PAPER Special Section on Next-generation Security Applications and Practice

Aggregate Signature Schemes with Traceability of Devices Dynamically Generating Invalid Signatures*

Ryu ISHII[†,††], **Kyosuke YAMASHITA**[††,†††], **Yusuke SAKAI**[††], ***Nonmembers***, **Tadanori TERUYA**[††], ***Member***, **Takahiro MATSUDA**[††], ***Nonmember***, **Goichiro HANAOKA**[††], ***Member***, **Kanta MATSUURA**[†], ***Senior Member***, ***and*** **Tsutomu MATSUMOTO**[††,††††a)], ***Member***

SUMMARY Aggregate signature schemes enable us to aggregate multiple signatures into a single short signature. One of its typical applications is sensor networks, where a large number of users and devices measure their environments, create signatures to ensure the integrity of the measurements, and transmit their signed data. However, if an invalid signature is mixed into aggregation, the aggregate signature becomes invalid, thus if an aggregate signature is invalid, it is necessary to identify the invalid signature. Furthermore, we need to deal with a situation where an invalid sensor generates invalid signatures probabilistically. In this paper, we introduce a model of aggregate signature schemes with interactive tracing functionality that captures such a situation, and define its functional and security requirements and propose aggregate signature schemes that can identify all rogue sensors. More concretely, based on the idea of Dynamic Traitor Tracing, we can trace rogue sensors dynamically and incrementally, and eventually identify all rogue sensors of generating invalid signatures even if the rogue sensors adaptively collude. In addition, the efficiency of our proposed method is also sufficiently practical.

key words: *sensor networks, aggregate signature schemes, fault-tolerant aggregate signature schemes, dynamic traitor tracing*

1. Introduction

Aggregate signature schemes allow multiple signatures to be aggregated into a short signature. In information systems with a large number of signed data, aggregate signature schemes enable us to aggregate the signatures on the communication channel into a single short signature, thus significantly reduce the communication cost. As observed in [2], [3], one example of such information systems is a sensor network where a large number of users and devices measure their environments and generate signatures to ensure the integrity of the measurements and transmit their signed data. However, if invalid signatures are produced due to a failure or a replacement of a sensor, an aggregate signature becomes invalid. If such a case, it is necessary to identify (and repair/exclude) a sensor that generated an invalid signature. At this point, if we know that there is only one such sensor, and that it always issues an invalid signature, it is trivial to identify the sensor. In practice, however, it is conceivable that multiple sensors that have failed or been replaced work in conjunction with each other to probabilistically generate invalid/valid signatures. At first glance, it seems trivial to detect invalid signatures if an aggregator verifies individual signatures before aggregation. In practice, however, aggregators are assumed to be relatively small devices on the communication path. In such a case, it would be difficult to verify many signatures sufficiently fast. For example, in Boneh-Gentry-Lynn-Shacham (BGLS) aggregate signature scheme [4], we need two pairing computations to verify a signature, which is the dominant factor of the computational cost. A pairing computation takes approximately 400 times as much as an addition on an elliptic curve.** If there are N signatures to be aggregated, $2N$ pairing operations are required to verify each of these signatures, but $N-1$ elliptic addition operations are sufficient just to aggregate them. We assume the aggregator as an inexpensive device that can aggregate fast but cannot verify sufficiently fast, and therefore cannot run individual signature verification before aggregation.

In this paper, we propose an aggregate signature scheme that can identify all rogue sensors even in such a case. More concretely, based on the idea of Dynamic Traitor Tracing [5], we can trace rogue sensors dynamically and incrementally, and finally identify all rogue sensors that generate invalid signatures.

1.1 Problem Setting

When we consider a sensor network in which a large number of devices send signed data periodically, it is natural to assume that the devices that generate invalid signatures may change from time to time due to failures. However, in previous works, the signature set to be aggregated is assumed to be constant during tracing, and there has been no method that can capture the situation where the devices generating invalid signatures could change from time to time. In this

Manuscript received February 17, 2022.
Manuscript revised May 6, 2022.
Manuscript publicized August 4, 2022.

[†]The authors are with the University of Tokyo, Tokyo, 153–8505 Japan.
[††]The authors are with the National Institute of Advanced Industrial Science and Technology (AIST), Tokyo, 135–0064 Japan.
[†††]The author is with the Osaka University, Suita-shi, 565–0871 Japan.
[††††]The author is with the Yokohama National University, Yokohama-shi, 240–8501 Japan.

*This paper is based on results obtained from a project, JPNP16007, commissioned by the New Energy and Industrial Technology Development Organization (NEDO). A preliminary version of this paper was presented at ACNS SCI 2021 [1]. This paper is added new results and the difference is described in Sect. 1.5.

a) E-mail: tsutomu@ynu.ac.jp

DOI: 10.1587/transinf.2022NGP0010

**In the setting shown in Sect. 5.4, it takes 0.679 microseconds per group operation and 286 micorseconds per pairing.

paper, we consider the situation where many users/devices send signed data periodically, an adversarial user/device set (that may generate invalid signatures) consists of multiple devices, and the user/device that generates the invalid signature could change dynamically from time to time. However, we assume that, when an aggregate signature is generated, at least one of the attackers sends an invalid signature.

Hartung et al. [6] proposed fault-tolerant aggregate signatures that allow a verifier to trace a certain number of invalid signatures. At first glance, it might seem that [6] can be used in our setting, but this is not the case. When [6] is used in the above setting, we need to assume that either (i) all rogue sensors always produce invalid signatures, or (ii) all signatures are temporarily stored before aggregation. Note that an adversary that captures the first condition does not meet our setting. Thus, the setting described above cannot be solved by a naive application of the method in [6].

1.2 Our Contribution

In this paper, we introduce *aggregate signature schemes with interactive tracing functionality* (ASIT scheme for short) that captures our problem setting, and define its functional and security requirements. In addition, we propose a generic construction of an ASIT scheme that combines an ordinary aggregate signature scheme and a dynamic traitor tracing (DTT) scheme [5]. A DTT is a method to trace adversaries who commit piracy in a contents distribution service. Since the definition in the original paper [5] is not so easy to work with, we also give a formalization of the syntax and security notions for DTT that is suitable and convenient for our purpose.

Our security definitions of ASIT scheme take into account all the situations of our problem setting mentioned earlier. Namely, it is possible to trace an adaptive collusion of rogue sensors. This situation was not captured in the previous work [6]. As an additional feature, the efficiency of our scheme is also sufficiently practical. For example, when $d = 10$ rogue sensors are mixed among $n = 100000$ sensors, each of which sends signed data every minute, they can always be traced within $R = d\log_2(n) + d \approx 176$ minutes. Furthermore, since our generic construction can be instantiated with any aggregate signature scheme and DTT, many different instantiations of an ASIT scheme can be obtained. We implement our scheme and conduct a preliminary experiment to confirm that the proposed scheme is indeed able to deal with the adaptive collusion.

1.3 Basic Idea of the Proposed Method

In the proposed ASIT scheme, we assume that a user/device periodically sends signed data to an aggregator. The aggregator generates an aggregate signature based on the feedback received from the verifier in the previous round for the signed data periodically sent by the user/device. The verifier, which has an internal state, verifies the aggregate signature from the aggregator, runs a tracing algorithm, updates the internal state and outputs a feedback and a set of excluded users/devices. This feedback is sent to the aggregator. These processes are repeated to finally find out all the users/devices that generated invalid signatures. The tracing algorithm of the proposed ASIT scheme is based on the tracing algorithm of the underlying DTT.

We consider two types of attacks on an ASIT scheme: a forgery of (aggregate) signatures and a generation of invalid signatures. The security notions against these attacks are EUF-CMA security and R-identifiability respectively, where the latter notion intuitively ensures that the set of users/devices that generated invalid signatures will be eventually identified within R rounds of interaction between the aggregator and the verifier. The security notion that guarantees that legitimate users/devices are not mistakenly traced is called correctness. The proposed method satisfies all of these security notions. For the formal definitions, see Sect. 4.

1.4 Related Work

The first aggregate signature scheme was proposed by Boneh et al. [4], which is in the random oracle model and uses bilinear maps. Hohenberger et al. [7] gave an aggregate signature scheme using multilinear maps in the standard model. These schemes can aggregate individual signatures as well as already aggregate signatures in any order. There are other types of aggregate signature schemes. One is sequentially aggregate signature, first proposed by Lysyanskaya et al. [8] in the random oracle model. Since then, a number of schemes have been proposed both in the random oracle model [8]–[10] and in the standard model [11], [12]. Another type is aggregate signature with synchronized aggregation, first proposed by Gentry and Ramzan [13] (in the identity-based setting) in the random oracle model. Again, since then, several constructions have been proposed both in the random oracle model [2], [13] and the standard model [2].

Hartung et al. [6] proposed fault-tolerant aggregate signatures that allow a verifier to trace a certain number of invalid signatures using a cover-free family. Sato et al. [14]–[16] proposed an aggregate message authentication code and an aggregate signature scheme with interactive tracing functionality using adaptive group tests.

1.5 Outline

In Sect. 2, we define basic notation and review standard aggregate signature schemes. In Sect. 3, we review and formulate DTT. In Sect. 4, we define the functional and security requirements of ASIT scheme that captures the problem setting of this study. In Sect. 5, we give a generic construction of an ASIT scheme combining a standard aggregate signature scheme and a DTT, prove its security and implement the proposed scheme. Finally, in Sect. 6, we conclude this work and discuss future works.

An additional result from the preliminary version is

given in Sect. 5.4: We add an experimental result for the proposed scheme and show that our implementation efficiently traces devices that dynamically generate invalid signatures.

2. Preliminaries

2.1 Notation

Throughout this paper, we let $\lambda \in \mathbb{N}$ be a security parameter, $n \in \mathbb{N}$ be the number of users, $[n] := \{1, \ldots, n\}$, ϵ be an empty string, $\emptyset$ be an empty set, $\mathrm{poly}(\cdot)$ be a polynomial function, $\mathrm{negl}(\cdot)$ be a negligible function, and $\mathcal{M} = \mathcal{M}(\lambda)$ be a message space. We say an algorithm is probabilistic polynomial time (PPT) if it is a probabilistic algorithm and its running time is polynomial in λ. We denote a subroutine func of an algorithm X by $X.\mathsf{func}$.

Let U be a set. We say P is a partition of the set U if P satisfies the followings: $P = (S_1, \ldots, S_p)$ where $p \in [|U|]$, $S_1, \ldots, S_p \in 2^U \setminus \{\emptyset\}$, $\bigcup_{i\in[p]} S_i = U$, and for every $i \neq j$ $(i, j \in [p])$, it holds that $S_i \cap S_j = \emptyset$. We denote by $|P| = p$.

2.2 Aggregate Signature Schemes

Here we review the syntax, correctness, and security definitions of aggregate signature schemes. We focus on an aggregate signature scheme which aggregates only one message and signature pair per one verification key[†].

