Regular Paper

A New Schnorr Multi-Signatures to Support Both Multiple Messages Signing and Key Aggregation

Rikuhiro Kojima[1] Dai Yamamoto[1] Takeshi Shimoyama[1] Kouichi Yasaki[1] Kazuaki Nimura[1]

Received: December 1, 2020, Accepted: June 7, 2021

Abstract: A digital signature is essential for verifying people's reliability and data integrity over networks and is used in web server certificates, authentication, and blockchain technologies. Specifically, to solve the bitcoin scalability problem, Multi-Signature (MS) schemes have recently attracted attention because the MS's aggregate algorithm can reduce the amount of signature data in transactions. While such schemes support only a single message signing, Interactive Aggregate Signatures (IAS) and Aggregate Multi-Signature Protocol (AMSP) support signing of multiple messages. However, there are some issues with these schemes, for example, key aggregation is unavailable. In this paper, we propose a key aggregatable IAS scheme called KAIAS that can sign multiple messages with key aggregation. In terms of cases using Multi-Signature, previous studies have mainly discussed the benefits of reducing the size of signatures. On the other hand, we also propose a practical application of KAIAS that leverages its benefits in aggregating both signatures and public keys with a low computing cost for verification.

Keywords: multi-signature, interactive aggregate signature, Schnorr signature, trust service, eIDAS

1. Introduction

Electronic signatures are used as data "fingerprints", mainly in identity verification and preventing spoofing from occurring. In recent signature applications, its attention has focused not only on using it as a conventional sign but also on using it as an authentication method.

For example, in authentication protocols such as SSH (Secure SHell) [27] and FIDO (Fast Identity Online Universal Authentication Framework) [16], the client sends a valid signature to the server instead of a password hash. This modification is useful and effective for security because the client does not have to remember the password. Another benefit is that the server does not have to keep the client's secret information in their secure storage.

Multi-Signatures

Multi-signature schemes have been attracting attention for their convenience and functionality.

A multi-signature scheme [22] is a scheme in which n parties (who have their own private key sk_i and public key pk_i where $i \in \{1, \cdots, n\}$) sign their signature σ_i for each $i \in \{1, \cdots, n\}$ in a common message m and output an aggregated signature σ, which is a static size independent of n. The verifier is given m that is attached with σ with public key list $L_{pk} = \{pk_1, \cdots, pk_n\}$ and confirms the correctness by running the verification algorithm, which outputs an accept or reject. Instead of a public key list L_{pk}, Maxwell et al. proposed verifying aggregate signatures using a single public key apk generated from L_{pk}, called an aggregate public key [18]. This method is called Key Aggregation.

Multi-signature schemes are constructed by expanding suitable provable signature schemes, e.g., a Schnorr signature scheme [6] or BLS (Boneh-Lynn-Shacham) signature scheme [7]. Also, RSA-based [22], pairing-based [4], and lattice-based [10] have been proposed.

Even if we construct a secure model under any assumption, naive multi-signature schemes are vulnerable to the "Rogue-key Attack" [4], [18], [25], which is caused by registering the corrupted public key. To avoid this attack, there are two model assumptions. In the first model, during the public key registration step, all users register their public key for trusted Certificate Authority (CA), and they prove their "knowledge" (or possession [25]) of the secret key to CA, in actual work situations, this knowledge is used as the certificate. This model is usually formalized as the knowledge of the secret key (KOSK).

In the second model, all users can generate their key pair locally without registering for CA. The model proposed by Bellare and Neven [6] is formalized as the plain public-key model. Bellare and Neven showed a provable multi-signature scheme in the plain public-key model under the Discrete Logarithm assumption. This scheme however does not support key aggregation. After that Maxwell et al. proposed improvements to include key aggregation, which has a verification algorithm does not require L_{pk} but only uses apk [18].

In the KOSK model, although dynamic key aggregation (aggregate new public key) is available, it still requires a strong assumption that a CA is a trusted third party. On the other hand, plain public-key models are limited to using static key aggrega-

[1] FUJITSU LABORATORIES LTD., Kawasaki, Kanagawa 211–8588, Japan

This paper is an updated version of Ref. [14] and we have added new sections like Section 3.2 (Security Proof), Section 4.2 (New Application), and Section 5 (Conclusion and Future works).

tion (cannot aggregate new public key) but do not require CA assumptions.

Multiple Messages Signing

Aggregate Signature [8] is a scheme in which n parties sign their own messages and output an aggregated signature σ (fixed size independent of n) in situations where each party has a single message. In general, by replacing $\{m_1, \cdots, m_n\}$ with m, we can translate aggregate signatures into multi-signatures. Thus, Aggregate Signature is defined as a generalization of a multi-signature scheme. This scheme does however have some restrictions. These include for example that the verifier much check which messages are signed by the signers before verification and that all message must differ from each other.

Also, a sequential aggregate signature has been proposed [5], [15], [31]. In this scheme, i-th signer with an index from $2, \cdots, n$ takes as input an aggregate signature σ_{i-1} from the $i-1$-th signer, aggregate with its own signature and outputs σ_i and sends it to the $i+1$-th signer until all messages are signed. This scheme requires n-round communication for n signers, even though the verifier can see the signer's aggregation order.

Prior works of aggregate signatures are based on the KOSK model s.t. requiring a CA's presence and do not support verification using an aggregated public key generated by Key Aggregation. To support plain public key models and key aggregation, we plan to construct algorithms based on MuSig instead of some Aggregate Signatures.

As described above, multi-signature schemes only support single message signing. The signers however may want to add or modify any message for signing dynamically and aggregate it in some use cases like aggregate signatures. Maxwell et al. proposed transforming multi-signature schemes to secure Interactive Aggregate Signature (IAS) [18], including a signing algorithm for the single combined message generated from multi messages. This scheme appears to work well for signing multiple messages thought it does not satisfy (1) verification in constant time because IAS does not support key aggregation. Boneh et al. proposed the Aggregate Multi-Signatures Protocol (AMSP) [4], including key aggregation algorithm like MuSig, but this scheme does not satisfy (2) signing in constant time because all signers have to sign all signer's messages.

We will discuss the issues in more detail in Section 2.3.

Our Proposal

To solve the above issues, we propose KAIAS which is a Key Aggregatable IAS scheme that does not require additional public-key model assumptions and is constructed from three-round protocols like MuSig. This scheme supports multiple message signing, in addition to (1) verification in constant time using an aggregated key, and (2) signing in constant time.

Compared with IAS and AMSP, KAIAS has a slightly larger aggregated signature size. This size is however negligible because it is a static value independent of the number of signers n, so by increasing n, we can benefit from signature aggregation and key aggregation the same as with other multi-signature schemes. **Table 1** shows a comparison of KAIAS, IAS, and AMSP.

Table 1 Comparison of the Multi-Signature schemes supporting multiple messages when using a group $\mathbb{G}$ and hash functions H where d, ℓ are the bit size of each of the elements $\mathbb{G}$ and H.

Schemes	sig. size	key agg.	sig. time
IAS [18]	$d+\ell$	no	1
AMSP [4]	$d+\ell$	yes	n
KAIAS (Proposal)	$2d+\ell$	yes	1

sig. size : Size of signature
key agg. : Key aggregation is available.
sig. time : Number of signing times for each co-signer.

Applications

Wuille et al. recently proposed a draft of Bitcoin Improvement Proposal (BIP) 340 (Schnorr signatures for secp256k1) [30], which is the standard implementation for improving the Bitcoin protocols [19] using the Schnorr signature scheme. BIP 340 uses the ECSDSA (Elliptic Curve Schnorr Digital Signature Algorithm) and has the following three advantages, Provable Security, Non-malleability and Linearity, over the ECDSA (Elliptic Curve Digital Signature Algorithm), which is the standard signature protocol in the current Bitcoin. By using the linearity of the Schnorr signature scheme, we can construct advanced Schnorr-based signature schemes such as Adaptor Signatures and Blind Signatures. In BIP 340, Schnorr signatures are not limited to solving the scalability problem in Bitcoin but are expected to improve computing efficiency and privacy protection by using these applications. This is why Bitcoin has recently been the most popular application of multi-signature schemes.

We can also consider applying KAIAS to Bitcoin applications like MuSig. On the other hand, recently there has been an increased demand for the application of Trust Services following eIDAS (Electronic Identification, Authentication and trust Services) regulation [11] of the EU. Trust Services are components for determining how to trust other entities such as public services or businesses over a network. Trust Services are composed of electronic signature, timestamps and e-Seal, etc. [28]. In Japan, in particular, Japan Digital Trust Forum (JDTF) has proposed Trust as a Service (TaaS) to facilitate the use of trust services by mediating between the cloud and clients [12]. Fujitsu Laboratories has also proposed specific schemes of TaaS [17].

In this paper, we propose using a contract between companies with TaaS and the use case of Code Signing with TaaS to apply KAIAS in the trust service (e-Seal, Code Signing).

Overview

We first describe the notations used in this paper including the discrete logarithm problem, multi-signature schemes, IAS, and AMSP in Section 2. We then present KAIAS and discuss its security proof in Section 3 and propose two practical applications with KAIAS in Section 4. Finally, we conclude and discuss some issues in Section 5.

2. Preliminaries

2.1 Definitions

2.1.1 Notation

Given a non-empty set S, we denote by $s \leftarrow_{\$} S$ which means the operation of sampling an element s from S uniformly at random. If y is an output of a randomized algorithm $\mathcal{A}$, we denote by $y \leftarrow \mathcal{A}(x_1, \cdots; \rho)$, where $x_1, \cdots$ are inputs and ρ are random coins, and $y \leftarrow_\$ \mathcal{A}(x_1, \cdots)$ when coins ρ are chosen uniformly at random.

