{\max}})$: As a prekey, it picks random index K for a bundling homomorphism h_η and sets $h := h_K, G := G_K$ and $H := H_K$. It outputs pp $:= (K, h, G, H, \omega_{\max})$.

KeyGen(pp) [*9] : Firstly, it checks if h is a group homomorphism with bundling degree 2^τ (e.g., using zero-knowledge proof) [*10]. If not, it executes *Setup* algorithm to obtain such a bundling homomorphism. Once it is convinced, it outputs a secret key as $sk_j = (sk_{j,1}, sk_{j,2}) \in_R G^2$ and its corresponding public key as $pk_j = (pk_{j,1}, pk_{j,2}) = (h(sk_{j,1}), h(sk_{j,2})) \in H^2$.

Sign(pp, sk_j, m_j) [*11] : The message space $\mathcal{M}$ is given as a subset of $\mathbb{Z}$. It computes a signature $s_j \in G$ as $s_j = sk_{j,1} + m_j sk_{j,2}$ and outputs (m_j, s_j).

Aggregation(pp, $\{m_j, s_j, pk_j\}_{j=1}^i$) : It computes an aggregate signature $S \in G$ as $S = \sum_{j=1}^i s_j$ with respect to the public keys on $L = \{pk_j\}_{j=1}^i$. If $i > \omega_{\max}$, then it outputs $\perp$. Otherwise, it outputs a tuple of $(\{m_j\}_{j=1}^i, S, L)$.

AggVerify(pp, $\{m_j\}_{j=1}^i, S, L$) : It outputs 1 if $h(S) = \prod_{j=1}^i (pk_{j,1} \times pk_{j,2}^{m_j})$ holds. Otherwise, it outputs 0.

ProveForgery(pp, $sk_j, \{m_j^\star\}_{j=1}^i, S^\star, L^\star$) : By invoking Sign and Aggregation, it computes a partial signature $s_j \in G$ as $s_j = sk_{j,1} + m_j^\star sk_{j,2}$, and then obtains $pr = \sum_{j=1}^i s_j$. If either $h(S^\star) \neq \prod_{j=1}^i (pk_{j,1} \times pk_{j,2}^{m_j^\star})$ or $S^\star = pr$ holds, then it outputs $\perp$. Otherwise, it outputs pr as a proof of forgery.

VerifyProof(pp, $\{m_j^\star\}_{j=1}^i, S^\star, L^\star, pr$) : It outputs 1 if $S^\star \neq pr$ and $h(S^\star) = h(pr)$ hold. Otherwise, it outputs 0.

We can prove the following theorems. Their proofs are almost the same as those in Appendix A.3 and so we omit the details of the proofs.

Theorem 9 The general construction of fail-stop aggregate signature is (t', q_c, ϵ')-secure for the verifier if the Pfitzmann's general construction of fail-stop signature is (t'', ϵ'')-secure. Here, $t'' = t' + O(q_c)$, $q_c \leq \omega_{\max} - 1$ and $\epsilon'' = \epsilon'$ hold.

Theorem 10 The general construction of fail-stop aggregate signature is secure for the signer with a probability $T_{\max}/2^\tau$.

A.5.4 Instantiation of Fail-Stop Aggregate Signatures

In this section, we propose an efficient FSAS scheme constructed from our general construction. Similarly to the proposed scheme in Section 5, our construction is based on the MO function $h(x) = x^a \bmod n$ [20]. Our concrete construction consists of the following algorithms:

Setup$(1^k, 1^\sigma, 1^{\omega_{\max}})$: It invokes Gen (in Section 2.2), and Gen chooses a σ-bit prime integer a and a prime p' such that $p' > 2a$. Gen computes a prime p as $p = 2ap' + 1$ and then chooses a prime q such that $|p| = |q|$, where $\gcd(q-1, a) = 1$ holds. Then, Gen computes $n = pq$ and outputs (n, a, p, q). At this point, the setup algorithm defines a group homomorphism h_n as $h_n : x \mapsto x^a \bmod n$ for $x \in \mathbb{Z}_n^*$, and finally outputs pp $:= (n, a, h_n, \mathbb{Z}_n^*, \omega_{\max})$.