Definition 2.1 (Aggregate Signature): An aggregate signature scheme consists of the five PPT algorithms (KeyGen, Sign, Verify, Agg, $\mathsf{AggVerify}$) that work as follows:

$\mathsf{KeyGen}(1^\lambda) \to (\mathsf{pk}, \mathsf{sk})$**:** KeyGen is the key generation algorithm that takes a security parameter 1^λ as input, and outputs a public/secret key pair $(\mathsf{pk}, \mathsf{sk})$.

$\mathsf{Sign}(\mathsf{sk}, m) \to \sigma$**:** Sign is the signing algorithm that takes a secret key sk and a message $m \in \mathcal{M}$ as input, and outputs a signature σ.

$\mathsf{Verify}(\mathsf{pk}, m, \sigma) \to 1/0$**:** Verify is the verification algorithm for non-aggregate signatures: It that takes a public key pk, a message m, and a signature σ as input, and outputs 1 (valid) or 0 (invalid).

$\mathsf{Agg}(\{(\mathsf{pk}_i, m_i, \sigma_i)\}_i) \to \tau$**:** Agg is the aggregation algorithm that takes a set of triplets of a public key, a message and a signature $\{(\mathsf{pk}_i, m_i, \sigma_i)\}_i$ as input, and outputs an aggregate signature τ.

$\mathsf{AggVerify}(\{(\mathsf{pk}_i, m_i)\}_i, \tau) \to 1/0$**:** $\mathsf{AggVerify}$ is the verification algorithm for aggregate signatures: It takes a set of pairs of a public key and a message $\{(\mathsf{pk}_i, m_i)\}_i$ and an aggregate signature τ as input, and outputs 1 (valid) or 0 (invalid).

[†]In general, aggregate signature schemes can aggregate multiple signatures even if they are generated under the same key, but for simplicity, we do not introduce such version in this paper.

Definition 2.2 (Correctness): An aggregate signature scheme $\Sigma_{\mathrm{AS}} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Agg}, \mathsf{AggVerify})$ satisfies correctness if for any $\lambda \in \mathbb{N}$, any $n = \mathrm{poly}(\lambda)$, and any $m_1, \ldots, m_n \in \mathcal{M}$, it holds that

$$\Pr\left[\begin{array}{l| l} 1 \leftarrow \mathsf{AggVerify}(\{(\mathsf{pk}_i, m_i)\}_{i\in[n]}, \tau) & \\ & \forall i \in [n],\ (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(1^\lambda), \\ & \text{and } \sigma_i \leftarrow \mathsf{Sign}(\mathsf{sk}_i, m_i); \\ & \tau \leftarrow \mathsf{Agg}(\{(\mathsf{pk}_i, m_i, \sigma_i)\}_{i\in[n]}) \end{array}\right] = 1.$$

For security, we consider EUF-CMA (Existential UnForgeability against Chosen Message Attacks) security in the model where all generated key pairs are generated honestly (honest-key model).

Definition 2.3 (EUF-CMA security): An aggregate signature scheme $\Sigma_{\mathrm{AS}} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Agg}, \mathsf{AggVerify})$ satisfies EUF-CMA security if for any $\lambda \in \mathbb{N}$, any $n = \mathrm{poly}(\lambda)$ and any PPT adversary $\mathcal{A}$, it holds that $\Pr\left[\mathsf{ExpAS}^{EUF\text{-}CMA}_{\Sigma_{\mathrm{AS}},\mathcal{A}}(\lambda, n) = 1\right] = \mathrm{negl}(\lambda)$ where $\mathsf{ExpAS}^{EUF\text{-}CMA}_{\Sigma_{\mathrm{AS}},\mathcal{A}}(\lambda, n)$ is the following experiment:

$$\begin{array}{l} \mathsf{ExpAS}^{EUF\text{-}CMA}_{\Sigma_{\mathrm{AS}},\mathcal{A}}(\lambda, n) \\ \hline \forall i \in [n],\ (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \Sigma_{\mathrm{AS}}.\mathsf{KeyGen}(1^\lambda); \\ Q := \emptyset; \\ (\{m_i\}_{i\in S}, \tau, S) \leftarrow \\ \mathcal{A}^{\Sigma_{\mathrm{AS}}.\mathsf{Sign}(\mathsf{sk}_1,\cdot)}(\mathsf{pk}_1, \{(\mathsf{pk}_i, \mathsf{sk}_i)\}_{i\in[n]\setminus\{1\}}) : \\ \text{Output 1 if } S \subseteq [n],\ 1 \in S,\ m_1 \notin Q \text{ and} \\ \Sigma_{\mathrm{AS}}.\mathsf{AggVerify}(\{(\mathsf{pk}_i, m_i)_{i\in S}, \tau\}) = 1, \\ \text{else output } 0 \end{array}$$

where when $\mathcal{A}$ makes a query $m \in \mathcal{M}$ to the signing oracle $\Sigma_{\mathrm{AS}}.\mathsf{Sign}(\mathsf{sk}_1, \cdot)$, it computes $\sigma \leftarrow \Sigma_{\mathrm{AS}}.\mathsf{Sign}(\mathsf{sk}_1, m)$, sends σ to $\mathcal{A}$, and sets $Q \leftarrow Q \cup \{m\}$.

Note that in the experiment, the user index 1 is used as a challenger user whose secret key is unknown to an adversary, and the remaining keys $(\mathsf{pk}_i, \mathsf{sk}_i)_{i\in[n]\setminus\{1\}}$ are directly given to $\mathcal{A}$. Thus, the signing oracle is provided only for the index 1.

3. Dynamic Traitor Tracing

A Dynamic Traitor Tracing (DTT) [5] is a method for tracing piracy by having a distributor of a video or another distribution service that delivers a variant of the content with a (generally) different digital watermark for each user. The distributor adaptively generates new watermarks when it finds a pirate version, repeats this process until it traces all of the pirates, and when the distributor finally identifies a pirate, it excludes the pirate from the user set.

In this section, we give our own formalization of DTT that is suitable and convenient for our purpose, based on the treatment in [5]. For DTT, we define the following two security requirements: *R-identifiability*, which ensures that the distributor can identify all pirates with R or fewer deliveries, and *completeness*, which ensures that the distributor does not exclude legitimate users. After defining the syntax

of DTT in Sect. 3.1, we define these two security notions in Sect. 3.2. Finally, we present two concrete schemes proposed by Fiat and Tassa [5] in Sect. 3.3.

3.1 Syntax

We define the syntax of a DTT as follows.

Definition 3.1 (Dynamic Traitor Tracing): A dynamic traitor tracing (DTT) scheme consists of the two PPT algorithms (Initialize, Trace) that work as follows:

$\mathsf{Initialize}(1^\lambda, 1^n) \to (\alpha, P)$**:** Initialize is the initialization algorithm that takes the security parameter 1^λ and the number of users 1^n (in the unary form) as input, and outputs an internal state α and a partition P of the user set $[n]$.

$\mathsf{Trace}(\alpha, i) \to (\alpha', P', V)$**:** Trace is the tracing algorithm that takes an internal state α and an index i as input, and outputs the new internal state α' and set partition P' for the next round, and a traced pirate set V.

This syntax captures the following usage scenario: First, the distributor executes Initialize to generate the initial partition $P = (S_1, \ldots, S_p)$ (for some p) of the user set $[n]$ (and initial internal state α), generates different watermarks for each element S_i in the partition P, and sends each user a watermarked content that corresponds to a subset S_i in P to which the user belongs. [†] Note that each subset S_i in the partition P is uniquely indexed. The collusion of pirate users chooses one of the watermarked contents sent by the distributor, and delivers it as a pirated version. The distributor identifies which watermark is embedded to the pirated version, and executes Trace based on the index of the pirate version and the distributor's internal state to determine the new partition for the next round. If a subset in the partition consists of only a single user (in other words, the distributor identifies a pirate), then Trace excludes the pirate. These procedures are repeated to exclude all pirates.

3.2 Security Requirements

Here, we formally define security requirements of DTT. In our model, we assume that an adversary declares a set C (where $|C| \leq d$) of pirates at first. Therefore, we consider DTTs that only excludes the pirates in this set. Furthermore, we assume that a pirate rebroadcasts the content that it receives. In other words, we do not consider the case where a pirate eavesdrops other user's content and rebroadcasts it. We remark that the same restrictions were implicitly put in [5]. We leave a definition of DTTs in the model where the set of traitors dynamically changes as an interesting future work.

As mentioned earlier, we define two security notions for DTT: *R-identifiability*, which ensures that a distributor can identify all the pirates within R (or less) rounds, and *completeness*, which ensures that a distributor does not trace legitimate users as pirates. These are defined using the following experiment $\mathsf{ExpDTT}_{\Sigma_{\mathrm{DTT}},\mathcal{A}}(\lambda, n)$ in which a stateful adversary $\mathcal{A}$ is executed.

$$\frac{\mathsf{ExpDTT}_{\Sigma_{\mathrm{DTT}},\mathcal{A}}(\lambda, n)}{\begin{array}{c}(\alpha_1, P_1) \leftarrow \Sigma_{\mathrm{DTT}}.\mathsf{Initialize}(1^\lambda, 1^n);\\ C \leftarrow \mathcal{A}(\alpha_1, P_1); t := 1; W := \emptyset; \text{run } \mathcal{A}^{O_T(\cdot)}(\alpha_1, P_1) :\\ \text{Output } (W, C)\end{array}}$$

where $\mathcal{A}$ may adaptively make multiple queries i_t to the tracing oracle O_T. However, $\mathcal{A}$'s t-th O_T query i_t must satisfy $i_t \in [|P_t|]$ and $S_{i_t,t} \cap C \neq \emptyset$, where α_t is the internal state and $P_t = (S_{1,t}, \ldots, S_{p_t,t})$ (for some natural number p_t) denotes the partition after $\mathcal{A}$'s $(t-1)$-th O_T query is answered. Given the t-th O_T query i_t from $\mathcal{A}$, O_T runs $(\alpha_{t+1}, P_{t+1}, V_t) \leftarrow \Sigma_{\mathrm{DTT}}.\mathsf{Trace}(\alpha_t, i_t)$, and returns $(\alpha_{t+1}, P_{t+1}, V_t)$ to $\mathcal{A}$. Then, O_T updates $W \leftarrow W \cup V_t$ and $t \leftarrow t + 1$. We remark that W in the output of the experiment is that at the point $\mathcal{A}$ halts.

Definition 3.2 (R-Identifiability): A DTT Σ_{DTT} satisfies R-identifiability if for any $\lambda \in \mathbb{N}$, any $n = \mathrm{poly}(\lambda)$, and any PPT adversary $\mathcal{A}$, it holds that

$$\Pr\left[C \not\subseteq W \,\middle|\, (W, C) \leftarrow \mathsf{ExpDTT}_{\Sigma_{\mathrm{DTT}},\mathcal{A}}(\lambda, n) \wedge t \geq R\right] = \mathrm{negl}(\lambda).$$

where t is the value of the counter when $\mathcal{A}$ stops.