The word "semi-honest" means that attackers comply with the protocol, but exploit the information gained in the meantime.

In all the following, Let $\mathbb{G} = \langle g \rangle$ be a cyclic group of prime order p, where p is a κ-bit integer, with generator g, and we call $(\mathbb{G}, p, g)$ the *group parameters*.

2.1.2 Discrete Logarithm Problem

The provable security of Schnorr Signature [26] is based on the discrete logarithm problem (DLP).

Def. 1 (Discrete Logarithm Problem (DLP))**.**

Let $(\mathbb{G}, p, g)$ be the *group parameter*. We define an $\mathsf{Adv}^{\mathsf{DL}}_{\mathbb{G}}$ as

$$Pr[y = g^x : y \leftarrow_\$ \mathbb{G}, x \leftarrow_\$ \mathcal{A}(y)]$$

which is a probability is taken over the random choice of $\mathcal{A}$ and random elements y. An algorithm $\mathcal{A}$ is said to (τ, ϵ)-solve DLP if it runs in time at most τ and $\mathsf{Adv}^{\mathsf{DL}}_{\mathbb{G}} > \epsilon$.

2.1.3 Generalized Forking Lemma

The forking lemma [24] is used to prove the security proof of Schnorr signature schemes in the Random Oracle model. As in Ref. [6], our security proofs rely on the following generalized forking lemma [3].

Lemma. 1 (Generalized Forking Lemma)**.**

Let q and ℓ be a integer, $\mathcal{A}$ be a randomized algorithm which takes **inp** as main inputs, ℓ-bit strings $h_1, \cdots, h_q \in \{0,1\}^\ell$ as random numbers, and ρ as $\mathcal{A}$'s random coins, and returns either a $\perp$ (*Failure*) or $(i, \mathbf{out})$ (*Success*), where $i \in \{1, \cdots, q\}$ and **out** is main outputs. We describe $acc(\mathcal{A})$ as the probability of accepting $\mathcal{A}$, which means $\mathcal{A}$ returns non-$\perp$ output. Following the above definition of $acc(\mathcal{A})$, we can consider algorithm $\mathsf{Fork}^{\mathcal{A}}$ which takes **inp** as the same input and returns either a $\perp$ or $(i, \mathbf{out}, \mathbf{out}')$ as outputs described on Listing 1 and let frk be the probability of accepting $\mathsf{Fork}^{\mathcal{A}}$, then we obtain

$$frk \geq acc(\mathcal{A})\left(\frac{acc(\mathcal{A})}{q} - \frac{1}{2^\ell}\right).$$

Listing 1: Forking Algorithm

```
$(h_1, \cdots, h_q) \leftarrow_\$ \{0,1\}^\ell$
$\alpha \leftarrow \mathcal{A}(\mathbf{inp}, h_1, \cdots, h_q; \rho)$
if $\alpha = \perp$ then return $\perp$
else parse $\alpha = (i, \mathbf{out})$
$h'_i, \cdots, h'_q \leftarrow_\$ \{0,1\}^\ell$
$\alpha' \leftarrow \mathcal{A}(\mathbf{inp}, h_1, \cdots, h_{i-1}, h'_i, \cdots, h'_q; \rho)$
if $\alpha' = \perp$ then return $\perp$
else parse $\alpha' = (i', \mathrm{out}')$
if $(i = i' \wedge h_i \neq h'_i)$ then return $(i, \mathbf{out}, \mathbf{out}')$
else return $\perp$
```

2.1.4 Multi-Signature Scheme

We follows the definition of Refs. [4] and [6], and we describe the scheme and security of multi-signatures. In general, a Multi-Signature scheme MS is constructed by algorithms Pg, Kg, Sign, SAg, KAg, and Vf is defined by the following.

$\mathsf{Pg}(1^\kappa)$
: Given the security parameter κ, output the system parameters params.

$\mathsf{Kg}(\mathsf{params})$
: Given the params, output a key pair (sk, pk) where sk is a secret key and pk is a public key.

$\mathsf{Sign}(\mathsf{params}, L_{pk}, sk, msg)$
: Given the $(\mathsf{params}, sk, L_{sk}, msg)$, where L_{pk} is the set of the public key, output a signature σ.

$\mathsf{SAg}(\mathsf{params}, L_{sig})$
: Given the $(\mathsf{params}, L_{sig})$, where L_{sig} is the set of the signature, output an aggregated signature $\widetilde{\sigma}$.

$\mathsf{KAg}(\mathsf{params}, L_{pk})$
: Given the $(\mathsf{params}, L_{pk})$, output an aggregated public key $\overline{pk}$.

$\mathsf{Vf}(\mathsf{params}, \overline{pk}, msg, \widetilde{\sigma})$
: Given the $(\mathsf{params}, \overline{pk}, msg, \widetilde{\sigma})$ and verify them. If $\widetilde{\sigma}$ is the valid signature for m, then output 1, otherwise output 0.

This scheme should satisfy completeness which means the following equation is valid for any n.

$$\mathsf{Vf}(\mathsf{params}, \mathsf{KAg}(\mathsf{params}, L_{pk}), msg, \mathsf{SAg}(\mathsf{params}, L_{sig})) = 1$$

where $L_{sig} = \{\sigma_1, \cdots, \sigma_n\}$, $\sigma_i = \mathsf{Sign}(\mathsf{params}, L_{pk}, sk_i, msg)$, $L_{pk} = \{pk_1, \cdots, pk_n\}$ and $(sk_i, pk_i) \leftarrow Kg(\mathsf{params})$ for each $i \in \{1, \cdots, n\}$.

In addition, this scheme should satisfy unforgeability. Unforgeability of MS is defined by the following three-stage game.

Setup. The challenger generates $\mathsf{params} \leftarrow \mathsf{Pg}(1^\kappa)$, and let $(sk^*, pk^*) \leftarrow \mathsf{Kg}(\mathsf{params})$ be a challenge key pair. It runs the adversary (the forger algorithm) $\mathcal{A}(\mathsf{params}, pk^*)$.

Signature queries. $\mathcal{A}$ has access to sign oracle $O^{\mathsf{Sign}(\mathsf{params}, \cdot, sk^*, \cdot)}$ for any message m and any set of signer public keys $L_{pk} = \{pk_1, \cdots, pk_n\}$ where $pk_i = pk^*$ for any i. This oracle will simulate the honest signer and output the forgery signature σ_i.

Output. The adversary outputs a forged Multi-Signature $\overline{\sigma}^*$, a message m^* which is not queried in previous stage, and a set L_{pk} including pk^*. The adversary wins this game if the following equation is satisfied.

$$\mathsf{Vf}(\mathsf{params}, \mathsf{KAg}(\mathsf{params}, L_{pk}), m^*, \overline{\sigma}) = 1$$

Finally, we recall the definition of the unforgeability of Multi-Signature.

Def. 2 (Unforgeability of Multi-Signature)**.**

We said $\mathcal{A}$ is a $(\tau, q_S, q_H, \epsilon)$-forger for the Multi-Signature Scheme $\mathsf{MS} = (\mathsf{Pg}, \mathsf{Kg}, \mathsf{Sign}, \mathsf{SAg}, \mathsf{KAg}, \mathsf{Vf})$, where q_S is its maximal number of sign oracle queries and q_H is its maximal number of random-oracle queries, if it runs in time τ and wins the above game with the probability of at least ϵ. MS is a $(\tau, q_S, q_H, \epsilon)$-unforgeable if no $(\tau, q_S, q_H, \epsilon)$-forger exists.

2.2 Prior Works

2.2.1 MuSig

We remark on the definition of Multi-Signature scheme $\mathsf{MuSig} = (\mathsf{Pg}, \mathsf{Kg}, \mathsf{Sign}, \mathsf{SAg}, \mathsf{KAg}, \mathsf{Vf})$, which is denoted by Maxwell et al. [18]. This scheme is executed by n signers, a verifier, and a "aggregator".

The aggregators are separated into a signature aggregator that performs SAg and a key aggregator that performs KAg. Since the aggregation of signatures assumes a malicious signer's presence, it is not necessary to assume that the signature aggregator is also trusted (Any signer may also act as the signature aggregator).

On the other hand, in the case of key aggregation, we assumed that the key aggregator is semi-honest as in the previous works because it is outside the scope of the verifier and must depend on the assurance of the key aggregator. If there are multiple key aggregators on the signer side and verifier side, and at least one key aggregator is semi-honest, then the aggregated public key which KAg generates is valid.

Parameter Generation. $\mathsf{MuSig.Pg}\ (1^\kappa)$

The aggregator generates the *group parameters* $\mathsf{params} \leftarrow \mathsf{Pg}\ (1^\kappa)$, where $\mathsf{params} = (\mathbb{G}, p, g)$, and prepares three different hash function H_0, H_1, and H_2 defined as follows.

$$H_0, H_1, H_2 : \{0,1\}^* \to \mathbb{Z}_p - \{0\}$$

Key Generation. $\mathsf{MuSig.Kg}\ (\mathsf{params})$

Each i-th signer generates a random parameter $x_i \leftarrow_\$ \mathbb{Z}_p$ and compute the public key $X_i = g^{x_i}$. Lets $L_{pk} = \{X_1, \cdots, X_n\}$ be the multi-set of public keys.

Key Aggregation. $\mathsf{MuSig.KAg}\ (L_{pk})$

Each i-th signer computes $a_i = H_1(X_i, L_{pk})$ and sends a_i to the aggregator. After that, the aggregator computes an aggregated key $\widetilde{X} = \prod_{i=1}^n X_i^{a_i}$ and outputs $\widetilde{X}$.

Signing. $\mathsf{MuSig.Sign}\ (\mathsf{params}, L_{pk}, x, m)$

This algorithm outputs an aggregated signature $\overline{\sigma}$ following three rounds of interactive protocol.