KeyGen(pp) [*12] : To verify that the prekey is correctly generated, the signer utilizes a zero-knowledge proof that a indeed divides $\varphi(n)$ (such a proof can be constructed from general zero-knowledge proofs [11]). If not, it executes Setup algorithm to obtain such a pp. Otherwise, it outputs a secret key as $sk_j = (sk_{j,1}, sk_{j,2}) \in_R \mathbb{Z}_n^{*2}$ and its corresponding public key as $pk_j = (pk_{j,1}, pk_{j,2}) = (sk_{j,1}^a \bmod n, sk_{j,2}^a \bmod n)$.

Sign(pp, sk_j, m_j) [*13] : The message space $\mathcal{M}$ is defined as $\mathcal{M} := \mathbb{Z}_a$. It computes a signature $s_j \in \mathbb{Z}_n^*$ as $s_j = sk_{j,1} sk_{j,2}^{m_j} \bmod n$ and outputs (m_j, s_j).

Aggregation(pp, $\{m_j, s_j, pk_j\}_{j=1}^i$) : Given $\{s_j\}_{j=1}^i$, it computes an aggregate signature $S \in \mathbb{Z}_n^*$ as $S = \prod_{j=1}^i s_j \bmod n$ with respect to the public keys on $L = \{pk_j\}_{j=1}^i$. If $i > \omega_{\max}$, then it outputs $\perp$. Otherwise, it outputs $(\{m_j\}_{j=1}^i, S, L)$.

AggVerify(pp, $\{m_j\}_{j=1}^i, S, L$) : It outputs 1 if $S^a = \prod_{j=1}^i (pk_{j,1} \times pk_{j,2}^{m_j}) \bmod n$ holds. Otherwise, it outputs 0.

ProveForgery(pp, $sk_j, \{m_j^\star\}_{j=1}^i, S^\star, L^\star$) : By invoking Sign and Aggregation, it computes a partial signature $s_j \in \mathbb{Z}_n^*$ as $s_j = sk_{j,1} sk_{j,2}^{m_j^\star} \bmod n$ and then obtains $pr = \prod_{j=1}^i s_j \bmod n \in \mathbb{Z}_n^*$. If either $(S^\star)^a \neq \prod_{j=1}^i (pk_{j,1} \times pk_{j,2}^{m_j^\star}) \bmod n$ or $S^\star = pr$ holds, then it outputs $\perp$. Otherwise, it outputs pr as a proof of forgery.

VerifyProof(pp, $\{m_j^\star\}_{j=1}^i, S^\star, L^\star, pr$) : It outputs 1 if $S^\star \neq pr$ and $S^{\star a} = (pr)^a \bmod n$ hold; otherwise it outputs 0.

We can prove the following theorems. Their proofs are almost the same as those in Appendix A.4, and we omit the details of the proofs.

Theorem 11 The proposed fail-stop aggregate signature scheme is (t, q_c, ϵ)-secure for the verifier if (t', ϵ')-factorization assumption with a-strong primes holds against a probabilistic polynomial time algorithm $\mathcal{A}_{\mathsf{FACT}}$ with respect to Gen, where $t' = t + 2t_e(1 + q_c)$, $q_c (\leq \omega_{\max} - 1)$, $\epsilon' = \epsilon$ and t_e is a computational time for one exponentiation.

Theorem 12 The proposed fail-stop aggregate signature scheme is secure for the signer with a probability $2^{-\sigma}$.

A.6 Cryptanalysis of the Original MO11 Scheme

In this section, we briefly review the original MO11 scheme [20] firstly, and then show that it is not secure for a signer. Moreover, we propose a countermeasure against the vulnerability.

A.6.1 Review of the Original MO11 Scheme

We briefly review the original MO11 scheme. In this scheme, they used a generic Rabin function, $h(x) = x^a \bmod n$, whose invertibility is equivalent to the problem of factoring the integer n, where a is a σ-bit *odd integer*. First, a verifier (or a trusted center) chooses a prime p' such that $p' > 2a$. Second, it computes $p = 2ap' + 1$, and checks if p is also prime. Finally, it chooses a prime q such that $|p| = |q|$, and computes $n = pq$. A prekey is (a, n), and each signer and the verifier execute the following

[*9] Note that the input $1^{N_{\max}}$ is omitted since the scheme we describe is used as a one-time signature scheme.

[*10] We note that this is provable via a general zero-knowledge proof for the NP-language.