Definition 3.3 (Completeness): A DTT Σ_{DTT} satisfies completeness if for any $\lambda \in \mathbb{N}$, any $n = \mathrm{poly}(\lambda)$, and any PPT adversary $\mathcal{A}$, it holds that

$$\Pr\left[([n] \setminus C) \cap W \neq \emptyset \,\middle|\, (W, C) \leftarrow \mathsf{ExpDTT}_{\Sigma_{\mathrm{DTT}},\mathcal{A}}(\lambda, n)\right] = \mathrm{negl}(\lambda).$$

3.3 Instantiations

Fiat and Tassa [5] proposed three concrete DTT. Here, we present the second scheme (FT-2) and the third scheme (FT-3) of [5], which are binary search-based methods. The DTT FT-2 and FT-3 are described in Fig. 1. Observe that in each algorithm, I is the set of users that are not in the tracing process. We call such a set as an *innocent set*. For these schemes, Fiat and Tassa [5] showed the following theorems.

Theorem 3.1: Let n be the number of users and d be the number of pirates. Then, FT-2 excludes all of the pirates within $d \log_2 n + d$ rounds. Furthermore, FT-2 uses at most $2d + 1$ variants in each round.

Theorem 3.2: Let n be the number of users and d be the number of pirates. Then, FT-3 excludes all of the pirates within $2 \cdot 3^d d \log_2 n + d$ rounds. Furthermore, FT-3 uses at most $d + 1$ variants in each round.

We now show that FT-2 and FT-3 satisfy the security notions formalized in this paper, namely R-identifiability and completeness, with an appropriate choice of R for each

[†] The mechanism for watermarking contents is detached from the syntax and beyond the scope of this primitive, which is the same treatment as in [5].

Initialize$(1^\lambda, 1^n)$:
$I := [n]$; $P_1 := (I)$; $\alpha_1 := P_1$
return (α_1, P_1)

Trace(α_t, i) :
$P := \alpha_t$; $p := (|P| - 1)/2$; $V := \emptyset$
parse $(Q_j)_{j\in[|P|]} \leftarrow P$
if $Q_i = I$ **then**
$(L_{p+1}, R_{p+1}) \leftarrow$ Halve(I); $I \leftarrow \emptyset$
$P \leftarrow P \cup (\{L_{p+1}\} \cup \{R_{p+1}\})$
$p \leftarrow p + 1$
else if $\exists j \in [p] : Q_i = L_j$ **then**
$\star$ $I \leftarrow I \cup R_j$
$\star$ **if** $|L_j| = 1$ **then**
$\star$ $V \leftarrow L_j$; $p \leftarrow p - 1$; $P \leftarrow P \setminus (\{L_j\} \cup \{R_j\})$
$\star$ **else** $(L_j, R_j) \leftarrow$ Halve(L_j)
else if $\exists j \in [p] : Q_i = R_j$ **then**
Execute the lines marked with $\star$ with switching L_j and R_j.
$P_{t+1} := P$; $\alpha_{t+1} := P$; $V_t := V$
return $(\alpha_{t+1}, P_{t+1}, V_t)$

Halve(I) :
$A :=$ the first $\lfloor |I|/2 \rfloor$ elements of I
$B := I \setminus A$
return (A, B)

Initialize$(1^\lambda, 1^n)$:
$I := [n]$; $P_1 := (I)$; $S_1 := P_1$; $\alpha_1 := (P_1, S_1)$
return (α_1, P_1)

Trace(α_t, i) :
parse $(P, S) \leftarrow \alpha_t$; $p = (|S| - 1)/2$; $V := \emptyset$
parse $(Q_j)_{j\in[|P|]} \leftarrow P$; $k := |P| - 1$
if $k = 2p$ or $(k < 2p$ and $i \in [k])$ **then**
if $Q_i = I$
$(L_{p+1}, R_{p+1}) \leftarrow$ Halve(I); $I \leftarrow \emptyset$
$S \leftarrow S \cup (\{L_{p+1}\} \cup \{R_{p+1}\})$; $p \leftarrow p + 1$
if $p > k$ **then** $k \leftarrow k + 1$
else if $\exists j \in [p] : Q_i = L_j$ **then**
$\star$ $I \leftarrow I \cup R_j$
$\star$ **if** $|L_j| = 1$ **then**
$\star$ $V \leftarrow L_j$; $S \leftarrow S \setminus (\{L_j\} \cup \{R_j\})$; $p \leftarrow p - 1$; $k \leftarrow k - 1$
$\star$ **else** $(L_j, R_j) \leftarrow$ Halve(L_j)
else if $\exists j \in [p] : Q_i = R_j$ **then**
Execute the lines marked with $\star$ with switching L_j and R_j.
else if $t = \sum_{k'=0}^{k} \binom{2p+1}{k'}$ **then** $k \leftarrow k + 1$
for $i := 1$ to k **do**
if $i \le p$ **then** $T_i := \{L_i, R_i\}$
else $T_i := S \setminus (\{Q_j\}_{j\in[i-1]})$
$Q'_i :=$ a randomly selected set from T_i
$Q'_{k+1} := [n] \setminus \bigcup_{i\in[k]} Q'_i$
$P_{t+1} := (Q'_i)_{i\in[k+1]}$; $S_{t+1} := S$
$\alpha_{t+1} := (P, S)$; $V_t := V$
return $(\alpha_{t+1}, P_{t+1}, V_t)$

Fig. 1 The descriptions of the DTT by Fiat and Tassa [5]: FT-2 (left-top), FT-3 (right), and the subalgorithm Halve used in both of the schemes (left-bottom).

scheme.

Lemma 3.1: FT-2 satisfies both $(d \log_2 n + d)$-identifiability and completeness.

Proof 3.1: Due to Theorem 3.1 and the assumption that at least one pirate does piracy in each round, $(d \log_2 n + d)$-identifiability is trivial if completeness is satisfied. Thus, we only prove completeness. It is sufficient to prove that FT-2 never excludes a valid user. Let L_i and R_i be singleton sets that appear during the execution of FT-2 (i.e., $|L_i| = |R_i| = 1$). Since we are assuming that a pirate does not rebroadcast other user's content, the users in the sets are excluded if and only if the corresponding variants are detected by the distributor. (Note that if they never rebroadcast the content anymore, we regard them as valid users.) Therefore, FT-2 never excludes a valid user, which guarantees completeness. □

Lemma 3.2: FT-3 satisfies both $(2 \cdot 3^d d \log_2 n + d)$-identifiability and completeness.

We skip the proof of Lemma 3.2, since it is almost the same as the proof of Lemma 3.1.

4. Aggregate Signature with Interactive Tracing Functionality

In this section, we give formal definitions for aggregate signature schemes with interactive tracing functionality (ASIT scheme). We define the syntax of ASIT scheme and the correctness of subalgorithms in Sect. 4.1. We then define the security and functionality requirements for ASIT scheme in Sect. 4.2, which include EUF-CMA security (in the honest-key model), R-identifiability, and correctness.

4.1 Syntax

Definition 4.1 (ASIT): An ASIT scheme consists of the six PPT algorithms (KeyGen, Sign, Agg, Verify, PartVerify, Trace) that work as follows:

KeyGen$(1^\lambda) \to$ (pk, sk)**:** KeyGen is the key generation algorithm that takes a security parameter 1^λ as input, and outputs a public/secret key pair (pk, sk).

Sign$(\mathsf{sk}, m) \to \sigma$**:** Sign is the signing algorithm that takes a secret key sk and a message $m \in \mathcal{M}$ as input, and outputs a signature σ.

Verify$(\mathsf{pk}, m, \sigma) \to 1/0$**:** Verify is the verification algorithm (for a non-aggregated signature) that takes a public key pk, a message m, and a signature σ as input, and outputs either 1 (valid) or 0 (invalid).

Agg$(f, \{(\mathsf{pk}_i, m_i, \sigma_i)\}_i) \to \tau$**:** Agg is the aggregation algorithm: It takes as input a feedback f (from the previous round) and a set of triplets of a public key, a message and a signature $\{(\mathsf{pk}_i, m_i, \sigma_i)\}_i$. Then, it outputs an aggregate signature τ.

PartVerify$(\beta, \{(\mathsf{pk}_i, m_i)\}_i, \tau, j) \to 1/0$**:** PartVerify is the "partial verification" algorithm that is used for defining EUF-CMA security. It takes as input an internal state β,

$\mathsf{ExpASIT}^{\mathrm{Trace}}_{\Sigma_{\mathrm{ASIT}}}(\lambda, n, \{m_i\}_{i\in[n]})$:
for $i \coloneqq 1$ to n **do**
  $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \Sigma_{\mathrm{ASIT}}.\mathsf{KeyGen}(1^\lambda)$
$t \coloneqq 1; V_1 \coloneqq \epsilon; \beta_1 \coloneqq \epsilon; f_0 \coloneqq \epsilon$
while true
  for $i \coloneqq 1$ to n **do**
    $\sigma_{i,t} \leftarrow \Sigma_{\mathrm{ASIT}}.\mathsf{Sign}(\mathsf{sk}_i, m_{i,t})$
  $\tau_t \leftarrow \Sigma_{\mathrm{ASIT}}.\mathsf{Agg}(f_{t-1}, \{(\mathsf{pk}_i, m_{i,t}, \sigma_{i,t})\}_{i\in[n]})$
  $(\beta_{t+1}, f_t, V_t) \leftarrow \Sigma_{\mathrm{ASIT}}.\mathsf{Trace}(\beta_t, \{(\mathsf{pk}_i, m_{i,t})\}_{i\in[n]}, \tau_t)$
  $t \leftarrow t + 1$

$\mathsf{ExpASIT}^{\mathrm{PartVrf}}_{\Sigma_{\mathrm{ASIT}}}(\lambda, n, \beta, j, \{m_i\}_{i\in[n]})$:
for $i \coloneqq 1$ to n **do**
  $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \Sigma_{\mathrm{ASIT}}.\mathsf{KeyGen}(1^\lambda)$
  $\sigma_i \leftarrow \Sigma_{\mathrm{ASIT}}.\mathsf{Sign}(\mathsf{sk}_i, m_i)$
Let f be the feedback and V be the set of
        traced users that are determined by β
$\tau \leftarrow \Sigma_{\mathrm{ASIT}}.\mathsf{Agg}(f, \{(\mathsf{pk}_i, m_i, \sigma_i)\}_{i\in[n]\setminus V})$
$v \leftarrow \Sigma_{\mathrm{ASIT}}.\mathsf{PartVerify}(\beta, \{(\mathsf{pk}_i, m_i)\}_{i\in[n]\setminus V}, \tau, j)$
return v

Fig. 2 The experiments used for defining the correctness of an ASIT scheme.

a set of tuples of a public key and a message $\{(\mathsf{pk}_i, m_i)\}_i$, an aggregate signature τ, and a user index j. Then, it outputs either 1 (indicating that τ is valid with respect to the user index j) or 0 (indicating that τ is invalid with respect to the user index j).