(1) Each i-th signer generates a random integer $r_i \leftarrow \mathbb{Z}_p$ and computes $R_i \leftarrow g^{r_i}$ and $t_i = H_2(R_i)$ and sends t_i as a "commitment" to the aggregator, after that, the aggregator broadcasts $\{t_j\}_{j\in\{1,\cdots,n\}}$ to each signer.

(2) Each i-th signer sends R_i to the aggregator, after that, the aggregator broadcasts $\{R_j\}_{j\in\{1,\cdots,n\}}$ to each signer, and all signers check that $t_j = H_2(R_j)$ for each $j \neq i$ and aborts this protocol if the equation is incorrect.

(3) The aggregator computes $\widetilde{X} \leftarrow \mathsf{KAg}\ (L_{pk})$ and lets $a_i = H_1(X_i, L_{pk})$ and broadcasts $(\widetilde{X}, \{a_j\}_{j\in\{1,\cdots,n\}})$ to each signer, and all signers check that $\widetilde{X} = \prod_{i=1}^n X_i^{a_i}$.

After 3-round protocols, the aggregator computes $\overline{R} = \prod_{i=1}^n R_i$, $c = H_0(\overline{R}, \widetilde{X}, m)$ and broadcast c for all signers, and each i-th signer computes $s_i = r_i + c \cdot a_i \cdot x \bmod p$, where $a_i = H_1(X_i, L_{pk})$, and outputs $\sigma_i = (s_i, R_i)$.

Signature Aggregation. $\mathsf{MuSig.SAg}\ (\mathsf{params}, \{\sigma_1, \cdots, \sigma_n\})$

The aggregator parses $\sigma_i = (s_i, R_i)$ and computes $\overline{s} \leftarrow \sum_{i=1}^n s_i$ and outputs an aggregated signature $\overline{\sigma} = (\overline{s}, \overline{R})$.

Verification. $\mathsf{MuSig.Vf}\ (\mathsf{params}, m, \overline{\sigma}, \widetilde{X})$

This algorithm outputs $b \in \{0,1\}$ which means the verifier accepts or rejects it. Given a message m, an aggregated signature $\overline{\sigma}$ and an aggregated public key $\widetilde{X}$, the verifier computes $c = H_0(\overline{R}, \widetilde{X}, m)$, and outputs 1 if and only if $g^{\overline{s}} = \overline{R}\widetilde{X}^c$, otherwise it outputs 0.

We can confirm the following equations to check their correctness.

$$g^{\overline{s}} = g^{s_1+\cdots+s_n} = R_1 X_1^{a_1 c} \cdots R_n X_n^{a_n c} = \overline{R}\widetilde{X}^c$$

Intuitionally, this 3-round protocol in the signing process appears to work well without a first-round (1), and the first version of Ref. [18] in fact omitted this round. However, Drijvers et al. [9] found the sub-exponential attack for such a 2-round protocol using Wagner's algorithm for generalized birthday problem [29]. Therefore, Maxwell et al. revised [18] to include a a 3-round protocol like the above described protocol.

As an improvement scheme of MuSig, Yannick Seurin et al. proposed MuSig-DN [21], and J. Nick et al. proposed MuSig2 [20]. MuSig-DN allows the use of a deterministic nonce generated from a private key instead of a nonce generated from a pseudo-random generator, and the other signers can verify the authenticity since this deterministic nonce was used for signing using zero-knowledge proofs. This improvement allows MuSig-DN to protect against key exfiltration attacks caused by the bad pseudo-random generator. MuSig2, on the other hand, has been modified to reduce the communication between users from 3-round to 2-round.

Their modification scope is limited to the nonce and key generation protocol, in contrast, our proposal's scope is the generation algorithm of the aggregate signature. Therefore, these are independent of each other, and we believe that we can easily combine our proposals with MuSig-DN and MuSig2.

2.2.2 Rogue-key Attack

If we define $\widetilde{X} = \prod_{i=1}^n X_i$ as an aggregated public key instead of $\mathsf{MuSig.KAg}$, the adversary (any corrupted signers) can execute a Rogue-key Attack via the following steps.

(1) The adversary has a new key pair $(x_{n+1}, X_{n+1}) \leftarrow \mathsf{Kg}\ (\mathsf{params})$ and computes a rogue-key $X' = X_{n+1} \cdot \widetilde{X}^{-1}$ and registers it.

(2) The aggregator computes a new aggregated public key as $\widetilde{X'} = \mathsf{KAg}\ (\{L_{pk}, X'\}) = X_{n+1}$, thus, the adversary can generate signatures which are verifiable with $\widetilde{X'}$ signed by sk_{n+1}.

Bellare and Neven's proposal includes an individual public key X_i and the public key list L_{pk} among the inputs of the hash value for avoiding the rogue-key attack [6]. In contrast, Maxwell et al. has proposed separating the hash value of the public keys $a_i = H_1(X_i, L_{pk})$ from the message hash value to allow public key aggregation [18].

Namely, in MuSig, a parameter a_i in a part of the aggregate public key $\widetilde{X} = \prod_{i=1}^n X_i^{a_i}$ exists as a countermeasure against a rogue-key attack.

2.2.3 Interactive Aggregate Signature (IAS)

Compared with the multi-signature schemes, which include only a signing algorithm for a common message, IAS supports a distinct messages signing algorithm. Bellare and Neven [6] suggested a generic approach to transform any multi-signature scheme into an IAS, and Maxwell et al. [18] then proposed a fixed secure IAS schemes.

In the IAS scheme, instead of a public key list $L_{pk} = \{X_1, \cdots, X_n\}$ and a single message m, the signer and the verifier use an ordered set of the tuple of public key and message pairs $S = \{(X_1, m_1), \cdots, (X_n, m_n)\}$ instead of the message.

We remark on the definition of IAS $\mathsf{IAS} = (\mathsf{Pg}, \mathsf{Kg}, \mathsf{Sign}, \mathsf{SAg}, \mathsf{KAg}, \mathsf{Vf})$ where $\mathsf{Pg} = \mathsf{MuSig.Pg}$, $\mathsf{Kg} = \mathsf{MuSig.Kg}$, $\mathsf{SAg} = \mathsf{MuSig.SAg}$, and $\mathsf{KAg} = \mathsf{MuSig.KAg}$ informally, which is denoted by Maxwell et al. in appendix A of Ref. [18].

Signing. $\mathsf{IAS.Sign}$ $(\mathsf{params}, S, \{sk_1, \cdots, sk_n\})$

After 3-round protocols in $\mathsf{MuSig.Sign}$, the aggregator computes

$$\widetilde{X} \leftarrow \mathsf{IAS.KAg}(L_{pk})$$
$$\overline{R} = \prod_{i=1}^{n} R_i$$
$$c_i = H_0(S, \widetilde{X}, i)$$

and each i-th signer computes

$$s_i = r_i + c_i x_i \bmod p$$

and output $\sigma_i = (s_i, R_i)$.

Verification. $\mathsf{IAS.Vf}$ $(\mathsf{params}, S, \overline{\sigma}, \widetilde{X})$

Given a set S, an aggregated signature

$$\overline{\sigma} \leftarrow \mathsf{IAS.SAg}\ (\mathsf{params}, \{\sigma_1, \cdots, \sigma_n\})$$

and an aggregated public key $\widetilde{X}$, the verifier computes $c_i = H_0(S, \widetilde{X}, i)$, and outputs 1 if and only if $g^{\overline{s}} = \overline{R} \prod_{i=1}^{n} X_i^{c_i}$ and otherwise outputs 0.

2.2.4 Aggregate Multi-Signature Protocol (AMSP)

To aggregate some signatures generated from distinct messages without using IAS, Boneh et al. introduced AMSP, which is a scheme that outputs an aggregated signature aggregated from some multi-signatures [4]. Besides pairing, this AMSP aggregation technique can also be applied to all Multi-Signature schemes whose signature aggregation has homomorphism.

We remark on the definition of $\mathsf{AMSP} = (\mathsf{Pg}, \mathsf{Kg}, \mathsf{Sign}, \mathsf{SAg}, \mathsf{KAg}, \mathsf{Vf}, \mathsf{AVf})$ where $\mathsf{Pg} = \mathsf{MuSig.Pg}$, $\mathsf{Kg} = \mathsf{MuSig.Kg}$, $\mathsf{Sign} = \mathsf{MuSig.Sign}$, $\mathsf{KAg} = \mathsf{MuSig.KAg}$, and $\mathsf{Vf} = \mathsf{MuSig.Vf}$ informally, which is denoted by Boneh et al. in Ref. [4].

Signature Aggregation. $\mathsf{AMSP.SAg}$ $(\overline{\sigma_1}, \cdots, \overline{\sigma_n})$

Parse $\overline{\sigma_i} = (\overline{s_i}, \overline{R_i})$ where

$$\overline{\sigma_i} \leftarrow \mathsf{MuSig.SAg}\ (\mathsf{params}, \{\sigma_{i,1}, \cdots, \sigma_{i,n}\})$$

for each $i \in \{1, \cdots, n\}$ and

$$\sigma_{i,j} \leftarrow \mathsf{MuSig.Sign}\ (\mathsf{params}, L_{pk}, x_j, m_i)$$

for each $j \in \{1, \cdots, n\}$, and outputs $\Sigma = (\overline{s}, \overline{R})$, where

$$(\overline{s}, \overline{R}) \leftarrow \mathsf{MuSig.SAg}\ (\mathsf{params}, \{\overline{\sigma_1}, \cdots, \overline{\sigma_n}\})$$

Aggregate Signature Verification.