[*11] Note that, similarly as mentioned on KeyGen, a counter ℓ is omitted of the input since we describe the scheme is used as a one-time signature scheme.

[*12] Note that the input $1^{N_{\max}}$ is omitted since the scheme we describe is used as a one-time signature scheme.

[*13] Note that, similarly as mentioned on KeyGen, a counter ℓ is omitted of the input since the scheme we describe is used as a one-time signature scheme.

algorithms:

KeyGen(K) : Generate a secret key as $(sk_1, sk_2) \in_R \mathbb{Z}_n^{*2}$ and compute its corresponding public key $(pk_1, pk_2) = (sk_1^a \bmod n, sk_2^a \bmod n)$.

Sign(sk, m) : The message space is defined to be $\mathcal{M} = \{1, \cdots, \varphi(n) - 1\} \backslash \{x > 1 | \gcd(a, x) \neq 1\}$, and $\varphi(n) = (p-1)(q-1)$. To sign a message $m \in \mathcal{M}$, compute a signature s as $s = sk_1 sk_2^m \bmod n$.

Verify(pk, m, s) : Output 1 if $s^a = pk_1 pk_2^m \bmod n$ holds. Otherwise, output 0.

ProveForgery($sk, m^\star, s^\star$) : Given a forgery $(m^\star, s^\star)$, compute a signature pr on the message $m^\star$ using the secret key (sk_1, sk_2). Output $\perp$ if either $(s^\star)^a \neq pk_1 pk_2^{m^\star} \bmod n$ or $s^\star = pr$ holds. Otherwise, output pr as a proof of forgery.

VerifyProof($pk, m^\star, s^\star, pr$) : Given two signatures $s^\star$ and pr on a same message $m^\star$, output 1 if $pr \neq s^\star$ and $pr^a = s^{\star a} \bmod n$ both hold. Otherwise, output 0.

A.6.2 How to Break the Original MO11 Scheme

We show that the original MO11 scheme is not secure for a signer. Firstly, an adversary obtains a signature $s = sk_1 sk_2^m \bmod n$ for an arbitrary $m \in \mathcal{M}$ from an honest signer by using the signing query. Next, the adversary defines the message $m^\star$ as $m^\star = m + ax$ for an arbitrary $x \in \mathbb{Z}$. Here, $m^\star \in \mathcal{M}$ and $m^\star \neq m$ hold. Finally, the adversary outputs a forged signature $s^\star$ as $s^\star = s \times pk_2^x$. Then, we have the following lemma:

Lemma 1 Define $m^\star = m + ax$ and $s^\star = s \times pk_2^x$. Then, $s^\star$ is an unprovable forgery on $m^\star$.

Proof For any $x \in \mathbb{Z}$, the adversary defines $m^\star$ as $m^\star = m + ax$ and $s^\star$ as $s^\star = s \times pk_2^x$. Then, the following equation holds;

$$\begin{aligned}\left(s^\star\right)^a &= (s \times pk_2^x)^a = s^a \times pk_2^{ax} \\ &= pk_1 pk_2^m pk_2^{ax} = pk_1 pk_2^{m+ax} = pk_1 pk_2^{m^\star} \bmod n.\end{aligned}$$

This equation means that the adversary outputs a new signature on $m^\star$. In this case, the honest signer tries to generate a proof of forgery pr on $m^\star$ by using ProveForgery. However, it returns only $\perp$ since pr becomes the same value $s^\star$. □

Consequently, we have the following theorem:

Theorem 13 The original MO11 scheme is not secure for the signer.

Remark: A possible countermeasure is to adopt a *prime number a* as a component of the prekey, and to reduce the message space $\mathcal{M}$ to $\mathbb{Z}_a$ as we do in our schemes. Unfortunately, the message space is restricted to 80 bits for 80-bit security (i.e., $\sigma = 80$) due to this countermeasure. This is not very practical. However, just by setting $\sigma = 160$, we can still use 2048-bit moduli. Then we can compress the message by using a collision resistant hash function when we sign the message larger than 160 bits.

A.7 Cryptographic Accumulators

In this section, we briefly review the collision-free accumulators [1] by using the works of Fazio and Nicolosi [8], and Mashatan and Ouafi [21] as references.

