IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 28 October 2023, accepted 15 January 2024, date of publication 26 January 2024, date of current version 5 February 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3358849

RESEARCH ARTICLE

Insecurity of Chait et al.'s RSA-Based Aggregate Signature Scheme

CHANHYEOK PARK[1], SANGRAE CHO[2], YOUNG-SEOB CHO[2], SOOHYUNG KIM[2], AND HYUNG TAE LEE[1]

[1]School of Computer Science and Engineering, Chung-Ang University, Seoul 06974, Republic of Korea
[2]Electronics and Telecommunications Research Institute (ETRI), Daejeon 34129, Republic of Korea

Corresponding author: Hyung Tae Lee (hyungtaelee@cau.ac.kr)

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. RS-2023-00229400, Development of user authentication and privacy preserving technology for a secure metaverse environment).

ABSTRACT Recently, Chait et al. proposed a new aggregate signature scheme under the RSA setting (IEEE Access, 2023). In this paper, we show that Chait et al.'s aggregate signature scheme is *insecure* when two signers collude with their own secret keys, by presenting an attack algorithm that forges aggregate signatures of aggregator or individual signatures of all other (non-colluding) users. More concretely, our attack algorithm consists of three sub-algorithms: The first sub-algorithm computes a multiple of $\phi(N)$ from secret keys of two users where N is the RSA modulus that is included in the public parameter of the system and ϕ is the Euler totient function. The second sub-algorithm recovers an equivalent secret key of a target user that is congruent to his/her original secret key modulo $\phi(N)$ from his/her public key and the multiple of $\phi(N)$ which is the output of the first sub-algorithm. Finally, with the equivalent secret key obtained by the second sub-algorithm, the last sub-algorithm generates valid aggregate/individual signatures of the target user. Our attack algorithm always succeeds in forging aggregate/individual signatures. Furthermore, it is lightweight in the sense that it requires several integer operations, gcd computations, and an execution of aggregate/individual signing algorithm only. For example, when the pubic parameter and secret keys of all users, except the target user, are provided, our experimental results demonstrate that the proposed attack algorithm takes less than 1 second only in total to forge an aggregate signature of 29 individual signatures including that of the target user, where N is 3,072 bits for 128-bit security.

INDEX TERMS Security analysis, aggregate signature, RSA-based, collusion attacks, secret key recovery.

I. INTRODUCTION

An aggregate signature scheme is a specialized type of signature scheme that offers an additional functionality: It allows to combine multiple signatures from different users on different messages into a compact aggregate signature. Then, when verified, this aggregate signature serves as a convincing proof to the verifier that all individual signatures involved are valid. It enables us to reduce the storage for storing signatures and to improve the efficiency for verification of signatures. So, it can be applied for various scenarios, e.g., sensor networks [1], vehicular ad-hoc network [2], smart city applications [3], and blockchain [4], [5]. Due to its wide-ranging applications, since it was firstly introduced by Boneh et al. [6], there have been proposed numerous aggregate signature schemes and their variants under diverse settings, e.g., discrete-log (DL) [1], [2], [3], [4], [5], [7], [8], [9], [10], [11], [12], [13], [14], lattices [15], [16], codes [17], and RSA [18], [19], [20], [21].

However, while many aggregate signature schemes and their variants have been proposed in the DL setting [1], [2], [3], [4], [5], [7], [8], [9], [10], [11], [12], [13], [14], only a few (variants of) aggregate signatures have been proposed in the RSA setting [18], [19], [20], [21]. For instance, in [18], Lysyanskaya et al. proposed sequential aggregate signature schemes from trapdoor permutations, in which

The associate editor coordinating the review of this manuscript and approving it for publication was Barbara Masucci.

the set of signers is ordered, and provided its instantiation under the RSA setting where several distinct RSA moduli were used. Selvi et al. [19] presented an identity-based aggregate signature scheme. Their construction hires only one RSA modulus, but a secret key of each user is an RSA signature of identity and so they did not exploit the key generation paradigm of the original RSA cryptosystem [22] several times. Guo and Wang [20] proposed an RSA-based synchronized aggregate signature scheme, in which a signer can generate at most one signature for each time period and aggregation of signatures is only possible for signatures generated within the same time period. However, in their construction, each user has the same secret key which is composed of factors of the common RSA modulus. So, it is insecure since the adversary in the security game of aggregate signature scheme can have secret keys of all users, except the target user. Later, Hohenberger and Waters [21] proposed another synchronized aggregate signature scheme under the RSA setting. Their construction uses only one RSA modulus N, but it exploits relations of DLs over a multiplicative group $\mathbb{Z}_N^*$ and does not generate a pair of public and secret keys as in the original RSA signature scheme [22] as well. Thus, to the best of our knowledge, there had been no known aggregate signature scheme under the RSA setting where it uses only one RSA modulus as well as exploits the key generation paradigm of the original RSA signature scheme which generates a random element and its inverse modulo $\phi(N)$ as public and secret keys, respectively, to generate a pair of public and secret keys of each user, where N is the RSA modulus and ϕ is the Euler totient function.

Very recently, Chait et al. proposed a new aggregate signature scheme under the RSA setting [23], where all users in their construction use the same RSA modulus N which is the product of two (safe) primes. In the general RSA setting, if a user has a pair of public and secret keys (e, d) satisfying $e \times d = 1 \pmod{\phi(N)}$, then he/she can recover factors of N, which is the secret information in the RSA-based cryptosystem. In fact, this feature is the main obstacle when designing a cryptosystem for multiple users under the RSA setting, while adhering to the key generation paradigm of the original RSA cryptosystem [22] with hiring only one RSA modulus. To avoid this obstacle, in the key generation algorithm of Chait et al.'s construction, it is assumed that there exists a trusted-third party (TTP) and the TTP generates pairs of public and secret keys of all users including the RSA modulus N. For this purpose, it first generates N and a pair (e_r, d_r) satisfying

$$e_r \times d_r = 1 \text{ and } d_r = t \times r \pmod{\phi(N)}$$

for some integers t and r. Then, it generates a pair of public and secret keys (e_i, d_i') for each user U_i so that

$$e_i \times d_i = 1 \text{ and } d_i' = d_i + r \pmod{\phi(N)}.$$

Here, it was expected that the exact value d_i is hard to be calculated when d_i' and other public parameters are given, because it was masked by r. In addition, even though secret keys of multiple users are provided as in the traditional security game of aggregate signature schemes, it was also claimed that recovering a secret key of the target user is hard and so forging a signature of the target user is infeasible.

In this paper, we analyze the security of Chait et al.'s aggregate signature scheme. To that end, we present an attack algorithm that forges Chait et al.'s aggregate/individual signatures of the target user when secret keys of two other users are given. We note that in the traditional security game of aggregate signature schemes, the adversary can possess secret keys of all users, except the target user, thus the requirement for our attack algorithm is natural in the security model of aggregate signature schemes.

Our attack algorithm can be divided into three sub-algorithms. The first sub-algorithm takes secret keys of two users, the public parameter including the RSA modulus N and public keys of users as inputs, and then returns a multiple of $\phi(N)$. The second sub-algorithm takes a public key of the target user and a multiple of $\phi(N)$, which is obtained by the first sub-algorithm, as inputs, and returns an equivalent secret key of the target user that is congruent to the original secret key of the target user modulo $\phi(N)$. In fact, after executing the second sub-algorithm, the adversary can get the equivalent secret key of the target user which is congruent to the original secret key of the target user modulo $\phi(N)$, and so it can generate any valid aggregate/individual signatures of the target user. Finally, with the equivalent secret key of the target user obtained by the second sub-algorithm and secret keys of other users, the adversary can generate a valid aggregate signature and an individual signature of the target user by calling Chait et al.'s original aggregate algorithm and signing algorithm, respectively.

Our attack algorithm is very efficient: It requires several integer operations, gcd computations, and an execution of Chait et al.'s original aggregate/signing algorithm only. We confirm it by presenting implementation results of our attack algorithm. Our experimental results demonstrate that for 128-bit security with 3,072-bit RSA modulus N, our attack algorithm takes less than 1 second in total to forge an aggregate signature of 29 individual signatures, when secret keys of two users as well as the public parameters are provided. In particular, under the same setting as above, it takes less than 30 ms to recover an equivalent secret key of the target user by running the first and second sub-algorithms sequentially. As the running time of the aggregation algorithm of Chait et al.'s scheme is proportional to the value of t which is the number of signatures to be aggregated at a time, our attack algorithm also takes a longer time as t is larger. However, our attack algorithm takes about 5.2 seconds only in total to forge an aggregate signature of $t = 1{,}009$ individual signatures when secret keys of all users, except the target user, are provided.