$\mathsf{AMSP.AVf}$ $(\{m_1, \cdots, m_n\}, \Sigma)$

Output 1 if and only if $g^{\overline{s}} = \overline{R}\widetilde{X}^{\overline{c}}$, where $\overline{c} = \sum_{i=1}^{n} c_i$ and $c_i = H_0(\overline{R}, \widetilde{X}, m_i)$ and otherwise outputs 0.

2.2.5 Optimistic Aggregate Signature (OAS)

As another approach using aggregate signature with multi-signature, a signature method called Optimistic Aggregate Signature (OAS) has been proposed by Ambrosin et al. [1].

OAS is a generalization of multi-signatures and aggregate signature based on pairing that expects most signers to sign the default message M. In OAS, the aggregator aggregates the same messages using multi-signatures and aggregates the different messages using aggregate signature's aggregation. This feature shows that both the aggregated signature size and the verification time are linear in this kind of message but independent of the number of signers who have signed M. Namely, OAS performs best when everyone signs the same message (Essentially identical to multi-signatures); however, when they sign different messages, they have the same construction as a typical aggregate signature.

Although OAS uses an aggregated public key consisting of a simple aggregation (Multiplication) of individual public keys for the verification like MuSig, since independent public keys embedded in signatures are essentially used for the verification, the verification time is linear in these kind of messages. In addition, to prevent rogue-key attacks, OAS must follow the KOSK model and not the plain public-key model like MuSig.

In summary, OAS can be regarded as an extended Multi-Signature that supports distinct message signing. However, because OAS is constructed by pairing-based technique (using bilinear maps) rather than Schnorr-based technique, it has different security models, and the signature size is linear to the number of signers, OAS is based on different assumptions and models from MuSig, so we reference OAS just in this section.

2.3 Issues

This section summarizes what has been achieved and what has not been achieved by conventional multi-signature application which support multi-message signing.

Signature Verification in Constant Time

In MuSig, using an aggregated public key generated from some individual keys with Key Aggregation, the verification cost is reduced in constant time (Although verification time for an aggregated key is linear in this kind of key, the verifier can delegate this computation to a trusted third party like the aggregator).

On the other hand, signature verification of IAS requires all individual public keys $pk_1, \cdots, pk_n$ included in a public key list L_{pk}, and verification with an aggregated public key is not supported. Namely, IAS has an issue that the cost of the verification time and the number of public keys required for the verification are linear in the kind of message. This problem requires that the verifier requires all individual public keys for the verification and therefore increases the signer's key management costs and privacy risk such as requiring certificates for each public key issued by the private CA.

Signing in Constant Time

As described above, AMSP supports verification for an aggregated signature using MuSig's aggregated key. However, all signers requires signing n times per signer's message. This cost issue is a minor problem because of the relatively low signing costs such as for the transaction processing proposed in Ref. [4]. However, in cases in which signing a message requires the signer's operations (consensus), such as a digital signature to a PDF file, increasing the signing time is not small problem for each signer.

In response to the issue above, we propose a new signature scheme that enables both signing and verification in constant time in the next section.

3. Our Proposal

3.1 Construction of KAIAS

To consider applying Multi-Signature for more practical use cases, we propose a new IAS scheme based on 3-round MuSig that resolve the above issues.

In prior IAS, the verifier needed the public key list L_{pk} instead of an aggregated public key $\widetilde{X}$ to confirm the equation $g^{\widetilde{s}} = \overline{R}\prod_{i=1}^{n} X_i^{c_i}$. As the improvement of it, please note that we introduced the message hash aggregation $\overline{c} \leftarrow \mathsf{KAIAS.MHAg}(\{m_1, \cdots, m_n\}, \overline{R}, \widetilde{X})$ as well as the aggregation of signatures and public keys, and we embedded the additional parameter $\{d_i\}_{i\in\{1,\cdots,n\}}$ in an aggregated signature $\widetilde{\sigma}$. As a consequence of this modification, the verifier can verify using only $\widetilde{X}$ to confirm the equation $g^{\widetilde{s}} = \widetilde{R}\widetilde{X}^{\overline{c}}$.

Now we introduce the new Key Aggregatable IAS $\mathsf{KAIAS} = (\mathsf{Pg}, \mathsf{Kg}, \mathsf{KAg}, \mathsf{Sign}, \mathsf{SAg}, \mathsf{KVf}, \mathsf{AVf})$.

Hash Function. H_0

Define H_0, H_1, H_2 as $H_0, H_1, H_2 : \{0,1\}^* \to \mathbb{Z}_p - \{0\}$, which returns a non-zero value.

Parameter Generation. $\mathsf{KAIAS.Pg}\ (1^\kappa)$

Same as $\mathsf{MuSig.Pg}(1^\kappa)$.

Key generation. $\mathsf{KAIAS.Kg}\ (\mathsf{params})$

Same as $\mathsf{MuSig.Kg}(\mathsf{params})$.

Key Aggregation. $\mathsf{KAIAS.KAg}\ (L_{pk})$

Same as $\mathsf{MuSig.KAg}(L_{pk})$.

Signing. $\mathsf{KAIAS.Sign}\ (\mathsf{params}, L_{pk}, sk_i, m_i)$

This algorithm outputs an aggregated signature $\widetilde{\sigma}$. After 3-round protocols in $\mathsf{MuSig.Sign}$, each i-th signer computes

$$\overline{R} = \prod_{i=1}^{n} R_i$$
$$c_i = H_0(\overline{R}, \widetilde{X}, m_i)$$
$$a_i = H_1(X_i, L_{pk})$$
$$s_i = r_i + c_i a_i x_i \bmod p$$

and outputs $(c_i, \sigma_i = (s_i, R_i))$.

Signature Aggregation.

$\mathsf{MuSig.SAg}\ (\mathsf{params}, \{(c_1, \sigma_1), \cdots, (c_n, \sigma_n)\})$

The aggregator checks that $c_i = H_0(\overline{R}, \widetilde{X}, m_i)$. If this equation is incorrect, the aggregator aborts this protocol and otherwise computes

$$\widetilde{s} = \sum_{i=1}^{n} s_i d_i,\ \widetilde{R} = \prod_{i=1}^{n} R_i^{d_i}$$

where $\{d_i\}_{i\in\{1,\cdots,n\}}$ is the "wormhole" product of $\{c_i\}_{i\in\{1,\cdots,n\}}$ defined as

$$d_i = \prod_{j\in\{1,\cdots,n\}-\{i\}} c_j = c_1 \cdots c_{i-1} c_{i+1} \cdots c_n$$

for each i, and outputs an aggregated signature $\widetilde{\sigma} = (\widetilde{s}, \widetilde{R}, \overline{R})$.

Message Hash Aggregation.

$\mathsf{KAIAS.MHAg}\ (\{m_1, \cdots, m_n\}, \overline{R}, \widetilde{X})$

This algorithm outputs an aggregated message hash value $\overline{c}$. Computes $\overline{c} = \prod_{i=1}^{n} c_i$, where $c_i = H_0(\overline{R}, \widetilde{X}, m_i)$ for each $i \in \{1, \cdots, n\}$, and outputs $\overline{c}$.

Key Verification. $\mathsf{KAIAS.KVf}\ (L_{pk}, \widetilde{X})$

Given a public key list L_{pk} and an aggregated public key $\widetilde{X}$, and it outputs 1 if and only if $\widetilde{X} = \mathsf{MuSig.KAg}(L_{pk})$ and otherwise outputs 0.

Signature Verification.

$\mathsf{KAIAS.AVf}\ (\mathsf{params}, \{m_1, \cdots, m_n\}, \widetilde{\sigma}, \widetilde{X})$

Given a message $\{m_1, \cdots, m_n\}$, an aggregated signature $\widetilde{\sigma} = (\widetilde{s}, \widetilde{R}, \overline{R})$ and an aggregated public key $\widetilde{X}$, the verifier computes $\overline{c} \leftarrow \mathsf{KAIAS.MHAg}\ (\{m_1, \cdots, m_n\}, \overline{R}, \widetilde{X})$, and outputs 1 if and only if $g^{\widetilde{s}} = \widetilde{R}\widetilde{X}^{\overline{c}}$ and otherwise outputs 0.

Note that anyone who knows all individual public keys can verify the aggregated public key's correctness, so the verifier doesn't have to execute $\mathsf{KAIAS.KVf}$. Suppose a key aggregator computes and publishes the aggregated public key, and a different third party performs $\mathsf{KAIAS.KVf}$. and outputs 1, the aggregated public key is valid if one entity among of them is honest. Therefore, if the aggregated public key is verified once after executing $\mathsf{KAIAS.KAg}$, then $\mathsf{KAIAS.KVf}$ is not required in the subsequent verification.

We can confirm the following equations to confirm the correctness of KAIAS.

$$\begin{aligned} g^{\widetilde{s}} &= g^{s_1 d_1 + \cdots + s_n d_n} \\ &= g^{r_1 d_1 + c_1 d_1 a_1 x_1 + \cdots + r_n d_n + c_n d_n a_n x_n} \\ &= R_1^{d_1} \cdots R_N^{d_n} (X_1^{a_1} \cdots X_n^{a_n})^{\overline{c}} \\ &= \widetilde{R}\widetilde{X}^{\overline{c}} \end{aligned}$$

This scheme should also satisfy completeness.

$$\begin{aligned} &\mathsf{KAIAS.AVf}\ (\mathsf{params}, \{m_1, \cdots, m_n\}, \\ &\qquad \mathsf{MuSig.SAg}\ (\mathsf{params}, \{(c_1, \sigma_1), \cdots, (c_n, \sigma_n)\}), \\ &\qquad \mathsf{KAIAS.KAg}\ (L_{pk})) = 1 \end{aligned}$$

In the KAIAS scheme, there are two verification algorithms: aggregated public key verification and signature verification. The verifier accepts the given aggregated signature if both algorithms output 1; moreover, we can delegate the aggregated public key verification to a trusted third party such as a CA (If a third party passed an invalid aggregated public key, the signature verification using that aggregated public key fails, so the verifier can detect that either the aggregated signature or the aggregated public key is invalid).