Basically, by using cryptographic accumulators, we can *accumulate* a large number of values into a single one. The concept of accumulators was firstly proposed by Benaloh and de Mare [3]. Later, Barić and Pfitzmann [1] defined and constructed a collision-free subtype. Moreover, as an application, they showed an FSS scheme in which many one-time public keys are accumulated into a short public key. As a result, they showed that the size of both public key and the signatures can be independent of the number of messages that can be signed.

The following definition is more general than that of Ref. [3] since it does not require the quasi-commutative hash function (i.e., a one-way hash function f that satisfies $f(f(x, y_1), y_2) = f(f(x, y_2), y_1)$) [3].

Definition 12 (Collision-Free Accumulators) A collision-free accumulator scheme is a tuple of algorithms (*AccKeyGen, Eval, Wit, Ver*) such that:

$\mathsf{AccKeyGen}(1^\lambda, 1^{N'_{\max}}) \to K_{\mathsf{acc}}$: This is a probabilistic algorithm that, on input of security parameter λ and an accumulation threshold $N'_{\max}$ (i.e., an upper bound on the total number of values that can be securely accumulated), outputs an accumulator key K_{acc}, which is public parameter, from a key space $\mathcal{K}_{\lambda, N'_{\max}}$.

$\mathsf{Eval}(K_{\mathsf{acc}}, Y) \to (z_{\mathsf{acc}}, \mathsf{aux})$: This is a deterministic algorithm that, on input of K_{acc} and a set $Y := \{y_1, \ldots, y_{N_{\max}}\}$ of $N_{\max} \leq N'_{\max}$ elements from an efficiently-samplable domain $\mathcal{Y}_{K_{\mathsf{acc}}}$, outputs an accumulated value $z_{\mathsf{acc}} \in \mathcal{Z}_{K_{\mathsf{acc}}}$ and some auxiliary information aux. Note that every execution of Eval on the same input (K_{acc}, Y) must yield the same value z_{acc}, whereas the auxiliary information aux can differ. After its execution, the value z_{acc} is made public.

$\mathsf{Wit}(K_{\mathsf{acc}}, z_{\mathsf{acc}}, y_r, \mathsf{aux}) \to \{w_r, \perp\}$: This is a deterministic algorithm that, on input of K_{acc}, z_{acc}, a value $y_r \in \mathcal{Y}_{K_{\mathsf{acc}}}$ and aux, outputs either a witness w_r from an efficiently-samplable witness space $\mathcal{W}_{K_{\mathsf{acc}}}$ that proves that y_r is accumulated within z_{acc}, or an error symbol $\perp$ if $y_r \notin Y$.

$\mathsf{Ver}(K_{\mathsf{acc}}, y_r, w_r, z_{\mathsf{acc}}) \to \{0, 1\}$: This is a deterministic algorithm that, on input of $K_{\mathsf{acc}}, y_r, w_r$ and z_{acc}, outputs 0, meaning that w_r does not constitute a valid proof that y_r has been accumulated within z_{acc}, or 1, meaning that w_r constitutes a valid proof that y_r has been accumulated within z_{acc}.

We require for a collision-free accumulator scheme to satisfy the following conditions as correctness.

Definition 13 (Correctness of Collision-Free Accumulators) This notion captures the requirement that every value that was accumulated in a set can be authenticated, i.e., for any $\lambda, N'_{\max} \in \mathbb{N}$, we have

$$\begin{aligned}&\Pr[\mathsf{Ver}(K_{\mathsf{acc}}, y_r, w_r, z_{\mathsf{acc}}) = 1, (y_1, \ldots, y_{N'_{\max}}) \in \mathcal{Y}_{K_{\mathsf{acc}}}, y_r \in Y] \\ &\quad = 1,\end{aligned}$$

where $K_{\mathsf{acc}} \leftarrow \mathsf{AccKeyGen}(1^\lambda, 1^{N'_{\max}})$, $Y := \{y_1, \ldots, y_{N'_{\max}}\}$, $(z_{\mathsf{acc}}, \mathsf{aux}) \leftarrow \mathsf{Eval}(K_{\mathsf{acc}}, Y)$ and $w_r \leftarrow \mathsf{Wit}(K_{\mathsf{acc}}, z_{\mathsf{acc}}, y_r, \mathsf{aux})$.

Moreover, we recall the definition of the collision-freeness.