$\mathsf{Trace}(\beta, \{(\mathsf{pk}_i, m_i)\}_i, \tau) \to (\beta', f, V)$**:** Trace is the tracing algorithm: It takes as input an internal state β, a set of tuples of a public key and a message $\{(\mathsf{pk}_i, m_i)\}_i$, and an aggregate signature τ. It then outputs the new internal state β' (for the next round), a feedback f, and a traced user set V (which could be empty). It is required that a feedback f and a traced user set V can be uniquely retrieved from an internal state β.

This syntax captures the following usage scenario: First, all signers execute KeyGen. Next, each signer periodically runs Sign, and sends signed data to the aggregator. The aggregator executes Agg on the signed data periodically received from the signers, based on the feedback received from the verifier in the previous round. The verifier executes Trace on the aggregate signature received from the aggregator, updates the internal state, and outputs the feedback and the traced user set V. (Note that V is the set of users providing an invalid signature that the verifier identifies and thus the users in V are excluded from the user set used in the aggregation from the next round.) This feedback is sent to the aggregator.

Readers may wonder that the above definition lacks the verification algorithm AggVerify for an aggregate signature. However, we implicitly assume that Trace works as the verification algorithm. That is, if Trace outputs $V \neq \emptyset$, it implies that the given aggregate signature τ is invalid. We remark that PartVerify is an algorithm that is used only to define EUF-CMA security.

Definition 4.2 (Correctness of Trace): Let Σ_{ASIT} be an ASIT scheme. The algorithm $\Sigma_{\mathrm{ASIT}}.\mathsf{Trace}$ satisfies correctness if for any $\lambda \in \mathbb{N}$, any $n = \mathrm{poly}(\lambda)$, any $t \in \mathbb{N}$, and any $m_1, \ldots, m_n \in \mathcal{M}$, it holds that $\Pr[V_t = \emptyset] = 1$ where V_t is the value in the experiment $\mathsf{ExpASIT}^{\mathrm{Trace}}_{\Sigma_{\mathrm{ASIT}}}(\lambda, n, \{m_i\}_{i\in[n]})$ described in Fig. 2.

Definition 4.3 (Correctness of PartVerify): Let Σ_{ASIT} be an ASIT scheme. The algorithm $\Sigma_{\mathrm{ASIT}}.\mathsf{PartVerify}$ satisfies correctness if for any $\lambda \in \mathbb{N}$, any $n = \mathrm{poly}(\lambda)$, any possible form of internal state β, any $j \in [n]$, and any $m_1, \ldots, m_n \in \mathcal{M}$, it holds that $\Pr[v = 1] = 1$ where v is an output of the experiment $\mathsf{ExpASIT}^{\mathrm{PartVrf}}_{\Sigma_{\mathrm{ASIT}}}(\lambda, n, \beta, j, \{m_i\}_{i\in[n]})$ described in Fig. 2.

4.2 Security Notions

We define three security properties: EUF-CMA security, R-identifiability, which means that the verifier traces all attackers within R executions, and correctness, which means that the verifier does not trace legitimate users.

4.2.1 EUF-CMA Security

We define EUF-CMA security for an ASIT scheme. A potential adversary in this security notion includes all signers apart from an honest one and an aggregator. Note that we regard the aggregator as a potential adversary as in the same manner as in ordinary aggregate signature schemes.

Definition 4.4: An ASIT scheme Σ_{ASIT} satisfies EUF-CMA security if for any $\lambda \in \mathbb{N}$, any $n = \mathrm{poly}(\lambda)$, and any PPT adversary $\mathcal{A}$, it holds that $\Pr[\mathsf{ExpASIT}^{\mathrm{EUF\text{-}CMA}}_{\Sigma_{\mathrm{ASIT}},\mathcal{A}}(\lambda, n) = 1] = \mathrm{negl}(\lambda)$ where $\mathsf{ExpASIT}^{\mathrm{EUF\text{-}CMA}}_{\Sigma_{\mathrm{ASIT}},\mathcal{A}}$ is the following experiment.

$\mathsf{ExpASIT}^{\mathrm{EUF\text{-}CMA}}_{\Sigma_{\mathrm{ASIT}},\mathcal{A}}(\lambda, n)$
$\forall i \in [n], (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \Sigma_{\mathrm{ASIT}}.\mathsf{KeyGen}(1^\lambda);$
$t \coloneqq 1; Q \coloneqq \emptyset; W_1 \coloneqq \emptyset;$
run $\mathcal{A}^{O_S(\cdot), O_V(\cdot)}(\mathsf{pk}_1, \{(\mathsf{pk}_i, \mathsf{sk}_i)\}_{i\in[n]\setminus\{1\}})$:
Output 0 when $\mathcal{A}$ halts

where $\mathcal{A}$ can halt at an arbitrary point, $\mathcal{A}$ is allowed to make arbitrarily polynomial many queries to the signing oracle O_S and the verification oracle O_V, which work as follows:

O_S**:** Given a query $m \in \mathcal{M}$, O_S runs $\sigma \leftarrow \Sigma_{\mathrm{ASIT}}.\mathsf{Sign}(\mathsf{sk}_1, m)$, returns σ to $\mathcal{A}$, and updates $Q \leftarrow Q \cup \{m\}$.

O_V**:** Given a set of pairs of an index and a message and an aggregate signature $(\{(i, m_{i,t})\}_{i\in I_t}, \tau_t)$, O_V outputs 1 (indicating that $\mathcal{A}$ wins) and terminates the experiment if $\Sigma_{\mathrm{ASIT}}.\mathsf{PartVerify}(\beta_t, \{(\mathsf{pk}_i, m_{i,t})\}_{i\in I_t}, \tau_t, 1) = 1$, $1 \notin W_t$, and $m_{1,t} \notin Q$. Otherwise, O_V executes $(\beta_{t+1}, f_t, V_t) \leftarrow \Sigma_{\mathrm{ASIT}}.\mathsf{Trace}(\beta_t, \{(\mathsf{pk}_i, m_{i,t})\}_{i\in I_t}, \tau_t)$, returns (f_t, V_t) to $\mathcal{A}$, and updates $W_t = W_t \cup V_t$ and $t \leftarrow t + 1$.

Note that the user index 1 is treated as the challenge user, and an adversary is given the secret keys for the remaining

users with index 2 to n, and thus the signing oracle is necessary only for the user index 1. Note also that the experiment can output 1 only if $\mathcal{A}$ makes an O_V-query that contains a forged signature with respect to the user index 1 (judged using the algorithm PartVerify).

PartVerify is introduced for readability, and it is indeed possible to define EUF-CMA security without PartVerify. However, without PartVerify, representing the definition of EUF-CMA security needs a combination of the verification algorithm of the underlying aggregate signature scheme and the structure of DTT, which is too complicated.

4.2.2 R-Identifiability and Correctness

Similar to DTT, we define R-identifiability and correctness of Σ_{ASIT}. A potential adversary in these security notions is a set of users $C \subseteq [n]$ that may generate invalid signatures. Thus, an aggregator and the verifier behave honestly. These security notions are defined based on the following experiment $\mathsf{ExpASIT}_{\Sigma_{\mathrm{ASIT}},\mathcal{A}}(\lambda, n)$ in which a stateful adversary $\mathcal{A}$ is executed:

$$\frac{\mathsf{ExpASIT}_{\Sigma_{\mathrm{ASIT}},\mathcal{A}}(\lambda, n)}{\begin{array}{c}\forall i \in [n], (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \Sigma_{\mathrm{ASIT}}.\mathsf{KeyGen}(1^\lambda);\\ C \leftarrow \mathcal{A}(\{(\mathsf{pk}_i, \mathsf{sk}_i)\}_{i\in[n]}); t := 1; r := 0; W_1 := \emptyset;\\ \beta_1 := \epsilon; f_0 := \epsilon; I_1 := [n]; J_1 := C;\\ \text{run } \mathcal{A}^{O_T(\cdot)}(\{(\mathsf{pk}_i, \mathsf{sk}_i)\}_{i\in[n]}) :\\ \text{Output } (W := \bigcup_{t'=1}^{t} V_{t'}, C, r) \text{ when } \mathcal{A} \text{ halts}\end{array}}$$

where $\mathcal{A}$ can halt at an arbitrary point, and $\mathcal{A}$ is allowed to make arbitrarily (polynomially) many queries to the tracing oracle O_T. Let $W_t := \bigcup_{t'=1}^{t} V_t$, $I_t := [n]\backslash W_t$, and $J_t := C \setminus W_t$. Given a query $(\{m_{i,t}\}_{i\in I_t}, \{(m_{j,t}, \sigma_{j,t})\}_{j\in J_t})$ from $\mathcal{A}$, O_T operates as follows:

1. If there exists $j \in J_t$ s.t. $\Sigma_{\mathrm{ASIT}}.\mathsf{Verify}(\mathsf{pk}_j, m_{j,t}, \sigma_{j,t}) = 0$, then set $r \leftarrow r + 1$.
2. For every $i \in I_t$, compute $\sigma_{i,t} \leftarrow \Sigma_{\mathrm{ASIT}}.\mathsf{Sign}(\mathsf{sk}_i, m_{i,t})$.
3. Compute $\tau_t \leftarrow \Sigma_{\mathrm{ASIT}}.\mathsf{Agg}(f_{t-1}, \{(\mathsf{pk}_i, m_{i,t}, \sigma_{i,t})\}_{i\in I_t\cup J_t})$.
4. Compute $(\beta_{t+1}, f_t, V_t) \leftarrow \Sigma_{\mathrm{ASIT}}.\mathsf{Trace}(\beta_t, \{(\mathsf{pk}_i, m_{i,t})\}_{i\in I_t\cup J_t}, \tau_t)$.
5. Return (f_t, V_t) to $\mathcal{A}$ and set $t \leftarrow t + 1$.

We define R-identifiability and correctness of an ASIT scheme as follows.

Definition 4.5 (R-Identifiability): An ASIT scheme Σ_{ASIT} satisfies R-identifiability if for any $\lambda \in \mathbb{N}$, any $n = \mathrm{poly}(\lambda)$, and any PPT adversary $\mathcal{A}$, we have

$$\Pr[(C \not\subseteq W) \,|\, (W, C, r) \leftarrow \mathsf{ExpASIT}_{\Sigma_{\mathrm{ASIT}},\mathcal{A}}(\lambda, n) \wedge (r \geq R)] = \mathrm{negl}(\lambda).$$

Definition 4.6 (Correctness): An ASIT scheme Σ_{ASIT} satisfies correctness if $\Sigma_{\mathrm{ASIT}}.\mathsf{Trace}$ and $\Sigma_{\mathrm{ASIT}}.\mathsf{PartVerify}$ satisfy correctnesses respectively, and for any $\lambda \in \mathbb{N}$, any $n = \mathrm{poly}(\lambda)$, and any PPT adversary $\mathcal{A}$, we have

$$\Pr[([n]\backslash C) \cap W \neq \emptyset \,|\, (W, C, r) \leftarrow \mathsf{ExpASIT}_{\Sigma_{\mathrm{ASIT}},\mathcal{A}}(\lambda, n)] = \mathrm{negl}(\lambda).$$

5. A Generic Construction of an ASIT from an Aggregate Signature Scheme and a DTT

In this section, we first present our generic construction of an ASIT scheme based on an aggregate signature scheme and a DTT, along with its security proofs and theoretical comparison with the existing schemes. Finally, we demonstrate an experimental result for the proposed scheme.