Since our attack requires secret keys of two users, Chait et al.'s scheme may be secure if there is only one user in the system or it is assumed that all users do not collude each other. However, these assumptions are too restricted in

aggregate signature schemes. Thus, their construction lost its own attraction as aggregate signature. Furthermore, it seems hard to fix their construction with maintaining the current setting that not only uses the common RSA modulus in the system, but also follows the key generation paradigm of the original RSA cryptosystem for key generation of users in the system, because a pair of public and secret keys generated by the RSA key generation algorithm may seem to leak the factoring information of the corresponding RSA modulus. Therefore, it also seems to be a challenging task to fix their scheme for supporting aggregation under the current setting and we leave it as an open problem.

Outline of the Paper. In the following section, we provide an overview of related works on aggregate signature schemes. Section III presents a comprehensive review of Chait et al.'s RSA-based aggregate signature scheme, including its system model, description, and security model. Our attack algorithm with theoretical analysis and toy example is given in Section IV. Section V provides experimental results of our attack algorithm. We conclude with remarks in Section VI.

II. RELATED WORKS

In this section, we provide a brief survey on RSA-based aggregate signature schemes, which are closely related to the work in this paper. The concept of aggregate signature with its instantiation was firstly introduced by Boneh et al. [6]. In aggregate signature schemes, multiple individual signatures from different users on different messages can be combined into a compact aggregate signature and then used to convince a verifier that all individual signatures involved are valid by verifying an aggregate signature only. It can be widely applied for various scenarios since it enables us to reduce the storage and to improve the efficiency of verification. Thus, there have been proposed variants of aggregate signatures, identity-based [8], [13], certificateless [2], sequential [7], [11], synchronized [1], and unrestricted [10]. However, most of existing aggregate signature schemes were mainly designed under the DL setting.

The first RSA-based aggregate signature scheme was presented by Lysyanskaya et al. [18]. They presented a generic construction of sequential aggregate signatures from trapdoor permutations, in which the set of signers to be aggregated is ordered, and provided an instantiation of their generic construction under the RSA setting. In their RSA-based instantiation, each signature is generated using a distinct RSA modulus: The authors of [18] first considered the case that RSA moduli satisfy the order $N_1 < N_2 < \cdots < N_\ell$ where N_i is the RSA modulus for the i-th signature in the aggregation sequence. Then, they presented a way to remove such a restriction. But, both cases exploit several RSA moduli in the system.

Later, in [19], Selvi et al. proposed an identity-based aggregate signature scheme from the (strong) RSA assumption. In their identity-based signature scheme, the trusted authority has a pair of public and secret keys of the original RSA signature scheme as the public parameter and the master secret key, respectively. Then, it issues a signature of the hash value of identity ID as the secret key of user ID. Then, each user who has his/her own secret key generates a signature of message by calculating modular exponentiations with message-dependent exponent. Their construction can be easily extended to identity-based aggregate signature schemes. So, their construction hires only one RSA modulus, but it does not use the key generation algorithm of the original RSA signature scheme which generates a pair of (e, d) such that $e \times d = 1 \pmod{\phi(N)}$, to issue a pair of public and secret keys of each user, where N is an RSA modulus and ϕ is the Euler totient function.

Guo and Wang [20] presented a synchronized aggregate signature scheme from the RSA assumption. In the synchronized aggregate signature scheme, an individual signer can generate at most one signature for each time period and only the set of signatures generated in the same period can be aggregated. However, in their construction, each user has the same secret key which consists of the factors of the common RSA modulus. It does not match the traditional security model of aggregate signature schemes where the adversary can have secret keys of all users, except the target user. That is, the construction proposed by Guo and Wang is insecure under the traditional security model of aggregate signature schemes.

In 2018, Hohenberger and Waters [21] proposed a synchronized aggregate signature scheme from the RSA assumption. In their construction, the key generation algorithm generates a product of e_i's, $E = \prod_{i=1}^{\ell} e_i$, where e_i's are easily computable. Then, when a signature is generated at time j, it computes a modular exponentiation of original signature with exponent E/e_j and then verifies using e_j and E. Thus, all users in their construction use the same RSA modulus, but it does not follow the key generation paradigm of the original RSA cryptosystem to generate each user's key.

Finally, very recently, Chait et al. [23] presented the RSA-based aggregate signature scheme. Differently from the previous (secure) constructions, Chait et al.'s construction follows the key generation paradigm of the original RSA cryptosystem to generate public and secret keys of each user. In the key generation algorithm of their construction, the TTP first generates an RSA modulus N and computes (e_r, d_r) such that

$$e_r \times d_r = 1 \text{ and } d_r = t \times r \pmod{\phi(N)}$$

for some integers t and r. Then, it generates a pair of public and secret keys (e_i, d_i') for each user U_i so that

$$e_i \times d_i = 1 \text{ and } d_i' = d_i + r \pmod{\phi(N)}.$$

The authors of [23] claimed that it is hard to calculate d_i' and/or d_i for the target user, when public parameters and secret keys of other users are given. However, in this paper, we show that their construction is not secure. Refer to Sections III and IV for the details of Chait et al.'s scheme and our attack algorithm, respectively.

III. CHAIT ET AL.'S RSA-BASED AGGREGATE SIGNATURE SCHEME

In this section, we review Chait et al.'s RSA-based aggregate signature scheme, including its system model, description, and security model.

A. SYSTEM MODEL FOR CHAIT ET AL.'S RSA-BASED AGGREGATE SIGNATURE SCHEME

We first recall the system model for Chait et al.'s scheme. It consists of a TTP and users. Throughout the paper, we denote by n the number of users that have the signing ability in the system. The TTP generates a public parameter of the system, and a pair of secret and public keys for each user in the system. Each user may play roles of both aggregator and individual signer. For simple description, we assume that user 1 is an aggregator and a signer, but all other users are just signers.

Once the system is established, the TTP first generates a public parameter, and a pair of secret and public keys for each user. Then, it publishes the public parameter and the public keys of all users. The secret key of user i is passed to user i through a secure channel. When user 1 who is the aggregator would like to generate an aggregate signature of message, it first generates a signature of that message and diffuses it in the system. Once each user receives that signature, it verifies, and if the verification is passed, then it generates a signature of the corresponding message under its own secret key. When user 1 collects t signatures including its own signature, it aggregates them into one aggregate signature. In this process, it additionally generates a signature of the list of users that contribute to generate the aggregate signature using its own secret key. Finally, anyone can confirm the validity of all signatures combined into the aggregate signature by checking the validity of the aggregate signature as well as the signature of the list of involved users.

B. DESCRIPTION OF CHAIT ET AL.'S RSA-BASED AGGREGATE SIGNATURE SCHEME

Now, we present the description of Chait et al.'s RSA-based aggregate signature scheme. Chait et al.'s scheme can generate an aggregate signature once t individual signatures are collected. Such an aggregate signature scheme is called a t-aggregate signature scheme. It consists of the following five algorithms: KeyGen, $\mathsf{IndividualSign}$, $\mathsf{IndividualVerify}$, $\mathsf{AggSign}$, and $\mathsf{AggVerify}$.