Definition 14 ($N'_{\max}$-times Collision-Freeness) We say that an accumulator scheme is $N'_{\max}$-times collision-free if, for any $\lambda, N'_{\max} \in \mathbb{N}$ and polynomial time adversary $\mathcal{A}_{pol}$, there exists a negligible function negl such that

$$\begin{aligned}&\Pr[\mathsf{Ver}(K_{\mathsf{acc}}, y', w', z_{\mathsf{acc}}) = 1, (y_1, \ldots, y_{N'_{\max}}, y') \in \mathcal{Y}_{K_{\mathsf{acc}}}, y' \notin Y] \\ &\quad < \mathsf{negl}(\lambda),\end{aligned}$$

where $K_{\mathsf{acc}} \leftarrow \mathsf{AccKeyGen}(1^\lambda, 1^{N'_{\max}})$, $(y_1, \ldots, y_{N'_{\max}}, y', w') \leftarrow$

$\mathcal{A}_{pol}(1^{\lambda}, 1^{N'_{\max}}, K_{\mathsf{acc}})$, $Y := \{y_1, \ldots, y_{N'_{\max}}\}$ and $(z_{\mathsf{acc}}, \mathsf{aux}) \leftarrow \mathsf{Eval}(K_{\mathsf{acc}}, Y)$.

Definition 15 (Collision-Freeness) We say that an accumulator scheme is collision-free if, for any $\lambda, N'_{\max} \in \mathbb{N}$, it is $N'_{\max}$-times collision-free.

A.8 Fail-Stop Multisignatures with Cryptographic Accumulators

In this section, we show a general construction of FSMS with the collision-free accumulators.

A general construction of FSMS with the collision-free accumulators can be constructed from the following algorithms. In this case, some conversion algorithms (which are deterministic) are necessary for accumulating public keys. However, in order to make understanding easy, we do not give such a conversion algorithm to the following construction. The details of the conversion algorithms are described in Ref. [1].

$\mathsf{Setup}(1^k, 1^{\sigma}, 1^{\omega_{\max}})$: As a prekey, it picks random index K for a bundling homomorphism h_{η} and sets $h := h_K, G := G_K$ and $H := H_K$. In parallel, it generates λ and $N'_{\max}$ and then sets the accumulator key for a collision-free accumulator scheme by invoking $\mathsf{AccKeyGen}(1^{\lambda}, 1^{N'_{\max}}) \rightarrow K_{\mathsf{acc}}$. It outputs $\mathsf{pp} := (K, K_{\mathsf{acc}}, h, G, H, \omega_{\max})$.

$\mathsf{KeyGen}(\mathsf{pp}, 1^{N_{\max}})$: Firstly, it checks if h is a group homomorphism with bundling degree 2^{τ}. If not, it executes Setup algorithm to obtain such a bundling homomorphism. Once it is convinced, it outputs a component of secret key as $sk_{j,N} = (sk_{j,N,1}, sk_{j,N,2}) \in_R G^2$ and its corresponding component of public key as $pk_{j,N} = (pk_{j,N,1}, pk_{j,N,2}) = (h(sk_{j,N,1}), h(sk_{j,N,2})) \in H^2$ for $N = 1 \ldots N_{\max}$. After that, it accumulates the set of $\{pk_{j,N}\}_{N=1}^{N_{\max}}$, i.e., it invokes $\mathsf{Eval}(K_{\mathsf{acc}}, Y_{j,1} := \{pk_{j,N,1}\}_{N=1}^{N_{\max}}) \rightarrow (z_{\mathsf{acc},j,1}, \mathsf{aux}_{j,1})$ and $\mathsf{Eval}(K_{\mathsf{acc}}, Y_{j,2} := \{pk_{j,N,2}\}_{N=1}^{N_{\max}}) \rightarrow (z_{\mathsf{acc},j,2}, \mathsf{aux}_{j,2})$. Finally, it outputs the secret key as $SK_j := (\{sk_{j,N}\}_{N=1}^{N_{\max}}, \mathsf{aux}_{j,1}, \mathsf{aux}_{j,2})$ and its corresponding public key as $PK_j := (z_{\mathsf{acc},j,1}, z_{\mathsf{acc},j,2})$. At the end of key generation, it sets a local-state counter $\ell \leftarrow 0$.