5.1 Construction

Let $\Sigma_{\mathrm{AS}} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Agg}, \mathsf{AggVerify})$ be an aggregate signature scheme, and let $\Sigma_{\mathrm{DTT}} = (\mathsf{Initialize}, \mathsf{Trace})$ be a DTT. Using Σ_{AS} and Σ_{DTT} as building blocks, we construct an ASIT scheme $\Sigma_{\mathrm{ASIT}} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Agg}, \mathsf{PartVerify}, \mathsf{Trace})$ as described in Fig. 3.

A verifier in Σ_{ASIT} maintains an internal state and a user partition of Σ_{DTT}. The algorithms $\Sigma_{\mathrm{ASIT}}.\mathsf{KeyGen}$, $\Sigma_{\mathrm{ASIT}}.\mathsf{Sign}$ and $\Sigma_{\mathrm{ASIT}}.\mathsf{Verify}$ are the same as those in Σ_{AS}. The verifier initially executes $\Sigma_{\mathrm{DTT}}.\mathsf{Initialize}$, and in subsequent rounds the aggregator execute $\Sigma_{\mathrm{AS}}.\mathsf{Agg}$ based on the verifier's feedback from the previous round. The algorithm $\Sigma_{\mathrm{ASIT}}.\mathsf{Agg}$ generates aggregate signatures for each subset of the user partition generated by the use of Σ_{DTT}. When $\Sigma_{\mathrm{ASIT}}.\mathsf{Trace}$ finds that an invalid signature is contained in τ, Σ_{ASIT} uses the index of the corresponding subset in the user partition to run $\Sigma_{\mathrm{DTT}}.\mathsf{Trace}$ (and does not verify the rest of the aggregate signatures).

Theorem 5.1 (Correctness of $\Sigma_{\mathrm{ASIT}}.\mathsf{Trace}$): If Σ_{AS} satisfies correctness, then $\Sigma_{\mathrm{ASIT}}.\mathsf{Trace}$ also satisfies correctness.

Proof 5.1: Observe that an output of $\Sigma_{\mathrm{ASIT}}.\mathsf{Agg}$ consists of multiple (polynomially many) aggregate signatures of the underlying scheme $\Sigma_{\mathrm{AS}}.\mathsf{Agg}$. Furthermore, $\Sigma_{\mathrm{ASIT}}.\mathsf{Trace}$ outputs a traced user set $V \neq \emptyset$ if and only if $\Sigma_{\mathrm{AS}}.\mathsf{AggVerify}$ outputs 0. Since each algorithm in $\mathsf{ExpASIT}^{\mathrm{Trace}}_{\Sigma_{\mathrm{ASIT}}}$ is executed honestly and Σ_{AS} is correct, the probability that $\mathsf{ExpASIT}^{\mathrm{Trace}}_{\Sigma_{\mathrm{ASIT}}}$ outputs $V \neq \emptyset$ is 0, which guarantees the correctness of $\Sigma_{\mathrm{ASIT}}.\mathsf{Trace}$. □

Theorem 5.2 (Correctness of $\Sigma_{\mathrm{ASIT}}.\mathsf{PartVerify}$): If Σ_{AS} satisfies correctness, then $\Sigma_{\mathrm{ASIT}}.\mathsf{PartVerify}$ also satisfies correctness.

Proof 5.2: Observe that an output of $\Sigma_{\mathrm{ASIT}}.\mathsf{Agg}$ consists of multiple (polynomially many) aggregate signatures of the underlying scheme $\Sigma_{\mathrm{AS}}.\mathsf{Agg}$, and an output of $\Sigma_{\mathrm{ASIT}}.\mathsf{PartVerify}$ is that of $\Sigma_{\mathrm{AS}}.\mathsf{AggVerify}$. Since each algorithm in $\mathsf{ExpASIT}^{\mathrm{PartVrf}}_{\Sigma_{\mathrm{ASIT}}}$ is executed honestly and Σ_{AS} is correct, $\mathsf{ExpASIT}^{\mathrm{PartVrf}}_{\Sigma_{\mathrm{ASIT}}}$ outputs 1 with probability 1, which guarantees the correctness of $\Sigma_{\mathrm{ASIT}}.\mathsf{PartVerify}$. □

$\mathsf{KeyGen}(1^\lambda)$:
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \Sigma_{\mathrm{AS}}.\mathsf{KeyGen}(1^\lambda)$
return $(\mathsf{pk}, \mathsf{sk})$

$\mathsf{Sign}(\mathsf{sk}, m)$:
$\sigma \leftarrow \Sigma_{\mathrm{AS}}.\mathsf{Sign}(\mathsf{sk}, m)$
return σ

$\mathsf{Verify}(\mathsf{pk}, m, \sigma)$:
$v \leftarrow \Sigma_{\mathrm{AS}}.\mathsf{Verify}(\mathsf{pk}, m, \sigma)$
return v

$\mathsf{Agg}(f, \{(\mathsf{pk}_i, m_i, \sigma_i)\}_{i\in[n]})$:
if $f = \epsilon$ **then**
$(\alpha, P) \leftarrow \Sigma_{\mathrm{DTT}}.\mathsf{Initialize}(1^\lambda, 1^n)$
else $P := f$
parse $(S_i)_{i\in[|P|]} \leftarrow P$
for $j := 1$ to $|P|$ **do**
$\tau_j \leftarrow \Sigma_{\mathrm{AS}}.\mathsf{Agg}(\{(\mathsf{pk}_i, m_i, \sigma_i)\}_{i\in S_j})$
return $\tau := (\tau_j)_{j\in[|P|]}$

$\mathsf{PartVerify}(\beta, \{(\mathsf{pk}_j, m_j)\}_{j\in[n]}, \tau, i)$:
if $\beta = \epsilon$ **then**
$(\alpha, P) \leftarrow \Sigma_{\mathrm{DTT}}.\mathsf{Initialize}(1^\lambda, 1^n)$
else parse $(\alpha, P) \leftarrow \beta$
parse $(S_i)_{i\in[|P|]} \leftarrow P$ and $(\tau_i)_{i\in[|\tau|]} \leftarrow \tau$
find i' s.t. $i \in S_{i'}$
$v \leftarrow \Sigma_{\mathrm{AS}}.\mathsf{AggVerify}(\{(\mathsf{pk}_j, m_j)\}_{j\in S_{i'}}, \tau_{i'})$
return v

$\mathsf{Trace}(\beta, \{(\mathsf{pk}_i, m_i)\}_{i\in[n]}, \tau)$:
if $\beta = \epsilon$ **then**
$(\alpha, P) \leftarrow \Sigma_{\mathrm{DTT}}.\mathsf{Initialize}(1^\lambda, 1^n)$
else parse $(\alpha, P) \leftarrow \beta$
parse $(S_i)_{i\in[|P|]} \leftarrow P$ and $(\tau_i)_{i\in[|\tau|]} \leftarrow \tau$
$f' := P$; $\beta' := (\alpha, P)$; $V := \emptyset$
for $i := 1$ to $|\tau|$ **do**
$v \leftarrow \Sigma_{\mathrm{AS}}.\mathsf{AggVerify}(\{(\mathsf{pk}_j, m_j)\}_{j\in S_i}, \tau_i)$
if $v = 0$ **then**
$(\alpha', P', V) \leftarrow \Sigma_{\mathrm{DTT}}.\mathsf{Trace}(\alpha, i)$; $f' := P'$; $\beta' := (\alpha', P')$
break (†)
return (β', f', V)

Fig. 3 Our generic construction of an ASIT scheme Σ_{ASIT} from an aggregate signature scheme Σ_{AS} and a DTT Σ_{DTT}. (†) End the **for** loop and proceed to the next line.

Discussion on The Construction

In Fig. 3, $\Sigma_{\mathrm{ASIT}}.\mathsf{Trace}$ executes DTT to generate new partitions, and sends them to $\Sigma_{\mathrm{ASIT}}.\mathsf{Agg}$. This aims to capture the situation where a verifier has a plenty of computational resource, but an aggregator does not. For instance, in IoT, it may be the case that a cloud server runs $\Sigma_{\mathrm{ASIT}}.\mathsf{Trace}$ and an edge device runs $\Sigma_{\mathrm{ASIT}}.\mathsf{Agg}$. However, if we allow an aggregator to have sufficient computational resource (and assume that the aggregator is honest), we might be able to reduce network bandwidth. That is, we let $\Sigma_{\mathrm{ASIT}}.\mathsf{Agg}$ to run DTT by itself. Observe that DTT is just a deterministic combinatorial algorithm. Hence, if $\Sigma_{\mathrm{ASIT}}.\mathsf{Trace}$ sends information about in which position to execute DTT to $\Sigma_{\mathrm{ASIT}}.\mathsf{Agg}$, then the both algorithms can obtain the same partitions. Namely, since $\Sigma_{\mathrm{ASIT}}.\mathsf{Trace}$ is not necessary to send partitions, we can reduce network bandwidth.

We note that the number of rounds that is necessary for Σ_{ASIT} to trace all adversaries could depend on the order of the verification. For instance, if $\Sigma_{\mathrm{ASIT}}.\mathsf{Trace}$ always verifies an aggregate signature of an innocent set first (if exists), then the set is kept to be small, which might result in an efficient tracing. However, we leave the description abstract, because for any construction, there might exist a strategic adversary that abuses such a construction.

5.2 Security Proofs

5.2.1 EUF-CMA Security

We first show the EUF-CMA security of Σ_{ASIT}.

Theorem 5.3: If Σ_{AS} satisfies EUF-CMA security, then Σ_{ASIT} satisfies EUF-CMA security.

Proof 5.3: We assume for contradiction that there exists a PPT adversary $\mathcal{A}_{\mathrm{ASIT}}$ that breaks EUF-CMA security of Σ_{ASIT} with non-negligible probability, and construct a PPT adversary $\mathcal{A}_{\mathrm{AS}}$ that breaks EUF-CMA security of Σ_{AS}. We first describe how our adversary $\mathcal{A}_{\mathrm{AS}}$ works in the experiment $\mathsf{ExpAS}^{\text{EUF-CMA}}_{\Sigma_{\mathrm{AS}},\mathcal{A}_{\mathrm{AS}}}$, and then show that $\mathcal{A}_{\mathrm{AS}}$ indeed breaks EUF-CMA security of Σ_{AS}.