- $\mathsf{KeyGen}(1^\lambda)$: This algorithm is run by the TTP. Given the security parameter λ, it performs as follows:
  1) Select two primes p, q and set $N = p \times q$.
  2) Compute $\phi(N) = (p-1) \times (q-1)$.
  3) Set the value of t, which corresponds to the number of signatures required for generating a valid aggregate signature. (We note that $\gcd(t, \phi(N)) = 1$ for the correctness.)
  4) Pick a random element d_r such that
  $$d_r = t \times r \pmod{\phi(N)} \text{ and } \gcd(d_r, \phi(N)) = 1.$$
  5) Compute e_r such that
  $$e_r \times d_r = 1 \pmod{\phi(N)}.$$
  6) For $\ell = 1, \ldots, n$, generate a pair (e_ℓ, d_ℓ) such that
  $$e_\ell \times d_\ell = 1 \pmod{\phi(N)}.$$
  7) Compute $d'_\ell = d_\ell + r \pmod{\phi(N)}$ for $\ell = 1, \ldots, n$.
  8) Generate a cryptographic hash function $\mathsf{H} : \{0,1\}^* \to \mathbb{Z}_N$.
  9) Publish a public parameter
  $$\mathsf{pp} = (N, t, e_r, \{e_\ell\}_{1 \le \ell \le n}) \text{ and } \mathsf{H},$$
  and pass a secret key $\mathsf{sk}_\ell = d'_\ell$ of user U_ℓ to U_ℓ through a secure channel.
- $\mathsf{IndividualSign}$: This algorithm is run by an individual user. It behaves differently with respect to the executor.
  - $\mathsf{IndividualSign}(\mathsf{pp}, \mathsf{H}, \mathsf{sk}_1, m)$: If it is run by the aggregator (i.e., user U_1), it takes the public parameter $\mathsf{pp} = (N, t, e_r, \{e_\ell\}_{1 \le \ell \le n})$, the hash function H, the secret key $\mathsf{sk}_1 = d'_1$ of user U_1, and a message m as inputs, and performs as follows:
    1) Calculate $h = \mathsf{H}(m)$.
    2) Calculate $s_1 = h^{d'_1} \pmod{N}$.
    3) Output (m, s_1).
  - $\mathsf{IndividualSign}(\mathsf{pp}, \mathsf{H}, \mathsf{sk}_\ell, (m, s_1))$: If it is run by an individual signer U_ℓ with $\ell \neq 1$, it takes the public parameter $\mathsf{pp} = (N, t, e_r, \{e_\ell\}_{1 \le \ell \le n})$, the hash function H, the secret key $\mathsf{sk}_\ell = d'_\ell$ of user U_ℓ, and a signature (m, s_1) of user U_1 as inputs, and performs as follows:
    1) Run $\mathsf{IndividualVerify}(\mathsf{pp}, \mathsf{H}, (m, s_1), 1)$, described below. If it outputs 1, then proceed the following steps. Otherwise, abort.
    2) Calculate $h = \mathsf{H}(m)$.
    3) Calculate $s_\ell = h^{d'_\ell} \pmod{N}$.
    4) Output (m, s_ℓ).
- $\mathsf{IndividualVerify}(\mathsf{pp}, \mathsf{H}, (m, s), \ell)$: This algorithm can be executed by anyone. Given the public parameter $\mathsf{pp} = (N, t, e_r, \{e_\ell\}_{1 \le \ell \le n})$, the hash function H, a signature (m, s) of user U_ℓ and the index ℓ, it runs as follows:
  1) Calculate $h = \mathsf{H}(m)$.
  2) Calculate $y = h^{e_r \times t + e_\ell} \pmod{N}$.
  3) Calculate $y' = s^{e_r \times t \times e_\ell} \pmod{N}$.
  4) Check if $y = y' \pmod{N}$. If it holds, return 1. Otherwise, return 0.
- $\mathsf{AggSign}(\mathsf{pp}, \mathsf{H}, \mathsf{sk}_1, (m, \{s_\ell\}_{\ell \in I, |I| = t}))$: This is done by the aggregator (i.e., user U_1). Given the public parameter

$\mathsf{pp} = (N, t, e_r, \{e_\ell\}_{1\le\ell\le n})$, the hash function H, the secret key sk_1 of user U_1, and the message m and t signatures $\{s_\ell\}_{\ell\in I}$ of message m where I is the set of indices of cardinality t, it performs as follows:

1) Run $\mathsf{IndividualVerify}(\mathsf{pp}, \mathsf{H}, (m, s_\ell), \ell)$ for each $\ell \in I$. If at least one of results is 0, then abort. Otherwise, proceed the following steps.
2) Calculate $\sigma = \prod_{\ell\in I} s_\ell \pmod N$.
3) Initialize L as an empty string.
4) Set $L = \ell_{j_1}\|\ell_{j_2}\|\cdots\|\ell_{j_t}$ by attaching all elements in $I = \{\ell_{j_1}, \ell_{j_2}, \cdots, \ell_{j_t}\}$.
5) Run $\mathsf{IndividualSign}(\mathsf{pp}, \mathsf{H}, \mathsf{sk}_1, L)$ to obtain (L, SL) where SL is a signature of message L under the signing key of the aggregator (i.e., user U_1).
6) Output (m, σ, L, SL).

- $\mathsf{AggVerify}(\mathsf{pp}, \mathsf{H}, (m, \sigma, L, SL))$: Given the public parameter $\mathsf{pp} = (N, t, e_r, \{e_\ell\}_{1\le\ell\le n})$, the hash function H, an aggregate signature (m, σ, L, SL), it performs as follows:
  1) Run $\mathsf{IndividualVerify}(\mathsf{pp}, \mathsf{H}, (L, SL), 1)$. If it outputs 0, abort. Otherwise, proceed the following steps.
  2) Calculate $h = \mathsf{H}(m)$.
  3) Calculate $h' = h^{e_r} \pmod N$.
  4) Set $I = \{\ell_{j_1}, \ell_{j_2}, \cdots, \ell_{j_t}\}$ by parsing $L = \ell_{j_1}\|\ell_{j_2}\|\cdots\|\ell_{j_t}$.
  5) Calculate
$$E = \sum_{i\in I}\left(\prod_{j\in I, j\neq i} e_j\right).$$
  6) Calculate $s = h'^E \pmod N$.
  7) Calculate $E' = \prod_{j\in I} e_j$.
  8) Calculate $s' = (\sigma^{e_r} \times h^{-1})^{E'} \pmod N$.
  9) Check if $s = s'$. If it holds, return 1. Otherwise, return 0.

We investigate the correctness of verification algorithms for individual and aggregate signatures of Chait et al.'s scheme. First, the correctness of the verification algorithm for individual signature (m, s) comes from the relation that

$$\begin{aligned} s^{e_r\times t\times e_\ell} &= (h^{d'_\ell})^{e_r\times t\times e_\ell} = h^{(d_\ell+r)\times e_r\times t\times e_\ell} \\ &= (h^{d_\ell\times e_r\times t\times e_\ell}) \times (h^{r\times e_r\times t\times e_\ell}) \\ &= h^{e_r\times t} \times h^{e_\ell} = h^{e_r\times t+e_\ell} = y \pmod N \end{aligned}$$

where $h = \mathsf{H}(m)$ since

$$\begin{aligned} d_\ell \times e_\ell &= 1 \pmod{\phi(N)} \text{ and} \\ e_r \times d_r &= e_r \times t \times r = 1 \pmod{\phi(N)}. \end{aligned}$$

Next, the correctness of the verification algorithm for aggregate signature (m, σ, L, SL) comes from the validity of individual signature (L, SL) and the relation that

$$\begin{aligned} s' &= (\sigma^{e_r} \times h^{-1})^{E'} = \Big(\big(\prod_{i\in I} h^{d'_i}\big)^{e_r} \times h^{-1}\Big)^{E'} \\ &= \left(h^{(\sum_{i\in I} d'_i)e_r - 1}\right)^{E'} = \left(h^{(\sum_{i\in I}(d_i+r))e_r - 1}\right)^{E'} \\ &= \left(h^{(tr+\sum_{i\in I} d_i)e_r - 1}\right)^{E'} = \left(h^{(\sum_{i\in I} d_i)e_r}\right)^{E'} \\ &= (h^{e_r})^{(\sum_{i\in I} d_i)E'} = h'^{(\sum_{i\in I} d_i)\prod_{j\in I} e_j} \\ &= h'^{(\sum_{i\in I}\prod_{j\in I, j\neq i} e_j)} = h'^E = s \pmod N. \end{aligned}$$

C. SECURITY MODEL FOR CHAIT ET AL.'S RSA-BASED AGGREGATE SIGNATURE SCHEME

Next, we take a look at the security model for Chait et al.'s scheme. It follows the original security model of aggregate signature schemes [6], [21] and is defined using the security game between the adversary $\mathcal{A}$ and the challenger $\mathcal{C}$.