$\mathsf{MSign}(\mathsf{pp}, sk_j, m, \ell)$: The message space $\mathcal{M}$ is given as a subset of $\mathbb{Z}$. It first starts with incrementing its local-state counter by one, i.e., $\ell \leftarrow \ell + 1$. Moreover, it sets $N \leftarrow \ell$. It computes and broadcasts a partial signature $s_{j,N} \in G$ as $s_{j,N} = sk_{j,N,1} + m_N sk_{j,N,2}$ and receives partial signatures of other signers. Given $\{s_{j,\cdot}\}_{j=1}^{i}$, then it can obtain a multisignature $S_{\cdot} \in G$ as $S_{\cdot} = \sum_{j=1}^{i} s_{j,\cdot}$ with respect to the public keys on $L_{\cdot} = \{pk_{j,\cdot}\}_{j=1}^{i}$. If $i > \omega_{\max}$, then it outputs $\perp$. After that, it computes the witness of $pk_{j,N}$ by invoking $\mathsf{Wit}(K_{\mathsf{acc}}, z_{\mathsf{acc},j,1}, pk_{j,N,1}, \mathsf{aux}_{j,1}) \rightarrow w_{j,N,1}$ and $\mathsf{Wit}(K_{\mathsf{acc}}, z_{\mathsf{acc},j,2}, pk_{j,N,2}, \mathsf{aux}_{j,2}) \rightarrow w_{j,N,2}$. It sets $w_{j,N} := (w_{j,N,1}, w_{j,N,2})$ and $W_{\cdot} := \{w_{j,\cdot}\}_{j=1}^{i}$. Finally, it outputs $(m_{\cdot}, S_{\cdot}, L_{\cdot}, W_{\cdot})$.

$\mathsf{MVerify}(\mathsf{pp}, m, S, L)$: It outputs 1 if, for $j = 1 \ldots i$, $\mathsf{Ver}(K_{\mathsf{acc}}, pk_{j,\cdot,1}, w_{j,\cdot,1}, z_{\mathsf{acc},j,1}) \rightarrow 1 \wedge \mathsf{Ver}(K_{\mathsf{acc}}, pk_{j,\cdot,2}, w_{j,\cdot,2}, z_{\mathsf{acc},j,2}) \rightarrow 1 \wedge h(S_{\cdot}) = \prod_{j=1}^{i}(pk_{j,\cdot,1} \times pk_{j,\cdot,2}^{m_{\cdot}})$ holds. Otherwise, it outputs 0.

$\mathsf{ProveForgery}(\mathsf{pp}, sk_j, m^{\star}, S^{\star}, L^{\star})$: Parse $pk'_{j,\cdot} := (pk'_{j,\cdot,1}, pk'_{j,\cdot,2})$ which is used in the verification test for $(m^{\star}_{\cdot}, S^{\star}_{\cdot}, L^{\star}_{\cdot}, W^{\star}_{\cdot})$:

- When $pk'_{j,\cdot,1} \in Y_{j,1} \wedge pk'_{j,\cdot,2} \in Y_{j,2}$ holds, it computes a partial signature $s_{j,\cdot} \in G$ as $s_{j,\cdot} = sk_{j,\cdot,1} + m^{\star}_{\cdot} sk_{j,\cdot,2}$, and outputs a proof of forgery $pr_{\cdot} = \sum_{j=1}^{i} s_{j,\cdot}$ via partial signatures of other signers by invoking MSign. If either $h(S^{\star}_{\cdot}) \neq \prod_{j=1}^{i}(pk'_{j,\cdot,1} \times pk'^{m^{\star}_{\cdot}}_{j,\cdot,2})$ or $S^{\star}_{\cdot} = pr_{\cdot}$ holds, then it outputs $\perp$.
- When $(pk'_{j,\cdot,1}, pk'_{j,\cdot,2}) \in \mathcal{Y}^2_{K_{\mathsf{acc}}} \wedge pk'_{j,\cdot,1} \notin Y_{j,1} \wedge pk'_{j,\cdot,2} \notin Y_{j,2}$ holds, it outputs a proof of collision, i.e., $pr_{\cdot} := (Y_{j,1}, Y_{j,2}, pk'_{j,\cdot,1}, pk'_{j,\cdot,2}, PK_j, W^{\star}_{\cdot})$. This proof shows that the assumption on which the accumulator is based has been broken. If $h(S^{\star}_{\cdot}) \neq \prod_{j=1}^{i}(pk'_{j,\cdot,1} \times pk'^{m^{\star}_{\cdot}}_{j,\cdot,2})$ holds, then it outputs $\perp$.