In $\mathsf{ExpAS}^{\text{EUF-CMA}}_{\Sigma_{\mathrm{AS}},\mathcal{A}_{\mathrm{AS}}}$, the challenger generates $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \Sigma_{\mathrm{AS}}.\mathsf{KeyGen}(\lambda)$ for all $i \in [n]$, gives pk_1 and $\{(\mathsf{pk}_i, \mathsf{sk}_i)\}_{i\in[n]\setminus\{1\}}$ to $\mathcal{A}_{\mathrm{AS}}$, and sets $Q := \emptyset$. Next, $\mathcal{A}_{\mathrm{AS}}$ sets $W_1 := \emptyset$. Then, $\mathcal{A}_{\mathrm{AS}}$, who is given access to the signing oracle $\Sigma_{\mathrm{AS}}.\mathsf{Sign}(\mathsf{sk}_1, \cdot)$, runs $\mathcal{A}^{O_S,O_V}_{\mathrm{ASIT}}(\mathsf{pk}_1, \{(\mathsf{pk}_i, \mathsf{sk}_i)\}_{i\in[n]\setminus\{1\}})$ with simulating O_S and O_V as follows.

O_S: When $\mathcal{A}_{\mathrm{ASIT}}$ makes an O_S-query m, $\mathcal{A}_{\mathrm{AS}}$ makes a signing query m to its own oracle, receives σ from the signing oracle, and returns σ to $\mathcal{A}_{\mathrm{ASIT}}$. $\mathcal{A}_{\mathrm{AS}}$ also updates $Q \leftarrow Q \cup \{m\}$.

O_V: When $\mathcal{A}_{\mathrm{ASIT}}$ makes a query $(\{(i, m_{i,t})\}_{i\in I_t}, \tau_t)$ to O_V, $\mathcal{A}_{\mathrm{AS}}$ responds as follows (where we implicitly assume that $I_t \subseteq [n]$):

- If $\Sigma_{\mathrm{ASIT}}.\mathsf{PartVerify}(\beta_t, \{(\mathsf{pk}_i, m_{i,t})\}_{i\in I_t}, \tau_t, 1) = 1$, $1 \notin W_t$, and $m_{1,t} \notin Q$, then halt the simulation, output τ_t, $\{m_{i,t}\}_{i\in I_t}$ and $S := I_t$ to $\mathcal{A}_{\mathrm{AS}}$'s EUF-CMA experiment, and terminate.
- Otherwise, execute $(\beta_{t+1}, f_t, V_t) \leftarrow \Sigma_{\mathrm{ASIT}}.\mathsf{Trace}(\beta_t, \{(\mathsf{pk}_i, m_{i,t})\}_{i\in I_t}, \tau_t)$, return (f_t, V_t) to $\mathcal{A}_{\mathrm{ASIT}}$, and set $W_{t+1} = W_t \cup V_t$ and $t \leftarrow t + 1$.

When $\mathcal{A}_{\mathrm{ASIT}}$ halts, $\mathcal{A}_{\mathrm{AS}}$ also halts. (Note that $\mathcal{A}_{\mathrm{AS}}$ perfectly simulates the EUF-CMA experiment for $\mathcal{A}_{\mathrm{ASIT}}$.)

We argue that $\mathcal{A}_{\mathrm{AS}}$ satisfies the winning condition in the EUF-CMA experiment with non-negligible probability. Suppose that $\mathcal{A}_{\mathrm{AS}}$ outputs τ_t, $\{m_{i,t}\}_{i\in I_t}$ and S (i.e., $\mathcal{A}_{\mathrm{ASIT}}$ makes a query to O_V on $(\{(i, m_{i,t})\}_{i\in I_t}, \tau_t)$ s.t. $\Sigma_{\mathrm{ASIT}}.\mathsf{PartVerify}(\beta_t, \{(\mathsf{pk}_i, m_{i,t})\}_{i\in I_t}, \tau_t, 1) = 1$, $1 \notin W_t$ and $m_{1,t} \notin Q$). As $\Sigma_{\mathrm{ASIT}}.\mathsf{PartVerify}(\beta_t, \{(\mathsf{pk}_i, m_{i,t})\}_{i\in I_t}, \tau_t, 1) = 1$, it

holds that $1 \in S \subseteq [n]$. Thus, $\mathcal{A}_{\mathrm{AS}}$ wins in this case. Observe that if $\mathcal{A}_{\mathrm{ASIT}}$ makes such a query, it means that $\mathcal{A}_{\mathrm{ASIT}}$ breaks the EUF-CMA security of Σ_{ASIT}. Hence, if $\mathcal{A}_{\mathrm{ASIT}}$ breaks the EUF-CMA security of Σ_{ASIT} with non-negligible probability, then $\mathcal{A}_{\mathrm{AS}}$ also breaks the EUF-CMA security of Σ_{AS} with non-negligible probability. □

5.2.2 R-Identifiability and Correctness

Here, we show R-identifiability and correctness of Σ_{ASIT}. Note that the correctness of $\Sigma_{\mathrm{ASIT}}.\mathsf{Trace}$ and $\Sigma_{\mathrm{ASIT}}.\mathsf{PartVerify}$ are already done.

Theorem 5.4: If Σ_{DTT} satisfies R-identifiability, then Σ_{ASIT} satisfies R-identifiability.

Proof 5.4: To prove the theorem, we demonstrate security reductions from Σ_{ASIT} to Σ_{DTT}. In what follows, we describe the adversary $\mathcal{A}_{\mathrm{DTT}}$ in $\mathsf{ExpDTT}_{\Sigma_{\mathrm{DTT}},\mathcal{A}_{\mathrm{DTT}}}$ that simulates the experiment $\mathsf{ExpASIT}^{\mathrm{Trace}}_{\Sigma_{\mathrm{ASIT}},\mathcal{A}_{\mathrm{ASIT}}}$ for an adversary $\mathcal{A}_{\mathrm{ASIT}}$.

In $\mathsf{ExpDTT}_{\Sigma_{\mathrm{DTT}},\mathcal{A}_{\mathrm{DTT}}}$, the challenger computes $(\alpha_1, P_1) \leftarrow \Sigma_{\mathrm{DTT}}.\mathsf{Initialize}(1^\lambda, 1^n)$, and gives (α_1, P_1) to $\mathcal{A}_{\mathrm{DTT}}$. $\mathcal{A}_{\mathrm{DTT}}$ first runs $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \Sigma_{\mathrm{ASIT}}.\mathsf{KeyGen}(1^\lambda)$ for $i \in [n]$, and then gives $\{(\mathsf{pk}_i, \mathsf{sk}_i)\}_{i\in[n]}$ to $\mathcal{A}_{\mathrm{ASIT}}$, which in turn outputs the set of users $C \subseteq [n]$. Then, $\mathcal{A}_{\mathrm{DTT}}$ outputs C to $\mathcal{A}_{\mathrm{DTT}}$'s experiment as a traitor set. $\mathcal{A}_{\mathrm{DTT}}$ then sets $t' \coloneqq 1$, $r \coloneqq 0$, $W_1 \coloneqq \emptyset$, $\beta_1 \coloneqq \epsilon$, $f_0 \coloneqq \epsilon$, $I_1 \coloneqq [n]$, and $J_1 \coloneqq C$, and continues the execution of $\mathcal{A}_{\mathrm{ASIT}}$ that now may start making O_T queries. When $\mathcal{A}_{\mathrm{ASIT}}$ makes an O_T-query $(\{m_i\}_{i\in I_t}, \{(m_{j,t}, \sigma_{j,t})\}_{j\in J_t})$, $\mathcal{A}_{\mathrm{DTT}}$ proceed as follows:

1. If there exists $j \in J_t$ s.t. $\Sigma_{\mathrm{ASIT}}.\mathsf{Verify}(\mathsf{pk}_j, m_{j,t}, \sigma_{j,t}) = 0$, then $r \leftarrow r + 1$.
2. For every $i \in I_t$, compute $\sigma_{i,t} \leftarrow \Sigma_{\mathrm{ASIT}}.\mathsf{Sign}(\mathsf{sk}_i, m_{i,t})$.
3. Compute $\tau_t \leftarrow \Sigma_{\mathrm{ASIT}}.\mathsf{Agg}(f_{t-1}, \{(\mathsf{pk}_i, m_{i,t}, \sigma_{i,t})\}_{i\in I_t\cup J_t})$.
4. Simulate $(\beta_{t+1}, f_t, V_t) \leftarrow \Sigma_{\mathrm{ASIT}}.\mathsf{Trace}(\beta_t, \{(\mathsf{pk}_i, m_{i,t})\}_{i\in I_t\cup J_t}, \tau_t)$ s.t. when $\Sigma_{\mathrm{DTT}}.\mathsf{Trace}(\alpha_t, i)$ needs to be executed, $\mathcal{A}_{\mathrm{DTT}}$ makes a query i to $\mathcal{A}_{\mathrm{DTT}}$'s own tracing oracle O_T to obtain $(\alpha_{t+1}, P_{t+1}, V_t)$ instead of running $\Sigma_{\mathrm{DTT}}.\mathsf{Trace}(\alpha_t, i)$ by itself. Note that in our construction Σ_{ASIT}, the update of the internal state $\beta = (\alpha, P)$ occurs only when $\Sigma_{\mathrm{DTT}}.\mathsf{Trace}(\alpha, i)$ is executed, and thus the internal state used by $\mathcal{A}_{\mathrm{DTT}}$ to simulate the tracing oracle for $\mathcal{A}_{\mathrm{ASIT}}$ and the internal state used in $\mathcal{A}_{\mathrm{DTT}}$'s tracing oracle are always synchronized.
5. Return (f_t, V_t) to $\mathcal{A}_{\mathrm{ASIT}}$, and set $W_{t+1} \leftarrow W_t \cup V_t$, $I_{t+1} \leftarrow I_t \setminus W_t$, $J_{t+1} \leftarrow C \setminus W_t$, and $t \leftarrow t + 1$.

When $\mathcal{A}_{\mathrm{ASIT}}$ halts, $\mathcal{A}_{\mathrm{DTT}}$ also halts. (Note that $\mathcal{A}_{\mathrm{DTT}}$ correctly simulates the tracing oracle O_T for $\mathcal{A}_{\mathrm{ASIT}}$. Furthermore, both $\mathcal{A}_{\mathrm{DTT}}$ and $\mathcal{A}_{\mathrm{ASIT}}$ run on the same traitor set C.)