We say that t-aggregate signature scheme is secure against existential forgery in the aggregate chosen-key model if there is no probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ whose advantage is negligible in the security parameter λ in the following game with the challenger $\mathcal{C}$:

- **Setup**: $\mathcal{A}$ receives a public key pk_1 for user U_1 which is randomly generated and $(\mathsf{pk}_i, \mathsf{sk}_i)$ for user U_i with $2 \le i \le t$.
- **Queries**: $\mathcal{A}$ may request signatures with the public key pk_1 on the message chosen by $\mathcal{A}$ itself.
- **Response**: $\mathcal{A}$ outputs an aggregate signature σ_{agg} which is composed of individual signatures of $U_1, \ldots, U_t$.

We say that $\mathcal{A}$ wins the above game if

- $\mathsf{AggVerify}(\mathsf{pp}, \mathsf{H}, \sigma_{\mathsf{agg}}) = 1$ and
- m' did not appear in the Queries phase.

The advantage of $\mathcal{A}$ is defined as the probability that $\mathcal{A}$ wins in the above game.

Remark 1: The security definition of t-aggregate signature schemes in [23] has some typos: It is stated that a secret key $\mathsf{sk}_1 = d'_1$ is given to $\mathcal{A}$ at the Setup phase. In their original description, a secret key $\mathsf{sk}_1 = d'_1$ should be replaced by a public key $\mathsf{pk}_1 = e_1$.

We note that our attack algorithm is sufficient if it takes two secret keys d'_i and d'_j with the public parameter pp. It does not need to have access to the signing oracle. The details of our attack algorithm will be provided in the next section.

IV. OUR ATTACK ALGORITHM

In this section, we provide our attack algorithm against Chait et al.'s aggregate signature scheme with its theoretical analysis and toy example.

A. OUR KEY RECOVERY AND FORGE ALGORITHMS

We first present the description of our attack algorithm that forges an individual signature for the target user or an aggregate signature which is generated by the target user as an aggregator. Our attack algorithm can be divided into three sub-algorithms: The first sub-algorithm, called $\mathsf{ComMulPhiN}$, computes a multiple of $\phi(N)$ when secret

keys d'_i, d'_j of two users U_i, U_j, respectively, in addition to the public parameter pp are given as inputs where N is the common RSA modulus of the system and ϕ is the Euler totient function. The second algorithm, called RecSecKey, recovers a value congruent to the secret key of the target user modulo $\phi(N)$ when the public key of the target user and the multiple of $\phi(N)$, which is obtained from the first algorithm, are given as inputs. The last algorithm, called Forge, generates an individual or aggregate signature using outputs of the first and second sub-algorithms.

For our attack, we assume that at least two users collude with their own secret keys and thus the adversary has secret keys d'_i, d'_j of two users U_i, U_j, respectively. This requirement is natural since in the security game it is assumed that the adversary has secret keys of all users, except the target user.

Let us elaborate our first algorithm that computes a multiple of $\phi(N)$. This algorithm takes the public parameter $\mathsf{pp} = (N, t, e_r, \{e_\ell\}_{1\le\ell\le n})$ and d'_i, d'_j as inputs. That is, we assume that users U_i and U_j collude to recover a secret key of user U_k. The algorithm computes

$$phi = (e_i e_j d'_i - e_j e_i d'_j) + (e_i - e_j).$$

Then, phi is a multiple of $\phi(N)$. (See Theorem 1 in Section IV-B for the correctness of the first algorithm.) The formal description of the first algorithm is given in Algorithm 1.

Algorithm 1 Compute a Multiple of $\phi(N)$: ComMulPhiN(pp, D'_i, D'_j)

Input: The public parameter $\mathsf{pp} = (N, t, e_r, \{e_\ell\}_{1\le\ell\le n})$, secret keys d'_i, d'_j of users U_i, U_j, respectively
Output: An integer phi

$phi \leftarrow e_i \times e_j \times d'_i$ (as integers)
$phi \leftarrow phi - e_i \times e_j \times d'_j$ (as integers)
$phi \leftarrow phi + (e_i - e_j)$ (as integers)
if $phi < 0$ **then**
    $phi \leftarrow -phi$
return phi

Next, on top of Algorithm 1, the second algorithm recovers an equivalent secret key congruent to the secret key d'_k of the target user U_k modulo $\phi(N)$ from the target public key e_k and phi, which is the output of Algorithm 1. Suppose that the second algorithm already obtained phi by running the first algorithm ComMulPhiN(pp, d'_i, d'_j). In the second algorithm with inputs $\mathsf{pp} = (N, t, e_r, \{e_\ell\}_{1\le\ell\le n})$, d'_i, d'_j and phi, it should calculate the inverse of e_k modulo phi, i.e., $e_k^{-1} \pmod{phi}$, where e_k is the public key of the target user U_k. Thus, $\gcd(e_k, phi) = 1$ should hold. Otherwise, i.e., if $g = \gcd(e_k, phi) \neq 1$, then the algorithm sets phi to phi/g and repeats this process until $\gcd(e_k, phi) = 1$. This additional process does not violate the correctness of our attack algorithm because the key generation algorithm of Chait et al.'s scheme selects e_k so that $\gcd(e_k, \phi(N)) = 1$. Similarly, $\gcd(e_r, phi) = 1$ and $\gcd(t, phi) = 1$ should also hold to calculate the inverse of $e_r \times t$ modulo phi and in fact they trivially hold since the key generation algorithm of Chait et al.'s scheme selects e_r and t such that $\gcd(e_r, \phi(N)) = 1$ and $\gcd(t, \phi(N)) = 1$. Then, it calculates

$$\overline{r} = (e_k \times t)^{-1} \pmod{phi}$$

and then

$$\overline{d'_k} = e_k^{-1} + \overline{r} \pmod{phi}.$$

Algorithm 2 provides the formal description of our second algorithm. The correctness of Algorithm 2 will be shown in Theorem 2 of Section IV-B.

Algorithm 2 Recover a Secret Key: RecSecKey(pp, D'_i, D'_j, Phi, k)

Input: The public parameter $\mathsf{pp} = (N, t, e_r, \{e_\ell\}_{1\le\ell\le n})$, secret keys d'_i, d'_j of users U_i, U_j, respectively, the modulus phi and the index of the target user U_k with $k \neq i, j$
Output: An integer $\overline{d'_k}$

while $\gcd(e_k, phi) \neq 1$ **do**
    $phi \leftarrow phi/\gcd(e_k, phi)$
while $\gcd(e_r, phi) \neq 1$ **do**
    $phi \leftarrow phi/\gcd(e_r, phi)$
while $\gcd(t, phi) \neq 1$ **do**
    $phi \leftarrow phi/\gcd(t, phi)$
$\overline{r} \leftarrow (e_r \times t)^{-1} \pmod{phi}$
$\overline{d_k} \leftarrow e_k^{-1} \pmod{phi}$
$\overline{d'_k} \leftarrow \overline{d_k} + \overline{r} \pmod{phi}$
return $\overline{d'_k}$

Finally, on top of ComMulPhiN and RecSecKey algorithms, we build the forge algorithm that generates an individual or aggregate signature of the target user U_k with respect to the input mode $mode \in \{\mathsf{Ind}, \mathsf{Agg}\}$ in Algorithm 3. It takes $\mathsf{pp} = (N, t, e_r, \{e_\ell\}_{1\le\ell\le n})$, secret keys $\{d'_\ell\}_{\ell\in I}$ where I is the set of indices of cardinality $t-1$, the index k of the target user U_k where $k \notin I$, a cryptographic hash function $H : \{0,1\}^* \to \mathbb{Z}_N$, and a mode $mode \in \{\mathsf{Ind}, \mathsf{Agg}\}$ as inputs. The input $mode$ determines a type of output signature between an individual signature and an aggregate signature: Ind and Agg indicate an individual signature and an aggregate signature, respectively. We remark that our algorithm requires secret keys of two users only to recover an equivalent secret key of the target user and to forge an individual signature. However, by considering the case for forging an aggregate signature as an aggregator, we describe the algorithm so that it takes $t-1$ secret keys as in the security game of aggregate signature schemes in Section III-C.