Therefore, the verifier does not need to obtain an individual public key list from the signers by introducing this party and can execute signature verification using only the aggregated public key. This modification is effective in protecting the privacy of individual public keys, as discussed in Ref. [18].

3.2 Security Proof of KAIAS

As with the security proof in Refs. [4] and [18], we use the "Double Forking Lemma" technique to construct the DLP solver. In this technique, the adversary executes the forger algorithm defined in Generalized Forking Lemma twice. First, the adversary executes algorithm $\mathcal{B}$ which outputs the discrete logarithm of the aggregated public key apk using algorithms $\mathcal{A}$ and $\mathsf{Fork}^{\mathcal{A}}$. Second, the adversary executes algorithm $\mathcal{D}$ which outputs the discrete logarithm of the input public key pk^* using algorithms $\mathcal{B}$ and $\mathsf{Fork}^{\mathcal{B}}$.

Thm. 1. KAIAS is an unforgeable Multi-Signature scheme (as defined in Section 2.1.4, Def.2) in the random-oracle model if the DLP is hard. In other words, we can construct $(\tau_{\mathcal{D}}, \epsilon_{\mathcal{D}})$-solve DLP $\mathcal{D}$ by using KAIAS $(\tau, q_S, q_H, n, \epsilon)$-forger algorithm $\mathcal{F}$ where

$$\tau_{\mathcal{D}} = 4\tau + 4n\tau_{exp} + O(nq_T)$$
$$\epsilon_{\mathcal{D}} \geq \frac{\epsilon^4}{q_T^3} - \frac{16nq_T^2}{p} - \frac{3}{2^\ell}$$

where $q_T = q_S + q_H + 1$, and τ_{exp} is the time needed to compute an aggregated public key apk at most.

Proof. We first construct an algorithm $\mathcal{A}$ (as described in Lemma. 1) which takes y as input and outputs a forgery multi-signature using forger algorithm $\mathcal{F}$. we then construct an algorithm $\mathcal{B}$ that takes y as input, and outputs an aggregated public key apk which includes y and its discrete logarithm ω, using generalized forking algorithm $\mathsf{Fork}^{\mathcal{A}}$ on the algorithm $\mathcal{A}$ which outputs two forgery multi-signatures which are generated by the same inputs. Finally, we construct a discreate-logarithm algorithm $\mathcal{D}$ that takes y as an input and outputs its discrete logarithm x, using the outputs of $\mathsf{Fork}^{\mathcal{B}}$ on the algorithm $\mathcal{B}$, that means if we define the $(\tau, q_S, q_H, \epsilon)$-forger algorithm $\mathcal{F}$, we can construct the DLP-solver $\mathcal{D}$.

Construction of algorithm $\mathcal{A}$.

$\mathcal{A}$ takes as input $in = (y, h_{1,1}, \cdots, h_{1,q_H})$, the randomness $(h_{0,1}, \cdots, h_{0,q_H})$, and ρ as $\mathcal{A}$'s random tape, and runs $pk^* = y$ with $i_0 = 0, i_1 = 0$, and initialized empty hash tables T_0, T_1 and T_2. $\mathcal{A}$ is described as the following.

$H_0(\overline{R}, apk, m_i)$

If $T_0(\overline{R}, apk, m_i)$ is undefined, then $\mathcal{A}$ sets $T_0(\overline{R}, apk, m_i) = H_{0,i_0}$ and $i_0 \leftarrow i_0 + 1$. It returns $T_0(\overline{R}, apk, m_i)$.

$H_1(pk_i, L_{pk})$

If $pk^* \in L_{pk}$ and $T_1(pk^*, L_{pk})$ is undefined, then $\mathcal{A}$ sets $T_1(pk^*, L_{pk}) = h_{1,i_1}$ and $i_1 \leftarrow i_1 + 1$ and assigns $T_1(pk, L_{pk}) \leftarrow_\$ \mathbb{Z}_p$ for all $pk \in L_{pk} - \{pk^*\}$. $\mathcal{A}$ computes $apk \leftarrow \prod_{pk \in L_{pk}} pk^{T_1(pk, L_{pk})}$. If $\mathcal{F}$ already sent a query involving apk, then we say that $\mathbf{bad}_1$ has happened and $\mathcal{A}$ return $(0, \perp)$, otherwise it returns $T_1(pk_i, L_{pk})$.

$H_2(R)$

If $T_2(R)$ is undefined, $\mathcal{A}$ assigns $T_2(R) \leftarrow_\$ \mathbb{Z}_p$. If

(i) there exists another $R \neq R'$ s.t. $T_2(R) = T_2(R')$, or

(ii) $T_2(R)$ has been already used in the first round of the signing query,

then we say that event $\mathbf{bad}_2$ has happened and $\mathcal{A}$ returns $(0, \perp)$, otherwise it returns $T_2(R)$

Sign $(L_{pk}, \{m_1, \cdots, m_n\})$

In first step, $\mathcal{A}$ computes apk according to three round protocols.

(1) $\mathcal{A}$ generates a random value $t_i \leftarrow \mathbb{Z}_p$ and sends it to the aggregator.

(2) After receiving $\{t_j\}_{j \in \{1, \cdots, n\}}$ from the aggregator, $\mathcal{A}$ detects R_i s.t. $t_i = H_2(R_i)$. If $\mathcal{A}$ can't find such a value, it sets the random value $R_i \leftarrow_\$ \mathbb{G}$.

$\mathcal{A}$ assigns $s_i, c_i \leftarrow_\$ \mathbb{Z}_p$, simulates an internal query $a_i = H_1(pk^*, L_{pk})$, computes $R_i = g^{s_i}(pk^*)^{-a_i c_i}$ and $\overline{R} = \prod_{j=1}^n R_j$, and sets the tables as $T_2(R_i) \leftarrow t_i$, $T_0(\overline{R}, apk, m_i) \leftarrow c_i$. If $T_0(\overline{R}, apk, m_i)$ is already defined, we say that event $\mathbf{bad}_3$ has happened and $\mathcal{A}$ returns $(0, \perp)$, otherwise it returns $T_2(R)$. $\mathcal{A}$ sends R_i to the aggregator.

After receiving $\{R_j\}_{j \in \{1, \cdots, n\}}$ from the aggregator, $\mathcal{A}$ checks that $t_j = H_2(R_j)$ for each $j \neq i$.

(3) $\mathcal{A}$ checks that $apk = \prod_{i=1}^n X_i^{a_i}$, and sends (m_i, s_i, c_i) to the aggregator.

If the forger $\mathcal{F}$ outputs a valid forgery KAIAS signature $(\widetilde{s}, \widetilde{R}, \overline{R})$ corresponding to the signature of n messages $\{m_1, \cdots, m_n\}$ and a public key list of the signers $L_{pk} = \{pk_1, \cdots, pk_n\}$, then $\mathcal{A}$ computes $apk \leftarrow$ KAg (L_{pk}), $\overline{c} =$ KAIAS.MHAg $(\{m_1, \cdots, m_n\}, \overline{\sigma}, apk)$, and $a_i = H_1(pk_i, L_{pk})$ for each $i \in \{1, \cdots, n\}$.

$\mathcal{A}$ returns $(i_0, (\widetilde{s}, \widetilde{R}, \overline{R}, \overline{c}, apk, L_{pk}, a_1, \cdots, a_n))$ where satisfies $apk = \prod_{i=1}^n pk_i^{a_i}$, $g^{\widetilde{s}} = \widetilde{R}\widetilde{X^{\overline{c}}}$, and there exists $i \in \{1, \cdots, n\}$ s.t. $c_i = h_{0,i_0}$.

If $\mathcal{F}$ is a $(\tau, q_S, q_H, \epsilon)$-forger, then we can compute the lower bound $\epsilon_{\mathcal{A}}$, which is the probability of accepting $\mathsf{Fork}^{\mathcal{A}}$ with the running time $\tau_{\mathcal{A}}$.

First, if $\mathbf{bad}_1$ has happened, the random-oracle returns a $H_1(pk, L_{pk})$ which is a value in conflict in T_1 with the probability $\frac{(q_S+q_H+1)}{p}$. Thus, considering as q_H is a maximal number of random-oracle queries, we describe $Pr[\mathbf{bad}_1] = \frac{q_H(q_S+q_H+1)}{p}$.

Second, if $\mathbf{bad}_2$ has happened, the random-oracle returns a $H_2(R)$ where (i) $R \neq R'$ and $H_2(R) = H_2(R')$ with the probability $\frac{(q_S+q_H)^2}{2p}$, and (i) used value in the first round with the probability $\frac{q_S q_H}{p}$ in n signers, thus we describe $Pr[\mathbf{bad}_2] = \frac{(q_S+q_H)^2}{2p} + \frac{nq_S q_H}{p}$.

