Half-Aggregation of Schnorr Signatures with Tight Reductions

Yanbo Chen and Yunlei Zhao(✉)

Fudan University, Shanghai, China
{ybchen,ylzhao}@fudan.edu.cn

Abstract. An aggregate signature (AS) scheme allows an unspecified aggregator to compress many signatures into a short aggregation. AS schemes can save storage costs and accelerate verification. They are desirable for applications where many signatures need to be stored, transferred, or verified together, including blockchain systems, sensor networks, certificate chains, network routing, etc. However, constructing AS schemes based on general groups, only requiring the hardness of the discrete logarithm problem, is quite tricky and has been a long-standing research question. Recently, Chalkias et al. [6] proposed a half-aggregate scheme for Schnorr signatures. We observe the scheme lacks a tight security proof and does not well support incremental aggregation, i.e., adding more signatures into a pre-existing aggregation.

This work's contributions are threefold. We first give a tight security proof for the scheme in [6] in the ROM and the algebraic group model (AGM). Second, we provide a new half-aggregate scheme for Schnorr signatures that perfectly supports incremental aggregation, whose security also tightly reduces to Schnorr's security in the AGM+ROM. Third, we present a Schnorr-based sequential aggregate signature (SAS) scheme that is tightly secure as Schnorr signature scheme in the ROM (without the AGM). Our work may pave the way for applying Schnorr aggregation in real-world cryptographic applications.

1 Introduction

The notion of *aggregate signatures* (AS) was proposed by Boneh et al. [5]. As a type of signature scheme, an AS scheme additionally allows an aggregator to compress an arbitrary number of individual signatures into a short aggregation. One can verify the validity of all those individual signatures by verifying the aggregate signature. The signers do not need to interact, and the aggregator can be an arbitrary one. AS schemes are very useful in applications where many signatures need to be stored, transferred, or verified together. Traditional application scenarios of AS schemes include sensor networks, software authentication [1], secure logging [16], etc. They can also be applied to blockchain systems like Bitcoin, e.g., to aggregate the signatures for multiple transactions to an aggregated one for improving throughput and reducing verification time.

Lysyanskaya et al. [19] proposed a useful variant of aggregate signatures, *sequential aggregate signatures* (SAS). In an SAS scheme, the signatures can

V. Atluri et al. (Eds.): ESORICS 2022, LNCS 13555, pp. 385–404, 2022.
https://doi.org/10.1007/978-3-031-17146-8_19

only be compressed sequentially. Specifically, a signer additionally gets a pre-existing aggregation as its input and directly produces a new aggregation based on the pre-existing one. Unlike traditional AS schemes, the signature aggregation cannot be made publicly by anyone but the signers involved in the SAS scheme. SAS schemes are suitable for applications like certificate chains, routing protocols, and secure logging. In these scenarios, the signatures are produced and passed in order, and a signer always knows the previous aggregation. For example, in a hierarchical public key infrastructure, a certificate on a user's public key consists of a chain of certificates issued by multiple certification authorities (CAs). Each CA certifies the CA at the next level, and the deepest CA directly certifies the user.

Boneh et al. pointed out in their seminal work [5] that the aggregation can be *incrementally* performed in their scheme. That is, the aggregator can add individual signatures into a pre-existing aggregation. In this work, we refer to AS schemes with such a feature as *incremental aggregate signature* (IAS) schemes.

In SAS schemes, the aggregation is naturally performed incrementally one-by-one. However, the feature of incremental aggregation is unspecified in the definition of AS schemes. Hence, AS schemes are not strictly stronger than SAS schemes, but IAS schemes can serve as both of them.

Most of the previous AS/SAS schemes are based on bilinear maps [3,5,17,18] and trapdoor permutations [19]. Some proposals work in the synchronized model [1,11]. Constructing AS/SAS schemes based on general groups, only requiring the hardness of discrete logarithm problem (DLP), is quite tricky and is a long-standing question.