This algorithm first selects two indices i, j, runs ComMulPhiN(pp, d_i, d_j) to obtain phi, and then runs RecSecKey(pp, d'_i, d'_j, phi, k) to obtain $\overline{d'_k}$. Subsequently, it selects a message m and then performs with respect to the input $mode$: If $mode = \mathsf{Ind}$, then it runs the

individual signing algorithm of Chait et al.'s scheme, $\mathsf{IndividualSign}(\mathsf{pp}, \mathsf{H}, \overline{d'_k}, m)$, to generate an individual signature of the target user U_k on message m. Otherwise, i.e., if $mode = \mathsf{Agg}$, then it generates t individual signatures by running $\mathsf{IndividualSign}(\mathsf{pp}, \mathsf{H}, \overline{d'_k}, m)$ for the target user U_k to obtain (m, s_k) and then $\mathsf{IndividualSign}(\mathsf{pp}, \mathsf{H}, d'_\ell, (m, s_k))$ for users U_ℓ to obtain (m, s_ℓ) with $\ell \in I$. Then, it runs the aggregate signing algorithm of Chait et al.'s scheme, $\mathsf{AggSign}(\mathsf{pp}, \mathsf{H}, \overline{d'_k}, (m, \{s_\ell\}_{\ell \in J}))$, to obtain an aggregate signature where $J = I \cup \{k\}$. We formally describe the third algorithm in Algorithm 3 and its correctness will be shown in Theorem 3 of Section IV-B.

Algorithm 3 Forge a Signature:
$\mathsf{Forge}(\mathsf{pp}, \{d'_\ell\}_{\ell \in I}, k, \mathsf{H}, mode)$

Input: The public parameter $\mathsf{pp} = (N, t, e_r, \{e_\ell\}_{1 \le \ell \le n})$, secret keys $\{d'_\ell\}_{\ell \in I}$ of users U_ℓ with I of cardinality $|I| = t - 1$, the index of the target user U_k with $k \notin I$, a cryptographic hash function H, and $mode \in \{\mathsf{Ind}, \mathsf{Agg}\}$
Output: An individual signature $(m, \overline{s_k})$ or an aggregate signature $(m, \overline{\sigma}, \overline{L}, \overline{SL})$

Select two indices i, j from I
$phi \leftarrow \mathsf{ComMulPhiN}(\mathsf{pp}, d'_i, d'_j)$
$\overline{d'_k} \leftarrow \mathsf{RecSigKey}(\mathsf{pp}, d'_i, d'_j, phi, k)$
Select a message m
if $mode = \mathsf{Ind}$ **then**
  $(m, \overline{s_k}) \leftarrow \mathsf{IndividualSign}(\mathsf{pp}, \mathsf{H}, \overline{d'_k}, m)$
  return $(m, \overline{s_k})$
if $mode = \mathsf{Agg}$ **then**
  $(m, s_k) \leftarrow \mathsf{IndividualSign}(\mathsf{pp}, \mathsf{H}, \overline{d'_k}, m)$
  for $\ell \in I$ **do**
    $(m, s_\ell) \leftarrow \mathsf{IndividualSign}(\mathsf{pp}, \mathsf{H}, d'_\ell, (m, s_k))$
  $J \leftarrow I \cup \{k\}$
  $(m, \overline{\sigma}, \overline{L}, \overline{SL}) \leftarrow \mathsf{AggSign}(\mathsf{pp}, \mathsf{H}, \overline{d'_k}, (m, \{s_\ell\}_{\ell \in J}))$
  return $(m, \overline{\sigma}, \overline{L}, \overline{SL})$

B. ANALYSIS OF OUR FORGE ATTACK

Now, we investigate the correctness and efficiency of three algorithms presented in Section IV-A.

Correctness of Our Attack. Theorem 1 shows the correctness of Algorithm 1.

Theorem 1: Algorithm 1 with inputs $(\mathsf{pp}, d'_i, d'_j)$ always returns a multiple of $\phi(N)$ where the public parameter $\mathsf{pp} = (N, t, e_r, \{e_\ell\}_{1 \le \ell \le n})$ and secret keys d_i, d_j of users U_i, U_j, respectively, are involved in the output of the key generation algorithm KeyGen of Chait et al.'s scheme.

Proof: Let (e_i, d'_i) and (e_j, d'_j) be pairs of public and secret keys of users U_i and U_j, respectively. Then, since they are generated by the key generation algorithm of Chait et al.'s signature scheme, they hold the relations

$$e_i \times d'_i = e_i d_i + e_i r = 1 + q_i \phi(N) + e_i r \text{ and}$$
$$e_j \times d'_j = e_j d_j + e_j r = 1 + q_j \phi(N) + e_j r$$

as integers for some integers q_i and q_j. Thus,

$$\begin{aligned} e_i e_j d'_i &= e_i e_j (d_i + r) = e_i e_j d_i + e_i e_j r \\ &= e_j (1 + q_i \phi(N)) + e_i e_j r = e_j + e_j q_i \phi(N) + e_i e_j r \end{aligned}$$

and

$$\begin{aligned} e_i e_j d'_j &= e_i e_j (d_j + r) = e_i e_j d_j + e_i e_j r \\ &= e_i (1 + q_j \phi(N)) + e_i e_j r = e_i + e_i q_j \phi(N) + e_i e_j r \end{aligned}$$

Therefore,

$$\begin{aligned} phi &= e_i e_j d'_i - e_i e_j d'_j + (e_i - e_j) \\ &= (e_j + e_j q_i \phi(N) + e_i e_j r) \\ &\quad - (e_i + e_i q_j \phi(N) + e_i e_j r) + (e_i - e_j) \\ &= (e_j q_i - e_i q_j) \phi(N) \end{aligned} \tag{1}$$

which is a multiple of $\phi(N)$. If phi in Equation (1) is negative, take $-phi$ so that the output of the algorithm is always positive. However, regardless of the sign of phi, phi is still a multiple of $\phi(N)$ and the theorem is proved. ■

Remark 2: The output of Algorithm 1 cannot be 0. First, since e_i and e_j are different, $d'_i = d_i + r$ and $d'_j = d_j + r$ are also different. Suppose that $d'_i - d'_j = c$ where c is a positive integer. Then,

$$\begin{aligned} phi &= e_i e_j (d'_i - d'_j) + (e_i - e_j) \\ &= c e_i e_j + e_i - e_j \\ &= \left(\sqrt{c} e_i - \frac{1}{\sqrt{c}}\right)\left(\sqrt{c} e_j + \frac{1}{\sqrt{c}}\right) + c > 0 \end{aligned}$$

since $e_i, e_j \ge 2$ and $c \ge 1$, and so $\sqrt{c} e_i - \frac{1}{\sqrt{c}} > 0$ and $\sqrt{c} e_j + \frac{1}{\sqrt{c}} > 0$.

Similarly, suppose that $d'_i - d'_j = c$ where c is a negative integer. Then, if $c' = -c$,

$$\begin{aligned} phi &= e_i e_j (d'_i - d'_j) + (e_i - e_j) \\ &= c e_i e_j + e_i - e_j \\ &= -(c' e_i e_j + e_j - e_i) \\ &= -\left(\sqrt{c'} e_i + \frac{1}{\sqrt{c'}}\right)\left(\sqrt{c'} e_j - \frac{1}{\sqrt{c'}}\right) - c' < 0 \end{aligned}$$

since $e_i, e_j \ge 2$ and $c' \ge 1$, and so $\sqrt{c'} e_i + \frac{1}{\sqrt{c'}} > 0$ and $\sqrt{c'} e_j - \frac{1}{\sqrt{c'}} > 0$.

Therefore, the output of Algorithm 1 cannot be 0. ■

Next, the following theorem shows the correctness of Algorithm 2 when Algorithm 1 is correct.

Theorem 2: Algorithm 2 returns a value congruent to the secret key of the target user U_k modulo $\phi(N)$ if Algorithm 1 works correctly as in Theorem 1.

Proof: Suppose that Algorithm 1 returns a multiple of $\phi(N)$. That is, $phi = q\phi(N)$ for some positive integer q. Assume that $\gcd(e_k, phi) = 1$, $\gcd(e_r, phi) = 1$, and $\gcd(t, phi) = 1$. Otherwise, we can adjust it by setting phi

to phi/g while $g = \gcd(e_k, phi) \neq 1$, $g = \gcd(e_r, phi) \neq 1$, or $g = \gcd(t, phi) \neq 1$ through Steps 1–6 of Algorithm 2.