$\mathsf{VerifyProof}(\mathsf{pp}, m^{\star}, S^{\star}, L^{\star}, pr)$: Parse $pr_{\cdot}$:

- When $pr_{\cdot} := \sum_{j=1}^{i} s_{j,\cdot}$, it outputs 1 if $S^{\star}_{\cdot} \neq pr_{\cdot}$ and $h(S^{\star}_{\cdot}) = h(pr_{\cdot})$ hold. Otherwise, it outputs 0.
- When $pr_{\cdot} := (Y_{j,1}, Y_{j,2}, pk'_{j,\cdot,1}, pk'_{j,\cdot,2}, PK_j, W^{\star}_{\cdot})$, it outputs 1 if $(pk'_{j,\cdot,1}, pk'_{j,\cdot,2}) \in \mathcal{Y}^2_{K_{\mathsf{acc}}} \wedge pk'_{j,\cdot,1} \notin Y_{j,1} \wedge pk'_{j,\cdot,2} \notin Y_{j,2} \wedge \mathsf{Ver}(K_{\mathsf{acc}}, pk'_{j,\cdot,1}, w'_{j,\cdot,1}, z_{\mathsf{acc},j,1}) \rightarrow 1 \wedge \mathsf{Ver}(K_{\mathsf{acc}}, pk'_{j,\cdot,2}, w'_{j,\cdot,2}, z_{\mathsf{acc},j,2}) \rightarrow 1$ holds. Otherwise, it outputs 0.

Nobuaki Kitajima received his B.Eng. degree from Department of Engineering, Chiba University, Japan, in 2012 and M.S.Eng. degree from Graduate School of Systems and Information and Engineering, University of Tsukuba, Japan, in 2014. He has recently worked in Tata Consultancy Services Japan, Limited, Japan.

Naoto Yanai received his B.Eng. degree from Ichinoseki National College of Technology, Japan, in 2009, M.S.Eng. degree from Graduate School of Systems and Information and Engineering, University of Tsukuba, Japan, in 2011, and Dr.E. degree from Graduate School of Systems and Information and Engineering, University of Tsukuba, Japan, in 2014. He is an assitant professor at Osaka University, Japan. His research is in the are of cryptography and information security.

Takashi Nishide received B.S. degree from The University of Tokyo in 1997, M.S. degree from the University of Southern California in 2003, and Dr.E. degree from the University of Electro-Communications in 2008. From 1997 to 2009, he had worked at Hitachi Software Engineering Co., Ltd., developing security products. From 2009 to 2013, he had been an assistant professor at Kyushu University and from 2013 he is an associate professor at University of Tsukuba. His research is in the areas of cryptography and information security.

Goichiro Hanaoka graduated from the Department of Engineering, The University of Tokyo in 1997. Received Ph.D. degree from The University of Tokyo in 2002. Joined AIST in 2005. Currently, Leader, Advanced Cryptosystems Research Group, Information Technology Research Institute, AIST. Engages in the R&Ds for encryption and information security technologies including the efficient design and security evaluation of public key cryptosystem. Received the Wilkes Award (2007), British Computer Society; Best Paper Award (2008), The Institute of Electronics, Information and Communication Engineers; Innovative Paper Award (2012, 2014), Symposium on Cryptography and Information Security (SCIS); Award of Telecommunication Advancement Foundation (2005); 20th Anniversary Award (2005), SCIS; Best Paper Award (2006), SCIS; Encouragement Award (2000), Symposium on Information Theory and its Applications (SITA); and others.

Eiji Okamoto received his B.S., M.S. and Ph.D. degrees in electronics engineering from the Tokyo Institute of Technology in 1973, 1975 and 1978, respectively. He worked and studied communication theory and cryptography for NEC central research laboratories since 1978. In 1991 he became a professor at Japan Advanced Institute of Science and Technology, then at Toho University. Now he is a professor at Faculty of Engineering, Information and Systems, University of Tsukuba. His research interests are cryptography and information security. He is a coeditor-in-chief of Internatinal Journal of Information Security, and a member of IEEE and ACM.