Third, if $\mathbf{bad}_3$ in n has happened, the random-oracle returns a $H_0(\overline{R}, apk, m_i)$ which is a value in conflict with T_0 with the probability $\frac{(q_S+q_H+1)}{p}$, which conflicts in T_1 for each $i \in \{1, \cdots, n\}$, so we describe $Pr[\mathbf{bad}_3] = \frac{nq_H(q_S+q_H+1)}{p}$ From the above,

$$\begin{aligned}
acc(\mathcal{A}) &= Pr[\mathcal{F}\,succeeds \wedge \overline{\mathbf{bad}_1} \wedge \overline{\mathbf{bad}_2} \wedge \overline{\mathbf{bad}_3}] \\
&\geq acc(\mathcal{F}) - Pr[\overline{\mathbf{bad}_1}] - Pr[\overline{\mathbf{bad}_2}] - Pr[\overline{\mathbf{bad}_3}] \\
&\geq \epsilon - \frac{q_H(q_S+q_H+1)}{p} - \left(\frac{(q_S+q_H)^2}{2p} + \frac{nq_S q_H}{p}\right) \\
&\quad - \frac{nq_H(q_S+q_H+1)}{p} \\
&\geq \epsilon - \frac{4nq_T^2}{p} = \epsilon - \delta = \epsilon_{\mathcal{A}}
\end{aligned}$$

where $\delta = \frac{4nq_T^2}{p}$.

Additionally, the running time of $\mathcal{A}$ is equal to the total running time of forger (t), aggregating of n keys $(n\tau_{exp})$, and all answering queries order $(O(n \cdot q_T))$. Therefore we can compute the running time of $\mathcal{A}$ at most as $\tau_{\mathcal{A}} = \tau + n\tau_{exp} + O(n \cdot q_T)$.

Construction of $\mathcal{B}$

Next, we construct algorithm $\mathcal{B}$ that runs the forking algorithm $\mathsf{Fork}^{\mathcal{A}}$ takes $(y, h_{1,1}, \cdots, h_{1,q_H})$ and random tape ρ as input, and $\mathcal{B}$ outputs

$$(i_0,(\widetilde{s},\widetilde{R},\overline{R},\overline{c},apk,L_{pk},a_1,\cdots,a_n),$$
$$(\widetilde{s}',\widetilde{R}',\overline{R}',\overline{c}',apk',L'_{pk},a'_1,\cdots,a'_n)).$$

Compare the query of $H_0(\overline{R},apk,m_i)$ in the first run of $\mathsf{Fork}^{\mathcal{A}}$ and $H_0(\overline{R}',apk',m'_i)$ in the second run of it, they are identical up to the view of $\mathcal{A}$ because of using the same inputs and random tapes, thus, $\overline{R} = \overline{R}'$ and $apk = apk'$, then apk and apk' are generated by same public key list, so $L_{pk} = L'_{pk}$, $(a_1,\cdots,a_n) = (a'_1,\cdots,a'_n)$, $\widetilde{R} = \widetilde{R}'$ and $\overline{c} \neq \overline{c}'$.

Using its outputs of $\mathsf{Fork}^{\mathcal{A}}$, we have the following two equations.

$$g^{\widetilde{s}} = \widetilde{R} \cdot apk^{\overline{c}}$$
$$g^{\widetilde{s}'} = \widetilde{R}'(apk')^{\overline{c}'} = \widetilde{R} \cdot apk^{\overline{c}'}$$

then we can compute $\omega \leftarrow (s - s')(\overline{c} - \overline{c}')^{-1} \bmod p$ s.t. $g^{s-s'} = g^{\omega(c-c')}$ which is the discreate-logarithm of $apk = g^{\omega}$.

Finally, $\mathcal{B}$ returns $(i_1,(\omega,L_{pk},a_1,\cdots,a_n))$, and we can compute the lower bound of the probability of accepting $\mathsf{Fork}^{\mathcal{B}}$ with the running time $\tau_{\mathcal{B}}$ from Section 2.1.2, Def.1 such as

$$\begin{aligned} acc(\mathcal{B}) &\geq acc(\mathcal{A}) \cdot \left(\frac{acc(\mathcal{A})}{q_T} - \frac{1}{2^{\ell}}\right) \\ &\geq \frac{(\epsilon-\delta)^2}{q_T} - \frac{(\epsilon-\delta)}{2^{\ell}} \\ &\geq \frac{\epsilon^2}{q_T} - \delta' = \epsilon_{\mathcal{B}} \end{aligned}$$

where $\delta' = 2\delta + \frac{1}{2^{\ell}}$. Note that the running time of $\mathcal{B}$ is twice the running time of $\mathcal{A}$, so we can compute $\tau_{\mathcal{B}} = 2\tau + 2n\tau_{exp} + O(n \cdot q_T)$

Construction of $\mathcal{D}$

As the above, we construct a DLP-solve algorithm $\mathcal{D}$ that runs the forking algorithm $Fork^{\mathcal{B}}$ takes y as input, and $\mathcal{B}$ outputs

$$(i_1,(\omega,L_{pk},a_1,\cdots,a_n),(\omega',L'_{pk},a'_1,\cdots,a'_n)).$$

Compare with the two executions of $\mathcal{B}$ in $\mathsf{Fork}^{\mathcal{B}}$, where i_1-th H_1 query $H_1(pk_{i_1},L_{pk})$ and $H_1(pk'_{i_1},L'_{pk})$ are identical up to the view of $\mathcal{B}$, so we have that $L_{pk} = L'_{pk}$. Since all values $T_1(pk,L_{pk})$ for $pk \neq pk^*$ are chosen randomly by $\mathcal{A}$ using the same inputs and random tapes, thus we obtain $a_i = a'_i$ for $i \neq i_1$, and $a_i \neq a'_i$ for $i = i_1$. By dividing the following two equations

$$apk = \prod_{i=1}^{n} pk_i^{a_i} = g^{\omega}$$
$$apk' = \prod_{i=1}^{n} pk_i^{a'_i} = g^{\omega'},$$

we have $(pk^*)^{a_{i_1}-a'_{i_1}} = g^{\omega-\omega'}$ and $\mathcal{D}$ can compute $x \leftarrow (\omega - \omega')(a_{i_1} - a'_{i_1})^{-1} \bmod p$ which is the DLP solutions of $y = pk^*$.

Finally, $\mathcal{D}$ takes y as input and outputs x s.t. $g^x = y$ with the probability at least of $\epsilon_{\mathcal{D}}$ and the running time $\tau_{\mathcal{D}}$ described as

$$\begin{aligned} acc(\mathcal{D}) &\geq acc(\mathcal{B}) \cdot \left(\frac{acc(\mathcal{B})}{q_T} - \frac{1}{2^{\ell}}\right) \\ &\geq \frac{(\epsilon^2/q_T - \delta')^2)}{q_T} - \frac{\epsilon^2/q_T - \delta'}{2^{\ell}} \\ &\geq \frac{\epsilon^4}{q_T^3} - \delta'' = \epsilon_{\mathcal{D}} \end{aligned}$$

where $\delta'' = 2\delta' + \frac{1}{2^{\ell}}$, and $\tau_{\mathcal{D}} = 4\tau + 4n\tau_{exp} + O(n \cdot q_T)$ □

4. Proposed Application

This section introduces our electronic signature application that uses KAIAS and can be applied to business tasks.

"Trust Services" are digital signature components constructed by signature-application technologies such as time-stamps, e-Seal (the signature issued by a corporation), and e-Delivery. These enables users to verify the completeness and reliability, which contains the data's evidence when signed and also who signed it. After publishing eIDAS regulations in the EU in 2014 [11], the criteria defined by eIDs (Electronic IDentification) and Trust Services were recognized. Therefore, it is expected that Trust Services will spread throughout the EU and in Japan and the US to achieve the Digital Single Market envisioned by the EU.

On the other hand, research on multi-signature schemes has been actively discussed to reduce the transaction data size using the multi-signature application in transaction technologies typified by Bitcoin. However, there has been less discussion on establishing the aggregation of each signers' approval using multi-signature schemes.

We introduce the following practical application in Trust Service for effectively applying KAIAS whose use is not restricted to transaction cases [*1].

4.1 Contracts among Companies

We introduce an effective KAIAS application that establishes a contract between two companies X and Y (Company X and Company Y, respectively), using an electronic signature.

We assume the contract process flow is as follows. Before signing the e-Seal, some approvers (the signers) in Company X attached their individual signatures to the contract document as a sign of approval (independent of e-Seal). Finally, Company X (the signer side) issues this document with an e-Seal which is a signature for verifying the contract's correctness signed by the responsible person in this contract. In response, Company Y (the verifier side) can confirm the document's validity by verifying its e-Seal using a public key corresponding to it.

In the above cases, the e-Seal signed by Company X allows Company Y to confirm only the validity of this document. However, the process assurance of this document signed by the individual approvers is outside the verifiers' scope. For example, Company Y's verifier cannot detect these documents' internal rewriting and cheating using only e-Seal.

To avoid these abuses, we use signature aggregation in KAIAS to define the binding between an e-Seal signed by Company X and individual signatures signed by some approvers. However, Company X has a motivation for not disclosing incomplete documents created before the final contract document regarding privacy and compliance.

We introduce "Message Hiding" as an extension of KAIAS to improve the trust of e-Seal to meet the following requirements for the above cases.

- Company X can keep the incomplete documents confidential from Company Y.

[*1] In the following use cases, we implicitly assume there exists a semi-honest third party as an aggregator.

- Company Y can verify that the final version has been issued by Company X according to the appropriate approval process without using the original incomplete documents.
- Company Y does not usually require incomplete documents for the verification but can verify their validity using a disclosure request.

Message Hiding conceals incomplete documents using the double hash chain and establishes the validity of its approval routes in Company X.

We now define the algorithms HCSign, HCVf, and DR in addition to KAIAS.

Hash Chain Signing.

KAIAS.HCSign (params, $\{m_1, \cdots m_n\}, \{sk_1, \cdots, sk_n\}, L_{pk}$)

This algorithm takes the same input as the KAIAS.Sign, and outputs normal KAIAS signature and hash chain parameters.