Observe that $\mathcal{A}_{\mathrm{DTT}}$ makes a query to O_T if and only if there exists $j \in J_t$ s.t. $\Sigma_{\mathrm{ASIT}}.\mathsf{Verify}(\mathsf{pk}_j, m_{j,t}, \sigma_{j,t}) = 0$, due to the design of $\Sigma_{\mathrm{ASIT}}.\mathsf{Trace}$. Thus, letting t' denote the counter denoting the number of O_T queries in $\mathcal{A}_{\mathrm{DTT}}$'s R-identifiability experiment, it holds that $t' = r$ when $\mathcal{A}_{\mathrm{DTT}}$ halts the simulation of $\mathcal{A}_{\mathrm{ASIT}}$'s R-identifiability experiment. Obviously, if $\mathcal{A}_{\mathrm{ASIT}}$ results in a set $W \notin C$, then so does $\mathcal{A}_{\mathrm{DTT}}$. Therefore, if there exists a PPT adversary $\mathcal{A}_{\mathrm{ASIT}}$ that breaks the R-identifiability of Σ_{ASIT}, then there exists a PPT adversary $\mathcal{A}_{\mathrm{DTT}}$ that violates the R-identifiability of Σ_{DTT}. □

Theorem 5.5: If Σ_{DTT} satisfies completeness, then Σ_{ASIT} satisfies correctness.

Proof 5.5: (of Theorem 5.5) Given an adversary $\mathcal{A}_{\mathrm{ASIT}}$ attacking the correctness of Σ_{ASIT}, we consider exactly the same reduction algorithm $\mathcal{A}_{\mathrm{DTT}}$ as in the proof of Theorem 5.4. Obviously, if $\mathcal{A}_{\mathrm{ASIT}}$ results in a set W_t s.t. $([n] \setminus C) \cap W_t \neq \emptyset$, then so does $\mathcal{A}_{\mathrm{DTT}}$. Therefore, the existence of such $\mathcal{A}_{\mathrm{ASIT}}$ means the existence of a PPT adversary $\mathcal{A}_{\mathrm{DTT}}$ that violates the completeness of Σ_{DTT}. □

5.3 Theoretical Comparison

We theoretically compare our schemes with the existing schemes by Hartung et al. [6] which we denote by HKK^+, and by Sato et al. [15]. In [6], they proposed a so-called a fault-tolerant signature scheme, which can be understood as an ASIT scheme with a one-round tracing algorithm. However, since this scheme can be trivially extended to a multi-round setting, for the comparison in this section, we use such this extension which we hereafter refer to as $\mathsf{multi\text{-}HKK}^+$. For completeness, we give the description of $\mathsf{multi\text{-}HKK}^+$ in Appendix Appendix. We remark that HKK^+ is able to detect at most d invalid signatures in a single round, where d is a fixed value hard-wired in the scheme. However, the correctness of HKK^+ is compromised if more than d invalid signatures are involved. $\mathsf{multi\text{-}HKK}^+$ inherits this restricted form of correctness. AS-FT-2 and AS-FT-3 denote the ASIT schemes obtained from our generic construction by instantiating the underlying DTT with FT-2 and FT-3, respectively.

Table 1 shows the comparison among these schemes in terms of the number of communication rounds, the maximum number of transmitted signatures per round, and the types of traceability that each scheme achieves: An ASIT scheme is *statically traceable* if it satisfies R-identifiability for some R and correctness in a setting where the adversary generates invalid signatures for all users in C (the adversarial set of users) in every round. In contrast, an ASIT scheme is *dynamically traceable* if it satisfies both of the security requirements as defined in Sect. 4.2 where the adversary may adaptively decide which user in C generates an invalid signature in each round. In Table 1, Rounds (R) is the maximum number of rounds that is necessary to trace all users providing invalid signatures, and Sigs (max) is the maximum number of signatures that are aggregated (and thus corresponds to the partition of the user set) in a single round.

Table 1 Comparison among existing schemes and proposed schemes. Each value represents the following: n is the number of users, and d is the number of users providing invalid signatures.

Scheme	Rounds (R)	Sigs (max)	Stat	Dyn
multi-HKK$^+$	$2d+1$	$2d+1$ (fixed)	✓	?
Sato et al. [15]	$d\log_2 n + d$	$2d+1$	✓	?
AS-FT-2	$d\log_2 n + d$	$2d+1$	✓	✓
AS-FT-3	$2\cdot 3^d d\log_2 n + d$	$d+1$	✓	✓

Stat and Dyn represent statically traceable and dynamically traceable, respectively.

As indicated in Table 1, multi-HKK$^+$ and the scheme in [15] are statically traceable, but it is unclear if they are dynamically traceable. The maximum numbers of signatures per round of these two schemes are the same ($2d+1$), but it is constantly $2d+1$ in every round in multi-HKK$^+$, while it can vary from 1 to $2d+1$ as the tracing proceeds in [15]. Comparing [15] and AS-FT-2, the number of rounds and the number of signatures per round are the same, but AS-FT-2 is traceable for both static and dynamic adversaries, while the dynamic traceability of [15] is unclear.

Our schemes are able to trace both dynamic and static adversaries, and the number of rounds required for tracing does not increase compared to the existing method. We remark that the number of rounds and signatures of AS-FT-2 and AS-FT-3 are due to Theorems 3.1 and 3.2, respectively. Therefore, our proposed schemes are more suitable for practical usage scenario (e.g. sensor networks) for all situations where invalid signatures could be generated constantly by dynamically determined users/devices.

5.4 Experimental Result

We now provide an experimental result for out scheme; namely we implement AS-FT-2 and compare AS-FT-2 against an adaptive adversary and a static adversary. We choose the BGLS signature [4] as the underlying aggregate signature scheme, and implement AS-FT-2 by C++ with using the mcl library for pairing-based cryptography by Mitsunari[†]. (Note that any aggregate signature scheme will do for our construction.) We implement the tracing algorithm so that it always verifies an aggregate signature of an innocent set first, if exists. Table 2 illustrates our experimental environment.

We compare our implementation in dynamic and static settings. (We emphasize that our scheme is the first fault-tolerant aggregate signature scheme that can efficiently deal with adaptive adversary.) We fix the number of signers as $N = 1000$, and set the number of adversaries as $d = 10$ or $d = 100$ for each experiment, where adversaries are randomly chosen from the signers. In the dynamic setting, in each round, exactly one adversary is randomly chosen and only the chosen one generates an invalid signature, whereas in the static setting, all adversaries send invalid signatures every round.

In Table 3, Adv stands for the types of adversaries, d stands for the number of adversaries, R stands for the number of rounds needed to trace all adversaries, Sigs stands for the number of aggregate signatures that are created during the entire tracing process, Partitions stands for the number of partitions that are generated during the entire tracing process, and Time [s] stands for the running time.

[†]https://github.com/herumi/mcl.

Table 2 Experimental environment.

Host OS	Windows10 Pro
Virtual OS	ubuntu 18.04 LTS
CPU	Intel Core i7-8550U @1.80GHz 4 cores 8 threads
Memory	16GB
c++	g++ 9.3.0
Elliptic curve	BN254

Table 3 Experimental result for dynamic and static settings. Each value is the average of one hundred executions.

Adv	d	R	Sigs	Partitions	Time [s]
dynamic	10	101.2	1210.0	1210.0	33.7
	100	917.7	52515.7	52515.7	240.4
static	10	90.7	709.9	709.9	21.6
	100	614.2	24976.7	24976.7	94.4

It takes more rounds (and thus, more aggregate signatures and partitions) to trace all adversaries in the dynamic setting than in the static setting. This is because only a single randomly chosen adversary sends an invalid signature in the dynamic setting, while every adversary sends an invalid signature in every round in the static setting. Note that the number of aggregate signatures is equal to the number of partitions, because aggregate signatures are generated based on partitions.

Recall that, as shown in Table 1, AS-FT-2 identifies all adversaries within at most $d\log_2 n + d$ rounds. Observe that our result satisfies this constraint; $101.2 < 10\log_2 1000 + 10 \approx 110$ for $d = 10$, and $917.7 < 100\log_2 1000 + 100 \approx 1097$ for $d = 100$. Hence, we argue that the implementation efficiently cope with adaptive adversaries.

6. Conclusion and Future Work

We have proposed a new fault-tolerant aggregate signature scheme [6], an Aggregate Signature scheme with Interactive Tracing functionality (ASIT scheme for short), that can efficiently deal with adversaries who behaves adaptively. We have provided a generic construction of an ASIT scheme, defined and proved its security, and implemented it to confirm that it indeed copes with adaptive adversary.

However, there is still room for consideration. Recall that we only conducted preliminary experiments for our scheme. Hence, we should use it in a concrete applications such as sensor networks, which requires to settle network architecture, message format for aggregate signature schemes and so on.

Regarding the problem setting, we restricted ourself to consider the situation where at least one adversary sends an invalid signature every round. A more practical situation is that we admit all adversaries not to send invalid signatures (in other words, we accept a round that every individual signature is valid). Such a setting might capture, for instance, a

sensor network where there is no malicious signer but sometimes a signer misbehaves probabilistically. Hence, we leave it as an open problem to consider a fault-tolerant aggregate signature scheme that can efficiently deal with such a situation.

References

[1] R. Ishii, K. Yamashita, Y. Sakai, T. Matsuda, T. Teruya, G. Hanaoka, K. Matsuura, and T. Matsumoto, "Aggregate signature with traceability of devices dynamically generating invalid signatures," ACNS 2021 Satellite Workshops, AIBlock, AIHWS, AIoTS, CIMSS, Cloud S&P, SCI, SecMT, and SiMLA, Proceedings, LNCS, vol.12809, pp.378–396, Springer, 2021.
[2] J.H. Ahn, M. Green, and S. Hohenberger, "Synchronized aggregate signatures: new definitions, constructions and applications," CCS 2010, pp.473–484, ACM, Oct. 2010.
[3] A. Makarov, "A survey of aggregate signature applications," Advanced Technologies in Robotics and Intelligent Systems, pp.309–317, Springer, 2020.
[4] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, "Aggregate and verifiably encrypted signatures from bilinear maps," EUROCRYPT 2003, LNCS, vol.2656, pp.416–432, Springer, 2003.
[5] A. Fiat and T. Tassa, "Dynamic traitor tracing," CRYPTO '99, pp.354–371, Springer, 1999.
[6] G. Hartung, B. Kaidel, A. Koch, J. Koch, and A. Rupp, "Fault-tolerant aggregate signatures," PKC 2016, LNCS, vol.9614, pp.331–356, Springer, 2016.
[7] S. Hohenberger, A. Sahai, and B. Waters, "Full domain hash from (leveled) multilinear maps and identity-based aggregate signatures," CRYPTO 2013, LNCS, vol.8042, pp.494–512, Springer, 2013.
[8] A. Lysyanskaya, S. Micali, L. Reyzin, and H. Shacham, "Sequential aggregate signatures from trapdoor permutations," EUROCRYPT 2004, LNCS, vol.3027, pp.74–90, Springer, 2004.
[9] G. Neven, "Efficient sequential aggregate signed data," EUROCRYPT 2008, ed. N.P. Smart, LNCS, vol.4965, pp.52–69, Springer, 2008.
[10] M. Gerbush, A.B. Lewko, A. O'Neill, and B. Waters, "Dual form signatures: An approach for proving security from static assumptions," ASIACRYPT' 12, LNCS, vol.7658, pp.25–42, Springer, 2012.
[11] S. Lu, R. Ostrovsky, A. Sahai, H. Shacham, and B. Waters, "Sequential aggregate signatures and multisignatures without random oracles," EUROCRYPT 2006, LNCS, vol.4004, pp.465–485, Springer, 2006.
[12] K. Lee, D.H. Lee, and M. Yung, "Sequential aggregate signatures with short public keys without random oracles," Theor. Comput. Sci., vol.579, pp.100–125, May 2015.
[13] C. Gentry and Z. Ramzan, "Identity-based aggregate signatures," PKC 2006, LNCS, vol.3958, pp.257–273, Springer, 2006.
[14] S. Sato and J. Shikata, "Interactive aggregate message authentication scheme with detecting functionality," AINA 2019, Advances in Intelligent Systems and Computing, vol.926, pp.1316–1328, Springer, 2019.
[15] S. Sato, J. Shikata, and T. Matsumoto, "Aggregate signature with detecting functionality from group testing," IACR Cryptol. ePrint Arch., vol.2020, p.1219, 2020.
[16] S. Sato and J. Shikata, "Interactive aggregate message authentication equipped with detecting functionality from adaptive group testing," IACR Cryptol. ePrint Arch., vol.2020, p.1218, 2020.