Recently, Chalkias et al. [6] provided an aggregate scheme for Schnorr signatures. We refer to their scheme as ASchnorr and Schnorr signature scheme as Schnorr for presentation simplicity. ASchnorr achieves "half-aggregation" rather than "full-aggregation", i.e., the total size is compressed a half, rather than to a constant size. The authors provided some evidence of the impossibility of fully aggregating Schnorr signatures. Anyhow, half-aggregation of Schnorr signatures still significantly reduces the storage, so it is very useful and timely as Schnorr signature was enforced in Bitcoin (and many other blockchain systems) in November of 2021 with the Taproot update [24].

We observe two problems of ASchnorr. In [6], ASchnorr's security is reduced to Schnorr's security in the random oracle model (ROM), and hence can be further reduced to the hardness of DLP. The first problem is that the reduction has a quadratic loss, due to the reliance on rewinding. The authors suggested ignoring the quadratic loss when setting parameters for ASchnorr in practice, just as people do for Schnorr. But for deploying ASchnorr in reality, particularly in cryptocurrency systems like Bitcoin, we may want more confidence in its security. They also designed another aggregate scheme, referred to as TightASchnorr. TightASchnorr permits a tight security reduction in the ROM but is relatively expensive in both space and time. Specifically, it achieves (half+ϵ)-aggregation rather than half-aggregation, where $\epsilon = \mathcal{O}(\lambda/\log\lambda)$ with λ as the security parameter. It also has a costly aggregating procedure and makes the verification

slower than verifying the batch of individual signatures one by one. Whether there exists half-aggregate schemes for Schnorr signatures that is tightly secure as Schnorr is a fundamental question to explore.

Second, ASchnorr does not support incremental aggregation well. In particular, it suffers from ambiguity and redundant operations. By "ambiguity" we mean the verifier cannot correctly verify an aggregation without knowing how it was produced (namely, whether and when incremental aggregation happened). Without the feature of incremental aggregation, only when all the signatures are received the aggregator can start the aggregation. In reality, particularly in asynchronous distributed systems, signatures are usually not produced and transferred at the same time. It is more convenient to perform the aggregation with part of the signatures and incrementally aggregate the others when they arrive. Incremental aggregation is especially important for fault tolerance in asynchronous systems. In such applications, there may exist both faulty nodes and delayed honest nodes. An aggregator should not assume that every node will eventually provide a valid signature. It should start aggregating when it receives some signatures rather than keep waiting, but later it may need to add delayed signatures to the aggregation. Moreover, non-incremental AS schemes cannot serve as SAS schemes, so they may not applicable in scenarios like certificate chains and network routing. Hence, incremental aggregation is crucial for practical use. Many schemes based on bilinear maps naturally support incremental aggregation, so the property is rarely mentioned explicitly in the previous works. However, this is not the case for Schnorr signature aggregation.

1.1 Contributions

The contributions of this work are threefold. For the first problem of ASchnorr about security tightness, we further justify its security. We reduce the security of ASchnorr to the security of Schnorr with a tight bound in the ROM and the algebraic group model (AGM) [9]. The AGM is similar to while weaker than the generic group model (GGM) [20,23]. In the AGM, we only consider adversaries as algebraic algorithms. This is reasonable, for no attack is so far known to be significantly more efficient than such algorithms on elliptic curve groups. The AGM is widely applied in security proofs for cryptographic schemes, including blind signatures [10] and multi-signatures [2].

For the second problem about incremental aggregation, our solution is a new half-aggregation scheme, referred to as IASchnorr. IASchnorr perfectly supports incremental signature aggregation. It no more suffers from ambiguity and redundant operations. It also permits a tight security reduction in the AGM+ROM.

Our third contribution is an SAS scheme, referred to as SASchnorr. We tightly reduce its security to Schnorr's security in the ROM (without the AGM). Unlike tightASchnorr proposed in [6], our scheme SASchnorr achieves half-aggregation, and it does not increase the verification time. On one hand, SASchnorr is the first to achieve half-aggregation of Schnorr signatures with a tight security proof in the ROM. On the other hand, as ASchnorr cannot serve as an SAS scheme while keeping secure in the ROM, SASchnorr is also the first to achieve sequential

half-aggregation of Schnorr signatures with an (even non-tight) security proof in the ROM.