(1) After 3-round protocols in MuSig.Sign, the 1st signer computes

$$\begin{aligned} h_1 &= H_0(\overline{R}, \widetilde{X}, m_1) \\ c_1 &= h_1 \\ a_1 &= H_1(X_1, L_{pk}) \\ s_1 &= r_1 + c_1 a_1 x_1 \bmod p \end{aligned}$$

and sends (m_1, s_1, c_1, h_1) to the aggregator.

(2) k-th signer gives h_{k-1} from the aggregator and computes

$$\begin{aligned} h_k &= H_0(\overline{R}, \widetilde{X}, m_k) \\ c_k &= H_0(c_{k-1}, h_k) \\ a_k &= H_1(X_k, L_{pk}) \\ s_k &= r_k + c_k a_k x_k \bmod p \end{aligned}$$

and sends (m_k, s_k, c_k, h_k) to the aggregator.

(3) Finally, the aggregator computes an aggregated signature $\widetilde{\sigma}$ and the hash chain parameters $hcp = \{h_1, h_2, \cdots, h_{k-1}\}$.

Figure 1 shows the construction of Hash Chain Signing.

Hash Chain Verification.

KAIAS.HCVf ($hcp, m_n, \widetilde{\sigma}, c_n$)

This algorithm checks the verification of hash chain between $\{h_1, \cdots, h_{n-1}\}$ and c_n before checking the KAIAS.Verity. Parses $\widetilde{\sigma} = (\widetilde{s}, \widetilde{R}, \overline{R})$ and computes recursive as follows.

$$\begin{aligned} c'_1 &= h_1 \\ c'_2 &= H_0(c'_1, h_2) \\ &\vdots \\ c'_{n-1} &= H_0(c'_{n-2}, h_{n-1}) \\ h'_n &= H_0(\overline{R}, \widetilde{X}, m_n) \\ c'_n &= H_0(c'_{n-1}, h'_n) \end{aligned}$$

If $c'_n = c_n$ then verification is successful which means the input parameters hcp and m_n are definitely non-forged and generates them in sequentially ascending order. **Figure 2** shows the construction of the Hash Chain Verification.

Disclosure Requirements. DR ($hcp, m, \widetilde{\sigma}$) (1) Compute $h' = H_0(\overline{R}, \widetilde{X}, m)$

(2) Output 1 if a i exists s.t. $h' = h_i$ ($i \in \{1, \cdots, n-1\}$) in hcp, otherwise outputs 0.

Hash chain verification KAIAS.HCVf and KAIAS aggregated signature verification KAIAS.AVf enables verifiers to verify process order and unforgeability. However, since the verifier cannot obtain $m_1, \cdots, m_{n-1}$ following the requirements, it cannot determine whether $h_k = H_0(\overline{R}, \widetilde{X}, m_k)$ for $k = 1, \cdots, n-1$. In other words, h_k is actually calculated by m_k. Therefore, we introduce the algorithm called "Disclosure Requirements" KAIAS.DR. Using this algorithm, the verifier can confirm whether $h_1, \cdots, h_{n-1}$ contains a hash value for m_i only when the i-th signer publishes an arbitrary message m_i. Thus, the verifier can confirm that m_i is correctly included in the process of generating m_n.

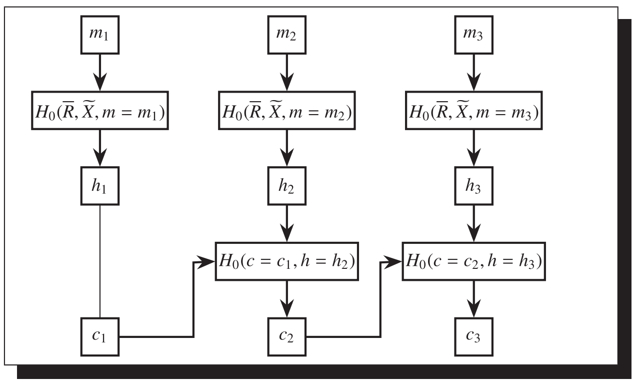
Using KAIAS with Message Hiding, both Company X and Company Y have the following advantages.

Signatures Aggregation

No matter how the structure of the internal approval path in Company X is constructed, the verifier can confirm that all signatures are valid by verifying just the aggregated signature. This representative signature can also be treated as evidence of passing Company X's approval flow, such as e-Seal.

Key Aggregation

As mentioned in Section 3.1, a third party can verify the aggregated public key instead of the verifier. Therefore, the verifier does not need to obtain all the individual public key certificates. Moreover, the verifier can verify in a constant time using the aggregated public key.



m_1
m_2
m_3
$H_0(\overline{R}, \widetilde{X}, m = m_1)$
$H_0(\overline{R}, \widetilde{X}, m = m_2)$
$H_0(\overline{R}, \widetilde{X}, m = m_3)$
h_1
h_2
h_3
$H_0(c = c_1, h = h_2)$
$H_0(c = c_2, h = h_3)$
c_1
c_2
c_3

Fig. 1 Construction of Hash Chain Signing.

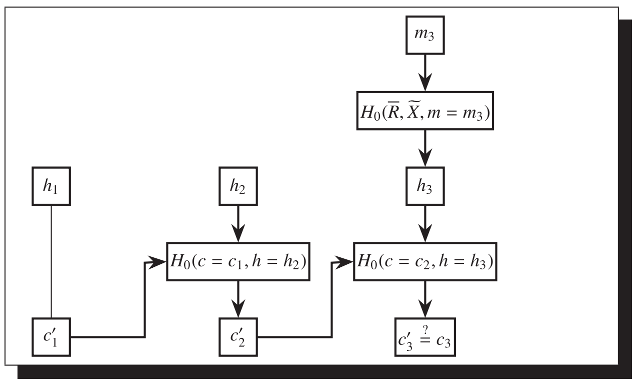


Fig. 2 Construction of Hash Chain Verification.

Disclosure Requirements

Company X keeps the incomplete documents confidential from Company Y using the hash chain until publishing it. Company Y can confirm the validity of whether the published document is really used in contract by using the hash chain.

4.2 Code Signings

In a software development company such as Apple [2], using Code Signing, which is the signature of the software's source code signed by the developer (the signer), the users (the verifier) can verify that this application was registered by a certain publisher and nobody has edited this code since last signed. On the other hand, in a version control service like Git, there is a commit signature to verify that all commits in the development process have not been forged or that commits by non-developers are included [13].

This paper considers the aggregate signature of the commit signatures as Code Signing and proposes a scheme that enables the users to verify the final version's source code and all commits.

Sign

Each j-th developer who has a key pair (sk_j, pk_j) for each $j \in \{1, \cdots, m\}$ commits a new commitment with a Schnorr signature σ_j signed by sk_j to the central server. Eventually, this server publishes all n $(n \geq m)$ commitments $\{m_1, \cdots, m_n\}$ with n signatures $\{\sigma_1, \cdots, \sigma_n\}$ for Code Signing.

Verify

To confirm the validity of the software, the user can verify n signatures corresponding to n commitments, and the entire developer's public key list $\{pk_1, \cdots, pk_m\}$.

In this naive usage case, however, the user has to run the verification algorithm n times which increases the cost of verification as the total n of commitments increases. We cannot regard that this cost as negligible for huge projects since for example in Google's repository, 25,000 developers commit 45 thousand commitments daily for 2 billion lines of code [23].

As explained above, the signer can aggregate n signatures into a single IAS signature by using IAS, but since IAS does not allow verification using aggregated public keys, users need a public key list L_{pk} for all developers. This public key list is used only among the developers to identify which part of the implementation each developer was responsible for. Therefore to justify using a certain public key for all users, a CA must issue a certificate for all public keys.

Using KAIAS, both developers and users have the following advantages.

Signatures Aggregation

No matter how many commits exist, the users can confirm the validity that all signatures are valid by verifying only a single aggregated signature.

Key Aggregation

As mentioned in Section 3.1, the users do not need all developers' public keys and can verify the signature with just a single aggregated public key provided by the developers. The verifier therefore does not require any information about the individual public keys.

5. Conclusion and Future Work

We present a new scheme called KAIAS which supports multiple message signing and public key aggregation while solving prior works issues in real use cases. Also, we propose two applications that utilize the advantages of KAIAS. We assume that the signature generated by the contract application corresponds to an e-Seal which is a digital signature issued by the company. It enables the verifier to confirm its authenticity and the internal process assurance by verifying the normal e-Seal. This extension of e-Seal is effective as a countermeasure against Business Email Compromise (BEC) which is a deception that makes a nonexistent company look like it exists.

In many Japanese companies with a deep-rooted culture of using paper signatures, it isn't easy to promote electronic signatures. Therefore, introducing the new values of digital signature application, such as the proposed, which were not possible with paper signatures, will lead to promoting use of DX (Digital Transformation) and archiving Society 5.0.

Finally, we conclude this section by discussing future works of KAIAS.

How secure is the aggregation message c_{agg}?

The c_{agg} is a parameter calculated using the Message Aggregation algorithm (KAIAS.MHAg) by the aggregator, the semi-honest entity. On the other hand, in some usage cases, the signer may be motivated to keep all messages $m_1, \cdots, m_n$ hidden from the aggregator. In this case, we modified how to generate the c_{agg} which each i-th signer calculates the message hash c_i in local, the signer sends the message hash c_i to the aggregator instead of the message m_i, and the aggregator calculates c_{agg} from the product of $c_1, \cdots, c_n$. It appears that the above motivation has been achieved. However, since c_{agg} is not a simple hash value, but a product of hash values, so we have to consider the risk that an adversary can change c_{agg} to a forged value by using Wagner's Algorithm to introduce a malicious message hash [29]. Since d, a wormhole product, is also a product of the hash values, the adversary can forge the same attack. The evaluation of Wagner's Attack on the product of hash values is a subject for future work. We also believe that the signer can avoid such an attack by introducing "tricks" such as Key Aggregation into Message Aggregation.