From Steps 7–9, $\overline{r}$, $\overline{d_k}$ and $\overline{d'_k}$ satisfy

$$e_r \times t \times \overline{r} = 1 \pmod{phi} \text{ and}$$
$$e_k \times \overline{d_k} = 1 \pmod{phi}.$$

Thus, it holds that

$$e_r \times t \times \overline{r} \equiv 1 \pmod{\phi(N)} \text{ and}$$
$$e_k \times \overline{d_k} \equiv 1 \pmod{\phi(N)}$$

since phi is a multiple of $\phi(N)$. So,

$$\overline{r} \equiv r \pmod{\phi(N)} \text{ and } \overline{d_k} \equiv d_k \pmod{\phi(N)}$$

from the definitions of r and d_k. Therefore, we have

$$\overline{d'_k} \equiv \overline{d_k} + \overline{r} \equiv d_k + r \equiv d'_k \pmod{\phi(N)}.$$

■

Finally, Theorem 3 shows that Algorithm 3 generates a valid individual or aggregate signature of the target user U_k.

Theorem 3: If Algorithms 1 and 2 work correctly, Algorithm 3 with inputs $(\mathsf{pp}, \{d'_\ell\}_{\ell \in I}, k, \mathsf{H}, mode)$ outputs a valid individual signature of the target user U_k when $mode = \mathsf{Ind}$ and a valid aggregate signature of the target user U_k when $mode = \mathsf{Agg}$, where the public parameter $\mathsf{pp} = (N, t, e_r, \{e_\ell\}_{1 \leq \ell \leq n})$ and secret keys $\{d'_\ell\}_{\ell \in I}$ of users U_ℓ for I which is the set of indices of cardinality $t-1$ are generated by the key generation algorithm of Chait et al.'s scheme, k is the index of the target user U_k, $\mathsf{H} : \{0,1\}^* \to \mathbb{Z}_N$ is a cryptographic hash function, and $mode \in \{\mathsf{Ind}, \mathsf{Agg}\}$.

Proof: The proof of this theorem is straightforward from the correctness of Algorithms 1, 2 of this paper and that of individual and aggregate signing algorithms of Chait et al.'s scheme. Algorithm 3 first calls $\mathsf{ComMulPhiN}(\mathsf{pp}, d'_i, d'_j)$ and $\mathsf{RecSecKey}(\mathsf{pp}, d'_i, d'_j, phi, k)$. As a result, we obtain an equivalent secret key $\overline{d'_k}$ which is congruent to the secret key of the target user U_k modulo $\phi(N)$ from the assumption that Algorithms 1 and 2 work correctly. Then, it selects a message m and then calls $\mathsf{IndividualSign}(\mathsf{pp}, \mathsf{H}, \overline{d'_k}, m)$ to forge an individual signature of the target user U_k on message m if $mode = \mathsf{Ind}$. Otherwise, if $mode = \mathsf{Agg}$, then it generates t individual signatures by running $\mathsf{IndividualSign}(\mathsf{pp}, \mathsf{H}, \overline{d'_k}, m)$ to obtain (m, s_k) and then $\mathsf{IndividualSign}(\mathsf{pp}, \mathsf{H}, d'_\ell, (m, s_k))$ to obtain (m, s_ℓ) for $\ell \in I$. Then, it runs the aggregate signing algorithm $\mathsf{AggSign}(\mathsf{pp}, \mathsf{H}, \overline{d'_k}, (m, \{s_\ell\}_{\ell \in J}))$ with t generated individual signatures where $J = I \cup \{k\}$. Therefore, since all inputs of the algorithms have the correct form, if Chait et al.'s construction is correct, Algorithm 3 returns a valid individual or aggregate signature. ■

From Theorems 1–3, we confirm that our attack succeeds.

Efficiency Analysis of Our Attack. We now analyze the efficiency of our attack algorithm by concentrating on the computationally intensive operations that primarily account for the execution time within the algorithm. Algorithm 1 takes 4 integer multiplications at Steps 1–2. Algorithm 2 requires several gcd computations at Steps 1–6, and 2 modular inverse calculations at Steps 7–8. We remark that the number of gcd computations is bounded above by $2 \log N$ when two factors of N are safe primes as in the key generation algorithm of Chait et al.'s scheme. However, according to our experiments in Section V, it is less than 4 on average if we select e_i, t, and r as random primes. Refer to Table 3 in Section V for detailed experimental results. Finally, for Algorithm 3 with input $mode = \mathsf{Ind}$, it requires executions of Algorithm 1, Algorithm 2, and Chait et al.'s individual signing algorithm each sequentially. For Algorithm 3 with input $mode = \mathsf{Agg}$, it additionally requires $t-1$ executions of the individual signing algorithm and 1 execution of the aggregate signing algorithm of Chait et al.'s scheme.

C. TOY EXAMPLE OF OUR ATTACK

In this subsection, we provide a toy example of our attack to help readers' understanding. In our example, we select two safe primes p, q as

$$p = 1,283, \quad q = 1,307,$$

and so N, $\phi(N)$ are set to

$$\begin{aligned} N &= p \times q \\ &= 1,283 \times 1,307 = 1,676,881, \\ \phi(N) &= (p-1) \times (q-1) \\ &= 1,282 \times 1,306 = 1,674,292. \end{aligned}$$

We set $t = 19$, $r = 7,919$, and compute

$$\begin{aligned} d_r &= t \times r \pmod{\phi(N)} \\ &= 19 \times 7,919 \pmod{1,674,292} = 150,461, \\ e_r &= d_r^{-1} \pmod{\phi(N)} \\ &= 150,461^{-1} \pmod{1,674,292} = 1,450,513. \end{aligned}$$

Next, we select three numbers e_1, e_2, e_3 as

$$e_1 = 3,271,$$
$$e_2 = 4,651,$$
$$e_3 = 5,003,$$

and compute

$$d_1 = 32,759,$$
$$d_2 = 123,115,$$
$$d_3 = 578,623$$

so that $e_i \times d_i \equiv 1 \pmod{\phi(N)}$ for $i = 1, 2, 3$. Then, we compute

$$\begin{aligned} d'_1 &= d_1 + r \pmod{\phi(N)} \\ &= 32,759 + 7,919 \pmod{1,674,292} \\ &= 40,678, \\ d'_2 &= d_2 + r \pmod{\phi(N)} \\ &= 123,115 + 7,919 \pmod{1,674,292} \end{aligned}$$

$$= 131,034,$$
$$d_3' = d_3 + r \pmod{\phi(N)}$$
$$= 578,623 + 7,919 \pmod{1,674,292}$$
$$= 586,542.$$

For our attack algorithm, suppose that d_2' and d_3', in addition to e_2, e_3, and N, are given. We now show that we can recover an equivalent secret key $\overline{d_1'}$ which is congruent to d_1' modulo $\phi(N)$. By executing Algorithm 1, we first calculate

$$e_2e_3d_2' = 4,651 \times 5,003 \times 131,034,$$
$$= 3,049,023,987,402,$$
$$e_2e_3d_3' = 4,651 \times 5,003 \times 586,542$$
$$= 13,648,218,230,526$$

and

$$phi = e_2e_3d_2' - e_2e_3d_3' + (e_2 - e_3)$$
$$= 3,049,023,987,402 - 13,648,218,230,526$$
$$+ (4,651 - 5,003)$$
$$= -10,599,194,243,476.$$

Since phi is negative, we replace phi by

$$-phi = 10,599,194,243,476.$$

We confirm that phi is a multiple of $\phi(N)$ from

$$phi = 6,330,553 \times \phi(N).$$

Next, we execute Algorithm 2. We check if $g_1 = \gcd(e_1, phi) = 1$, $g_2 = \gcd(e_r, phi) = 1$, and $g_3 = \gcd(t, phi) = 1$. If any of three does not hold, we divide phi by phi/g_i for $g_i \neq 1$ until all three relations hold. In our example, $g_1 = \gcd(e_1, phi) = 1$ and $g_2 = \gcd(e_r, phi)$, but $\gcd(t, phi) = 19$. So, we divide phi by 19 and set the result as a new phi:

$$phi = 557,852,328,604.$$

This new phi satisfies $\gcd(t, phi) = 1$ as well.