Tight security analysis of ASchnorr and the construction of IASchnorr pave the way for applying Schnorr aggregation in distributed ledger systems like Bitcoin, and IASchnorr may be best applicable in these application scenarios. SASchnorr may not be appropriate directly in so many scenarios as IASchnorr, but it is useful in many other applications like certificate chains, network routing, and secure logging. It achieves (sequential) half-aggregation of Schnorr signatures with a tight security reduction in the ROM, which is of theoretical interest. Meanwhile, getting rid of loose security bounds and the AGM could also be desirable for real-world cryptography.

2 Preliminaries

2.1 Aggregate Signatures

An aggregate signature (AS) scheme AS consists of five algorithms KGen, Sign, Vf, Agg, and AggVf. The first three algorithms KGen, Sign, and Vf constitute a traditional signature scheme. The signature scheme must be complete for AS to be complete. Algorithm Agg takes as inputs an arbitrary number of signatures σ_1, ..., σ_n, corresponding messages m_1, ..., m_n and public keys pk_1, ..., pk_n and outputs an aggregate signature $\tilde{\sigma}$. Algorithm AggVf takes as inputs an aggregate signature $\tilde{\sigma}$, messages m_1, ..., m_n, and public keys pk_1, ..., pk_n and outputs 0 or 1, representing $\tilde{\sigma}$ is valid or not. The completeness requirement here is that: if some signatures are correctly generated with Sign, then their aggregation must be verified as valid on/under corresponding messages/public keys.

An incremental aggregate signature (IAS) scheme is an AS scheme that additionally contains an algorithm IncrAgg. Algorithm IncrAgg takes as inputs an existing aggregate signature $\tilde{\sigma}$, corresponding messages $m_1, \ldots, m_n$ and public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$, an arbitrary number of individual signatures $\sigma_{n+1}, \ldots, \sigma_{n'}$, and corresponding messages $m_{n+1}, \ldots, m_{n'}$ and public keys $\mathsf{pk}_{n+1}, \ldots, \mathsf{pk}_{n'}$ and outputs a new aggregation $\tilde{\sigma}'$. The completeness requirement here is that: if some signatures are correctly generated with Sign, then their aggregation, no matter how they are aggregated (incrementally or not), must be verified as valid on/under corresponding messages/public keys.

Security. Boneh et al. [5] defined the existential unforgeability under chosen-message attacks (EUF-CMA) [12] of AS schemes in the *aggregate chosen-key model*. We abbreviate the EUF-CMA security in this model as CK-AEUF-CMA. The CK-AEUF-CMA game consists of three stages defined as follows:

Setup. The forger $\mathcal{F}$ is given a public key pk^* generated by KGen.

Queries. The forger $\mathcal{F}$ has access to a signing oracle. It can adaptively requests signatures under pk^* on messages of its choice.

Response. The forger $\mathcal{F}$ outputs an arbitrary number n of public keys pk_1, ..., pk_n, n messages m_1, ..., m_n, and an aggregate signature $\tilde{\sigma}$.

We say $\mathcal{F}$ wins this game if $\tilde{\sigma}$ is a valid aggregate signature on m_1, ..., m_n under pk_1, ..., pk_n, there exists $k \in \{1, \ldots, n\}$ such that $\mathsf{pk}_k = \mathsf{pk}^*$, and $\mathcal{F}$ has not queried m_k to the signing oracle.

In this work, we only consider the security in the ROM, and the security results for AS schemes in this work are independent of the maximum number of aggregated signatures. We say a forger $\mathcal{F}$ $(t, q_{\mathsf{H}_1}, \ldots, q_{\mathsf{H}_l}, q_{\mathsf{S}}, \varepsilon)$-breaks the CK-AEUF-CMA security of an AS scheme AS in the ROM if: $\mathcal{F}$ runs in time at most t; $\mathcal{F}$ makes at most q_{H_1}, ..., q_{H_l} queries respectively to the random oracles H_1, ..., H_l modeling the hash functions used in AS; $\mathcal{F}$ makes at most q_{S} queries to the signing oracle; and $\mathcal{F}$ wins the CK-AEUF-CMA game with probability at least ε.