What about the use of deterministic nonce and modification to the 2-round Protocol?

As two advanced applications of MuSig, MuSig-DN [21] supports deterministic nonce instead of using a random number generator, and MuSig2 [20] supports a two-round protocol, respectively. Since these schemes are improvements related to the signing algorithm, we believe that updating KAIAS supports these modifications. The construction of such schemes will be a subject for future works.

References

[1] Ambrosin, M., Conti, M., Ibrahim, A., Neven, G., Sadeghi, A.-R. and Schunter, M.: Sana: Secure and scalable aggregate network attestation, *Proc. 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp.731–742, Association for Computing Machinery (2016).

[2] Code signing services. available from ⟨https://developer.apple.com/documentation/security/code_signing_services⟩.

[3] Bagherzandi, A., Cheon, J.H. and Jarecki, S.: Multisignatures secure under the discrete logarithm assumption and a generalized forking lemma, *ACM Conference on Computer and Communications Security*, 2008.

[4] Boneh, D., Drijvers, M. and Neven, G.: Compact Multi-signatures for Smaller Blockchains, Peyrin, T. and Galbraith, S. (Eds.), *Advances in Cryptology – ASIACRYPT 2018*, pp.435–464, Springer International Publishing (2018).

[5] Boldyreva, A., Gentry, C., O'Neill, A. and Yum, D.H.: Ordered multisignatures and identity-based sequential aggregate signatures, with applications to secure routing, *Proc. 14th ACM Conference on Computer and Communications Security* (*CCS '07*), pp.276–285, Association for Computing Machinery (2007).

[6] Bellare, M. and Neven, G.: Multi-signatures in the plain public-key model and a general forking lemma, *Proc. 13th ACM Conference on Computer and Communications Security* (*CCS '06*), pp.390–399, Association for Computing Machinery (2006).

[7] Boldyreva, A.: Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme, *PKC 2003*, pp.31–46 (2003).

[8] Boneh, D.: *Aggregate Signatures*, p.27, Springer (online) DOI: 10.1007/978-1-4419-5906-5_139 (2011).

[9] Drijvers, M., Edalatnejad, K., Ford, B., Kiltz, E., Loss, J., Neven, G. and Stepanovs, I.: On the security of two-round multi-signatures, *2019 IEEE Symposium on Security and Privacy* (*SP 2019*), pp.1084–1101, IEEE (2019).

[10] El Bansarkhani, M. and Sturm, J.: An efficient lattice-based multisignature scheme with applications to bitcoins, Foresti, S. and Persiano, G. (Eds.), *Cryptology and Network Security*, pp.140–155, Springer International Publishing (2016).

[11] Regulation (eu) no 910/2014 of the European parliament and of the council (2014), available from ⟨https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=uriserv:OJ.L_.2014.257.01.0073.01.ENG⟩.

[12] Japan Digital Trust Forum: Announcement of establishment of jdtf:japan digital trust forum (Japanese) (2020), available from ⟨https://d-trust.sfc.keio.ac.jp/jdtf/index.html⟩.

[13] GitHub, Inc.: Signing commits (2020), available from ⟨https://docs.github.com/en/github/authenticating-to-github/signing-commits⟩.

[14] Kojima, R., Yamamoto, D., Shimoyama, T., Yasaki, K. and Nimura, K.: A novel scheme of Schnorr multi-signatures for multiple messages with key aggregation, Barolli, L., Takizawa, M., Enokido, T., Chen, H.-C. and Matsuo, K. (Eds.), *Advances on Broad-Band Wireless Computing, Communication and Applications*, pp.284–295, Springer International Publishing (2021).

[15] Lysyanskaya, A., Micali, S., Reyzin, L. and Shacham, H.: Sequential aggregate signatures from trapdoor permutations, Cachin, C. and Camenisch, J.L. (Eds.), *Advances in Cryptology - EUROCRYPT 2004*, pp.74–90, Springer Berlin Heidelberg (2004).

[16] Lindemann, R. and Tiffany, E.: Fido uaf protocol specification (2017), available from ⟨https://fidoalliance.org/specs/fido-uaf-v1.1-ps-20170202/fido-uaf-protocol-v1.1-ps-20170202.pdf⟩.

[17] Fujitsu Laboratories Ltd.: Fujitsu develops digital trust management technology to ensure authenticity of business data (2020), available from ⟨https://www.fujitsu.com/global/about/resources/news/press-releases/2020/1006-01.html?_fsi=owQvoDf9&_fsi=owQvoDf9⟩.

[18] Maxwell, G., Poelstra, A., Seurin, Y. and Wuille, P.: Simple Schnorr multi-signatures with applications to bitcoin, *Designs, Codes and Cryptography* (2019).

[19] Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2009), available from ⟨https://metzdowd.com⟩.

[20] Nick, J., Ruffing, T. and Seurin, Y.: Musig2: Simple two-round Schnorr multi-signatures, Cryptology ePrint Archive, Report 2020/1261 (2020), available from ⟨https://eprint.iacr.org/2020/1261⟩.

[21] Nick, J., Ruffing, T., Seurin, Y. and Wuille, P.: Musig-DN: Schnorr multi-signatures with verifiably deterministic nonces, *Proc. 2020 ACM SIGSAC Conference on Computer and Communications Security* (2020).

[22] Okamoto, T.: A digital multisignature scheme using bijective public-key cryptosystems, *ACM Trans. Computer Systems*, Vol.6, No.4, pp.432–441 (1988).

[23] Potvin, R. and Levenberg, J.: Why google stores billions of lines of code in a single repository, *Comm. ACM*, Vol.59 No.7, pp.78–87 (2016).

[24] Pointcheval, D. and Stern, J.: Security proofs for signature schemes, *International Conference on the Theory and Applications of Cryptographic Techniques*, pp.387–398, Springer (1996).

[25] Ristenpart, T. and Yilek, S.: The power of proofs-of-possession: Securing multiparty signatures against rogue-key attacks, Springer-Verlag (2007).

[26] Schnorr, C.: Efficient signature generation by smart cards, *Journal of Cryptology*, Vol.4, pp.161–174 (1991).

[27] SSH.COM: Public key authentication for ssh, available from ⟨https://www.ssh.com/ssh/public-key-authentication⟩.

[28] Trust services and electronic identification (eID) (2018), available from ⟨https://ec.europa.eu/digital-single-market/en/trust-services-and-eid⟩.

[29] Wagner, D.: A generalized birthday problem, Yung, M. (Ed.), *Advances in Cryptology — CRYPTO 2002*, pp.288–304, Springer Berlin Heidelberg (2002).

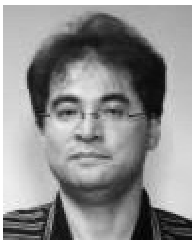
[30] Wuille, P., Nick, J. and Ruffing, T.: Schnorr signatures for secp256k1 (2020), available from ⟨https://github.com/bitcoin/bips/blob/master/bip-0340.mediawiki⟩.

[31] Yanai, N., Tso, R., Mambo, M. and Okamoto, E.: A certificateless ordered sequential aggregate signature scheme secure against super adversaries, *2011 3rd International Conference on Intelligent Networking and Collaborative Systems* (2011).

Rikuhiro Kojima was born in 1993. He received his B.Sc. and M.Sc. degree in information science in 2016, 2018 respectively from University of Tsukuba. He has been engaged in research on cryptography, trust as a service in FUJITSU LABORATORIES LTD. since 2018.

Dai Yamamoto received his B.E. and M.E. degrees in information networking from Osaka University, Osaka, Japan, in March 2005 and March 2007, respectively. He received his Ph.D. degree in engineering from The University of Electro-Communications, Tokyo, Japan, in March 2015. He joined FUJITSU LABORATORIES LTD. in April 2007 and has been engaged in research and development on hardware and software implementations of cryptosystems, physically unclonable functions, authentication protocols, IoT/OT security, and network security. He is currently leading research and development of Trust as a Service (TaaS) which is a new trust architecture in the Zero Trust era. He is serving as a secretary of IEICE ISEC Technical Committee, an associate editor of IEICE Transaction A, and a committee member of ISO/IEC JTC1/SC27/WG2. He was a visiting researcher at the KU Leuven, Belgium, from October 2011 to October 2012. He was awarded the Symposium on Cryptography and Information Security (SCIS) paper prize in 2014. He is a member of IEICE.

Takeshi Shimoyama received his B.S., M.S. degrees in mathematics from Yokohama City University in 1989 and 1991, respectively, and a D.E. degree in information and system engineering from Chuo University in 2000. He was a research engineer of FUJITSU LABORATORIES LTD. from 1991 until 2020. He has been a researcher by Special Appointment of the National Institute of Informatics since 2020. His current research interest is information science. He was awarded SCIS paper prize in 1997, IWSEC paper prize in 2007, OHM Technology Award in 2007, IPSJ Kiyasu Special Industrial Achievement Award in 2007, Innovation paper award at the 29th Symposium on Cryptography and Information Security in 2012. IPSJ Kiyasu Special Industrial Achievement Award in 2013, NTT DoCoMo Mobile Science Award in 2013, and IEICE Achievement Award in 2014.



Kouichi Yasaki received his B.E. and M.E. degrees in electronic engineering from University of Osaka Prefecture, Japan, in 1994 and 1996, respectively. He joined FUJITSU LABORATORIES LTD. in 1996. His current research includes data security.

Kazuaki Nimura is a project director at data & security laboratory in Fujitsu research. He received his M.E. of information and communication engineering from Tokyo Denki University. He joined Fujtsu in 1994 where he was engaged in research and development for personal computing system. He received Ph.D. in 2018, from Shizuoka University, Japan. His current research interest includes digital trust, data security, and web based platform technology.