Then, we calculate

$$\overline{r} = (e_r \times t)^{-1} \pmod{phi}$$
$$= (1,450,513 \times 19)^{-1} \pmod{557,852,328,604}$$
$$= 27,559,747^{-1} \pmod{557,852,328,604}$$
$$= 150,269,389,211$$

and

$$\overline{d_1} = e_1^{-1} \pmod{phi}$$
$$= 3,271^{-1} \pmod{557,852,328,604}$$
$$= 519,820,818,583.$$

Therefore, $\overline{d_1'}$ is

$$\overline{d_1'} = \overline{d_1} + \overline{r} \pmod{phi}$$
$$= 519,820,818,583 + 150,269,389,211 \pmod{phi}$$
$$= 670,090,207,794 \pmod{phi}$$
$$= 112,237,879,190.$$

Finally, we investigate the correctness of our attack in the above example by checking that the following relation holds:

$$\overline{d_1'} \pmod{\phi(N)} = 112,237,879,190 \pmod{1,674,292}$$
$$= 40,678 = d_1'.$$

In fact, we additionally demonstrate that

$$\overline{d_1} \pmod{\phi(N)} = 519,820,818,583 \pmod{1,674,292}$$
$$= 32,579 = d_1$$

and

$$\overline{r} \pmod{\phi(N)} = 150,269,389,211 \pmod{1,674,292}$$
$$= 7,919 = r.$$

We omit an example for Algorithm 3 because its correctness trivially comes from that of Chait et al.'s scheme. In summary, from the example given in this subsection, we confirmed the correctness of our attack algorithm.

V. EXPERIMENTAL ANALYSIS

In this section, we provide experimental results of our attack algorithm. The source code related to our implementation in this section is publicly available at GitHub repository.[1]

A. EXPERIMENTAL ENVIRONMENTS

Our source code was written in C++. The OpenSSL library [24] of version 3.0.10 was used for integer and modular operations with large numbers, and cryptographic hash function, SHA-256. We have tested on the machine running Ubuntu 20.04.6 LTS on virtual machine which was installed on Windows Server 2019 Standard with Intel(R) Xeon(R) Silver 4215R processor at 3.20 GHz with 128 GB of RAM.

B. EXPERIMENTAL RESULTS

Now, we present experimental results of our attack algorithm. We conducted several types of experiments with respect to the bit size of N, the value of t, and the bit size of user's public key e_i. All experiments were executed 100 times and all numbers in the tables are averages of 100 trials each.

Table 1 shows execution times of our attack algorithm for various sizes of RSA modulus N and users' public keys when the number t of individual signatures required for aggregation is fixed. We set t to 13 for all experiments in Table 1. The bit sizes of RSA modulus N, which is a product of two safe primes, were set to 1024, 2048, and 3072. For sizes of user's public keys, we consider two cases: (1) each e_i is selected randomly from $\mathbb{Z}^*_{\phi(N)}$ as the original description of Chait et al.'s scheme and (2) each e_i is selected randomly from the set of 40-bit integers which are relatively prime with

[1] https://github.com/CRYPTO-REPO/Attacks-on-Chait-s-scheme

$\phi(N)$ by reflecting the case that uses small public keys to improve the efficiency in practice. Table 1 presents execution times of Algorithm 1, Algorithm 2, and Algorithm 3 with Ind and Agg as input *mode*. It demonstrates that the execution time of our attack algorithm increases as the bit size of N increases. Since each algorithm performs operations with e_i's, the running times of algorithms depend on the size of e_i, and our algorithms are slightly faster when e_i's are small. But, for all cases our attack algorithm takes less than 21 ms only to recover an equivalent secret key of the target user by running Algorithm 1 and Algorithm 2, and takes less than 1 second only to forge an aggregate signature by running Algorithm 2 with input $mode = \mathsf{Agg}$.

TABLE 1. Execution time of our attack algorithm with respect to the bit sizes of N and e_i when $t = 13$ (unit: *ms*).

e_i: random in $\mathbb{Z}_N$			
Bit Size of N	Alg 1 (μs)	Alg 2	Alg 3 - Ind
			Alg 3 - Agg
1024	2.422	2.772	3.500
			6.466
2048	6.747	9.806	14.564
			34.063
3072	13.971	20.293	35.441
			96.842
e_i: 40-bit random			
Bit Size of N	Alg 1 (μs)	Alg 2	Alg 3 - Ind
			Alg 3 - Agg
1024	0.602	0.711	0.984
			2.439
2048	0.870	1.985	3.713
			23.481
3072	1.042	3.615	8.871
			70.982

While other algorithms do not depend on the value of t, Algorithm 3 with input $mode = \mathsf{Agg}$ depends on the value of t since it generates an aggregate signature of t individual signatures. Table 2 presents execution times of algorithms for various $t = 29, 109, 251, 503, 1009$ and two cases of e_i as in Table 1. It shows that the running time of Algorithm 3 with input $mode = \mathsf{Agg}$ increases as t increases, but those of all algorithms does not depend on the value of t.

Finally, to make up for our theoretical analysis on the number of required gcd computations in Algorithm 2, we measured the number of gcd computations through Steps 1–6 of Algorithm 2 in our experiments. In Section IV-B, we roughly stated that the number of required gcd computations is bounded above by $2 \log N$. Table 3 presents average numbers of gcd computations at Steps 1–2, Steps 3–4, and Steps 5–6, respectively, in Algorithm 2 with various t and two sizes of e_i as in Table 1 when N is 3072 bits. It shows that in fact the number of required gcd computations is less than 4 for all cases.

TABLE 2. Execution time of our attack algorithm with respect to the value of t and the bit size of e_i when N is 3072 bits (unit: *ms*).

e_i: random in $\mathbb{Z}_N$			
t	Alg 1 (μs)	Alg 2	Alg 3-Ind
			Alg 3-Agg
29	13.909	19.453	34.501
			179.139
109	14.152	20.495	36.099
			609.463
251	14.811	20.913	36.283
			1328.486
503	14.617	19.653	34.987
			2593.659
1009	14.830	20.831	36.187
			5180.115
e_i: 40-bit random			
t	Alg 1 (μs)	Alg 2	Alg 3- Ind
			Alg 3 - Agg
29	0.890	3.581	8.798
			154.648
109	1.107	3.613	8.850
			562.025
251	1.091	3.588	8.957
			1300.306
503	1.227	3.581	8.996
			2597.220
1009	1.253	3.524	8.726
			5165.971

TABLE 3. The average number of gcd computations with respect to the value of t and the bit size of e_i when N is 3,072 bits.

e_i: random in $\mathbb{Z}_N$			
t	Steps 1–2	Steps 3–4	Steps 5–6
29	1.15	1.22	1.01
109	1.14	1.15	1.04
251	1.23	1.12	1.00
503	1.26	1.16	1.00
1009	1.36	1.13	1.00
e_i: 40-bit random			
t	Steps 1–2	Steps 3–4	Steps 5–6
29	1.25	1.14	1.03
109	1.09	1.26	1.00
251	1.28	1.22	1.00
503	1.15	1.19	1.01
1009	1.25	1.17	1.00

VI. CONCLUSION

In this paper, we assessed the security of the recent RSA-based aggregate signature scheme proposed by Chait et al. [23] which follows the paradigm of the key generation algorithm of the original RSA cryptosystem [22] under only one RSA modulus for key generation of users in the system. For this purpose, we presented an attack algorithm that recovers an equivalent secret key of the target user

and then forges an individual or aggregate signature of the target user with the obtained equivalent secret key. Our attack algorithm is critical since it can recover an equivalent secret key of any users in the system with relatively cheap operations, e.g., several integer operations and modular operations, when secret keys of two users are given. We showed the efficiency of our attack algorithm by presenting experimental results. According to the experimental results, for the parameters of 128-bit security our algorithm forges an aggregate signature of 29 individual signatures in 1 second.

Since our attack works when secret keys of two users are given, Chait et al.'s scheme is no longer attractive as an aggregate signature. Furthermore, fixing it with minor modifications seems challenging due to the inherent leakage of information about the factorization of the RSA modulus when using key pairs generated by the original RSA cryptosystem, even with some modifications. We leave it as an open problem to design an RSA-based aggregate signature scheme under the common RSA modulus while adhering to the key generation paradigm of the original RSA for user key generation.

REFERENCES

[1] J. H. Ahn, M. Green, and S. Hohenberger, "Synchronized aggregate signatures: New definitions, constructions and applications," in *Proc. ACM CCS*, E. Al-Shaer, A. D. Keromytis, and V. Shmatikov, Eds., 2010, pp. 473–484.

[2] J. Cui, J. Zhang, H. Zhong, R. Shi, and Y. Xu, "An efficient certificateless aggregate signature without pairings for vehicular ad hoc networks," *Inf. Sci.*, vols. 451–452, pp. 1–15, Jul. 2018.