Appendix: multi-HKK$^+$

Here, we give the description of multi-HKK$^+$ that is constructed based on an ordinary aggregate signature scheme Σ_{AS} and a cover free family. Recall that a d-cover free family (d-CFF) $\mathcal{F} = (\mathcal{S}, \mathcal{B})$ consists of a set $\mathcal{S}$ of m elements and a set $\mathcal{B}$ of n subsets of $\mathcal{S}$, where $d < m < n$, s.t. for any d subsets $B_{i_1}, \ldots, B_{i_d} \in \mathcal{B}$ and for all distinct $B \in \mathcal{B} \backslash \{B_{i_1}, \ldots, B_{i_d}\}$, it holds that $B \notin \bigcup_{j\in[d]} B_{i_j}$.

Let d be an integer s.t. there exists a prime $q = 2d + 1$. Let $\mathcal{F} = (\mathcal{S}, \mathcal{B})$ be a d-CFF based on quadratic polynomials where $\mathcal{S}$ and $\mathcal{B}$ are defined as follows:
$\mathcal{S} := \{(x_i, y_i) : i = 0, \ldots, q^2 - 1\}$, $\mathcal{B} := \{B_{f_0}, \ldots, B_{f_{q^3-1}}\}$ where
$(x_i, y_i) := (\lfloor i/q \rfloor, i \bmod q)$,
$B_{f_j} := \{(0, f_j(0)), \ldots, (q-1, f_j(q-1))\}$,
and $f_j(X) := (\lfloor j/q^2 \rfloor)X^2 + (\lfloor (j \bmod q^2)/q \rfloor)X + (j \bmod q)$.

Figure A·1 describes multi-HKK$^+$ where $T_i = \{j \in \{0, \ldots, q^{k+1} - 1\} \mid f_j(x_i) = y_i\}$ $(i = 0, \ldots, q^2 - 1)$.

$\mathsf{KeyGen}(1^\lambda)$:
  $(\mathsf{pk}, \mathsf{sk}) \leftarrow \Sigma_{\mathrm{AS}}.\mathsf{KeyGen}(1^\lambda)$
  return $(\mathsf{pk}, \mathsf{sk})$

$\mathsf{Sign}(\mathsf{sk}, m)$:
  $\sigma \leftarrow \Sigma_{\mathrm{AS}}.\mathsf{Sign}(\mathsf{sk}, m)$
  return σ

$\mathsf{Verify}(\mathsf{pk}, m, \sigma)$:
  $v \leftarrow \Sigma_{\mathrm{AS}}.\mathsf{Verify}(\mathsf{pk}, m, \sigma)$
  return v

$\mathsf{Agg}(f, \{(\mathsf{pk}_i, m_i, \sigma_i)\}_{i\in[n]})$:
  if $f = \epsilon$ **then** $P := [n]$
  else $P := f$
  parse $(S_i)_{i\in[|P|]} \leftarrow P$
  for $j := 1$ to $|P|$ **do**
    $\tau_j \leftarrow \Sigma_{\mathrm{AS}}.\mathsf{Agg}(\{(\mathsf{pk}_i, m_i, \sigma_i)\}_{i\in S_j})$
  return $\tau := (\tau_j)_{j\in[|P|]}$

$\mathsf{PartVerify}(\beta, \{(\mathsf{pk}_j, m_j)\}_{j\in[n]}, \tau, i)$:
  parse $(f, W, t) \leftarrow \beta$
  parse $(S_i)_{i\in[|P|]} \leftarrow P$; $(\tau_i)_{i\in[|\tau|]} \leftarrow \tau$
  find i' s.t. $i \in S_{i'}$
  $v \leftarrow \Sigma_{\mathrm{AS}}.\mathsf{AggVerify}(\{(\mathsf{pk}_j, m_j)\}_{j\in S_{i'}}, \tau_{i'})$
  return v

$\mathsf{Trace}(\beta, \{(\mathsf{pk}_i, m_i)\}_{i\in[n]}, \tau)$:
  if $\beta = \epsilon$ **then** $P := \emptyset$, $W := [n]$, $t := 0$
  else parse $(P, W, t) \leftarrow \beta$
  parse $(S_i)_{i\in[|P|]} \leftarrow P$ and $(\tau_i)_{i\in[|\tau|]} \leftarrow \tau$
  for $i := 1$ to $|\tau|$ **do**
    $v \leftarrow \Sigma_{\mathrm{AS}}.\mathsf{AggVerify}(\{(\mathsf{pk}_j, m_j)\}_{j\in S_i}, \tau_i)$
  $t \leftarrow t + 1$
  if $t = q$ **then** $V := W \setminus \bigcup_{v_i=1} S_i$
  else $V := \emptyset$; $W' := W \setminus \bigcup_{v_i=1} S_i$
  $f' \leftarrow (T_{qt}, \ldots, T_{qt+(q-1)})$; $\beta' \leftarrow (f', W', t)$
  return (β', f', V)

Fig. A·1 The ASIT scheme multi-HKK$^+$ based on an ordinary aggregate signature scheme Σ_{AS} and a cover free family.

Ryu Ishii received the B.E. degree from Keio University and the M.E. degree from the University of Tokyo, in 2019 and 2021, respectively. As of 2022, he is a Ph.D. student at the University of Tokyo, Graduate School of Information Science and Technology. As of 2022, he is a research assistant at the National Institute of Advanced Industrial Science and Technology (AIST), Japan.

Kyosuke Yamashita received the B.E., M.E. and Ph.D. degrees from Kyoto University, in 2013, 2015 and 2021, respectively. Currently he is a postdoctoral researcher at National Institute of Advanced Industrial Science and Technology (AIST), Japan. He received SCIS Paper Prize from IEICE in 2019.

Yusuke Sakai received his B.E., M.E., and Ph.D. degrees from the University of Electro-Communications, Tokyo, Japan, in 2009, 2011, and 2014, respectively. From 2012 to 2014, and from 2014 to 2017, he was a research fellow of Japan Society for the Promotion of Science (JSPS). In 2017, he joined the National Institute of Advanced Industrial Science and Technology (AIST), Japan. He is presently engaged in research on cryptography. He received SCIS Paper Prize from IEICE in 2011 and the Best Student Award in IWSEC 2010.

Tadanori Teruya received the M.E. degree and the Ph.D. degree in engineering from the University of Tsukuba, Japan, in 2009 and 2012, respectively. He worked as a postdoctoral researcher in the Faculty of Engineering, Information and Systems, University of Tsukuba, Japan (2012–2013), and in the National Institute of Advanced Industrial Science and Technology (AIST), Japan (2013–2016), and then he worked as a researcher in AIST (2016–2018). He is currently a senior researcher in AIST since 2018. His research interests include cryptography, information security, and privacy-enhancing technologies, in particular, practical aspects of cryptography based on elliptic curves, lattices, and secure computation and their applications.

Takahiro Matsuda received his bachelors, masters, and Ph.D. degrees in Information and Communication Engineering from the University of Tokyo in 2006, 2008, and 2011, respectively. From 2009 to 2011 and from 2011 to 2013, he had been a Research Fellow of Japan Society for the Promotion of Science (JSPS). From 2011, he has been with the National Institute of Advanced Industrial Science and Technology (AIST), Japan, where he currently works as a senior research scientist. His research interests are in the areas of public key cryptography and theory of cryptography.

Goichiro Hanaoka graduated from the Department of Engineering, the University of Tokyo in 1997. He received the Ph.D. degree from the University of Tokyo in 2002. Goichiro joined AIST in 2005. Currently, he is a Prime Senior Researcher, Cyber Physical Security Research Center, AIST. He engages in the R&D for encryption and information security technologies including the efficient design and security evaluation of public key cryptosystems. He has received numerous awards including the DoCoMo Mobile Science Award (2016), Mobile Communication Fund; the Wilkes Award (2007), British Computer Society; Best Paper Award (2008, 2019), The Institute of Electronics, Information and Communication Engineers (IEICE); and Innovative Paper Awards (2012, 2014), Symposium on Cryptography & Information Security (SCIS), IEICE.

Kanta Matsuura received his Ph.D. degree in electronics from the University of Tokyo in 1997. He is currently a Professor of Institute of Industrial Science at the University of Tokyo. His research interests include cryptography, cybersecurity, and security management such as security economics. He was an Associated Editor of IPSJ Journal (2001–2005) and IEICE Transactions on Communications (2005–2008), and won Distinguished-Service Award from the IEICE Communications Society in 2008. He was Editor-in-Chief of Security Management (2008–2012), and is an Editorial-Board member of Design, Codes, and Cryptography (2010-present). He is a fellow of IPSJ, and a senior member of IEEE, ACM, and IEICE. He is President of JSSM (Japan Society of Security Management) (2021-present). He is a member of Science Council of Japan (2017-present).

Tsutomu Matsumoto is a professor of the Faculty of Environment and Information Sciences, Yokohama National University. He also serves as the Director of the Cyber Physical Security Research Center (CPSEC) at the National Institute of Advanced Industrial Science and Technology (AIST). Starting from Cryptography in the early '80s, Prof. Matsumoto has opened up the field of security measuring for logical and physical security mechanisms. He received a Doctor of Engineering degree from the University of Tokyo in 1986. He serves as the chair of the Japanese National Body for ISO/TC68 (Financial Services) and the Cryptography Research and Evaluation Committees (CRYPTREC) and as an associate member of the Science Council of Japan (SCJ). He received the IEICE Achievement Award, the DoCoMo Mobile Science Award, the Culture of Information Security Award, the MEXT Prize for Science and Technology, and the Fuji Sankei Business Eye Award.