[3] N. Eltayieb, R. Elhabob, M. U. Aftab, R. Kuleev, M. Mazzara, and M. Ahmad, "Secure aggregate signature scheme for smart city applications," *Comput. Commun.*, vol. 193, pp. 388–395, Sep. 2022.

[4] A. Saxena, J. Misra, and A. Dhar, "Increasing anonymity in Bitcoin," in *Financial Cryptography and Data Security* (Lecture Notes in Computer Science), vol. 8438, R. Böhme, M. Brenner, T. Moore, and M. Smith, Eds. Cham, Switzerland: Springer, 2014, pp. 122–139.

[5] Y. Zhao, "Practical aggregate signature from general elliptic curves, and applications to blockchain," in *Proc. AsiaCCS*, S. D. Galbraith, G. Russello, W. Susilo, D. Gollmann, E. Kirda, and Z. Liang, Eds., 2019, pp. 529–538.

[6] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, "Aggregate and verifiably encrypted signatures from bilinear maps," in *Advances in Cryptology—EUROCRYPT 2003* (Lecture Notes in Computer Science), vol. 2656, E. Biham, Ed. Cham, Switzerland: Springer, 2003, pp. 416–432.

[7] S. Lu, R. Ostrovsky, A. Sahai, H. Shacham, and B. Waters, "Sequential aggregate signatures and multisignatures without random Oracles," in *Advances in Cryptology—EUROCRYPT 2006* (Lecture Notes in Computer Science), vol. 4004, S. Vaudenay, Ed. Cham, Switzerland: Springer, 2006, pp. 465–485.

[8] C. Gentry and Z. Ramzan, "Identity-based aggregate signatures," in *Proc. PKC* (Lecture Notes in Computer Science), vol. 3958, M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, Eds. Cham, Switzerland: Springer, 2006, pp. 257–273.

[9] A. Boldyreva, C. Gentry, A. O'Neill, and D. H. Yum, "Ordered multisignatures and identity-based sequential aggregate signatures, with applications to secure routing," in *Proc. ACM CCS*, P. Ning, S. D. C. di Vimercati, and P. F. Syverson, Eds., 2007, pp. 276–285.

[10] M. Bellare, C. Namprempre, and G. Neven, "Unrestricted aggregate signatures," in *Proc. ICALP*, in Lecture Notes in Computer Science, vol. 4596, L. Arge, C. Cachin, T. Jurdzinski, and A. Tarlecki, Eds. Cham, Switzerland: Springer, 2007, pp. 411–422.

[11] D. Ma and G. Tsudik, "Extended abstract: Forward-secure sequential aggregate authentication," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2007, pp. 86–91.

[12] M. Rückert and D. Schröder, "Aggregate and verifiably encrypted signatures from multilinear maps without random oracles," in *Proc. ISA*, in Lecture Notes in Computer Science, vol. 5576, J. H. Park, H. Chen, M. Atiquzzaman, C. Lee, T. Kim, and S. Yeo, Eds. Cham, Switzerland: Springer, 2009, pp. 750–759.

[13] S. Hohenberger, A. Sahai, and B. Waters, "Full domain hash from (leveled) multilinear maps and identity-based aggregate signatures," in *Proc. CRYPTO*, in Lecture Notes in Computer Science, vol. 8042, R. Canetti and J. A. Garay, Eds. Cham, Switzerland: Springer, 2013, pp. 494–512.

[14] S. Hohenberger, V. Koppula, and B. Waters, "Universal signature aggregators," in *Proc. EUROCRYPT*, in Lecture Notes in Computer Science, vol. 9057, E. Oswald and M. Fischlin, Eds. Cham, Switzerland: Springer, 2015, pp. 3–34.

[15] K. Boudgoust and A. Roux-Langlois, "Overfull: Too large aggregate signatures based on lattices," *Comput. J.*, vol. 2023, Mar. 2023, Art. no. bxad013.

[16] K. Boudgoust and A. Takahashi, "Sequential half-aggregation of lattice-based signatures," in *Proc. ESORICS*, in Lecture Notes in Computer Science, vol. 14434, G. Tsudik, M. Conti, K. Liang, and G. Smaragdakis, Eds. Cham, Switzerland: Springer, 2023, pp. 270–289.

[17] B. Dou, L. Xu, X. Yu, L. Mei, and C. Zuo, "Code-based sequential aggregate signature scheme," *Comput., Mater. Continua*, vol. 73, no. 3, pp. 5219–5231, 2022.

[18] A. Lysyanskaya, S. Micali, L. Reyzin, and H. Shacham, "Sequential aggregate signatures from trapdoor permutations," in *Proc. EUROCRYPT*, in Lecture Notes in Computer Science, vol. 3027, C. Cachin and J. Camenisch, Eds. Cham, Switzerland: Springer, 2004, pp. 74–90.

[19] S. S. D. Selvi, S. S. Vivek, and C. P. Rangan, "Deterministic identity based signature scheme and its application for aggregate signatures," in *Proc. ACISP*, in Lecture Notes in Computer Science, vol. 7372, W. Susilo, Y. Mu, and J. Seberry, Eds. Cham, Switzerland: Springer, 2012, pp. 280–293.

[20] X. Guo and Z. Wang, "An efficient synchronized aggregate signature scheme from standard RSA assumption," *Int. J. Future Gener. Commun. Netw.*, vol. 7, no. 3, pp. 229–240, Jun. 2014.

[21] S. Hohenberger and B. Waters, "Synchronized aggregate signatures from the RSA assumption," in *Proc. EUROCRYPT*, in Lecture Notes in Computer Science, vol. 10821, J. B. Nielsen and V. Rijmen, Eds. Cham, Switzerland: Springer, 2018, pp. 197–229.

[22] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 26, no. 1, pp. 96–99, Jan. 1983.

[23] K. Chait, A. Laouid, M. Kara, M. Hammoudeh, O. Aldabbas, and A. T. Al-Essa, "An enhanced threshold RSA-based aggregate signature scheme to reduce blockchain size," *IEEE Access*, vol. 11, pp. 110490–110501, 2023.

[24] (2023). *OpenSSL-Cryptography and SSL/TLS Toolkit*. [Online]. Available: https://www.openssl.org/

CHANHYEOK PARK received the B.S. degree in computer science engineering from Chung-Ang University, Republic of Korea, in 2022, where he is currently pursuing the master's degree in computer science engineering. His research interests include cryptography and information security.

SANGRAE CHO received the B.Eng. degree in computing from Imperial College London, in 1996, and the M.Sc. degree in information security from the Royal Holloway, University of London, in 1997. He started his career as a Researcher with the LG Corporate Technology Institute, in 1997, and has worked with ETRI, South Korea, as a Security Researcher for more than 15 years. During that time, he was actively involved in constructing a national PKI infrastructure project, until 2001. He is currently a Senior Researcher with the Authentication Research Team, ETRI. From 2004, he did several projects relating to digital identity management, including SAML v2.0 and authentication technology based on fast identity online (FIDO) specifications.

YOUNG-SEOB CHO received the Ph.D. degree in computer science from Inha University, South Korea. Since 1998, he has been working on research projects with the Electronics and Telecommunications Research Institute (ETRI) and continuously conducted numerous projects on national and international levels. He is currently a Principal Researcher with the Information Security Research Division, ETRI. His current research interests include multiparty computation, blockchain identity management, and AI security.

SOOHYUNG KIM received the B.S. and M.S. degrees in computer science from Yonsei University, Seoul, South Korea, in 1996 and 1998, respectively, and the Ph.D. degree in computer science from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 2016. He is currently a Project Leader of the Information Security Research Division, Electronics and Telecommunications Research Institute (ETRI), Daejeon. His research interests include biometrics, identity management, payment systems, and network and system security.

HYUNG TAE LEE received the B.Sc., M.Sc., and Ph.D. degrees in mathematics from Seoul National University, Republic of Korea, in 2006, 2008, and 2013, respectively. He was a Research Fellow with Nanyang Technological University, Singapore, and an Assistant Professor with Jeonbuk National University, Republic of Korea. He is currently an Assistant Professor with the School of Computer Science and Engineering, Chung-Ang University, Republic of Korea. His research interests include computational number theory, cryptography, and information security.