2.2 Sequential Aggregate Signatures

A sequential aggregate signature (SAS) scheme SAS consists of three algorithm KGen, $\mathsf{SeqSign}$, and Vf. Algorithms KGen and Vf are the same as the ones in a normal signature scheme. Algorithm $\mathsf{SeqSign}$ takes an existing aggregate signature $\tilde{\sigma}_{n-1}$, corresponding messages m_1, ..., m_{n-1} and public keys pk_1, ..., pk_{n-1}, a secret key sk_n, and a message m_n as inputs and outputs a new aggregation $\tilde{\sigma}_n$. With $n = 1$, the behavior of $\mathsf{SeqSign}$ is the same as algorithm Sign in a normal signature scheme, and it indeed constitutes a normal scheme together with KGen and Vf. The completeness requirement is that: if a sequential aggregate signature is generated correctly by sequentially running $\mathsf{SeqSign}$ multiple times, then it must be verified as valid on/under corresponding messages/public keys.

Security. Lysyanskaya et al. [19] defined the EUF-CMA security in the *sequential aggregate chosen-key model* (CK-SAEUF-CMA) for SAS schemes. We introduce the security notion here, while we will prove the security of our scheme in a modified model which we will define later in Sect. 5.2. The three-stage CK-SAEUF-CMA game is defined as follows:

Setup. The forger $\mathcal{F}$ is given a public key pk^* generated by KGen.

Queries. The forger $\mathcal{F}$ has access to a signing oracle. It can adaptively requests signatures under pk^* on messages, existing aggregations, and previous public keys and messages of its choice.

Response. The forger $\mathcal{F}$ outputs an arbitrary number n of public keys pk_1, ..., pk_n, n messages m_1, ..., m_n, and a sequential aggregate signature $\tilde{\sigma}$.

We say $\mathcal{F}$ wins this game if $\tilde{\sigma}$ is a valid aggregate signature on m_1, ..., m_n under pk_1, ..., pk_n, there exists k such that $\mathsf{pk}_k = \mathsf{pk}^*$, and $\mathcal{F}$ has not queried m_k together with previous public keys and messages $\{(\mathsf{pk}_1, m_1), \ldots, (\mathsf{pk}_{k-1}, m_{k-1})\}$. Note it is allowed to query m_k with another set of previous public keys and messages.

We say a forger $\mathcal{F}$ $(t, q_{\mathsf{H}_1}, \ldots, q_{\mathsf{H}_l}, q_{\mathsf{S}}, N, \varepsilon)$-breaks the CK-SAEUF-CMA security of an SAS scheme SAS in the ROM if: $\mathcal{F}$ runs in time at most t; $\mathcal{F}$ makes at

$\mathsf{KGen}()$	$\mathsf{Sign}(\mathsf{sk}, m)$	$\mathsf{Vf}(\mathsf{pk}, m, \sigma)$
$x \leftarrow\$ \mathbb{Z}_p$	$x := \mathsf{sk}; X := g^x$	$X := \mathsf{pk}$
$X := g^x$	$r \leftarrow\$ \mathbb{Z}_p; R := g^r$	$(c, s) := \sigma$
$\mathsf{sk} := x$	$c := \mathsf{H}_1(R, m)$	$R := g^s / X^c$
$\mathsf{pk} := X$	$s := r + cx$	**return** $[\![\mathsf{H}_1(R, m) = c]\!]$
return $(\mathsf{sk}, \mathsf{pk})$	**return** $\sigma := (c, s)$	

Fig. 1. Description of Schnorr signatures. The cyclic group $\mathbb{G}$, of order p with a generator g and the hash functions H_1 are scheme-level parameters.

most $q_{\mathsf{H}_1}, \ldots, q_{\mathsf{H}_l}$ queries respectively to the random oracles $\mathsf{H}_1, \ldots, \mathsf{H}_l$ modeling the hash functions used in SAS; $\mathcal{F}$ makes at most q_{S} queries to the signing oracle; $\mathcal{F}$ gives a forged sequential aggregate signature of length at most N; and $\mathcal{F}$ wins the CK-SAEUF-CMA game with probability at least ε.

2.3 Algebraic Group Model

The algebraic group model (AGM) is an ideal model proposed in [9]. In the AGM, we require the adversary to provide the representations of any group elements it outputs as a product of the elements it received. The AGM lies between the generic group model (GGM) [20,23] and the realistic world. While the GGM is useful for proving information-theoretical bounds, the AGM is useful for making reductions.

To be more specific, consider a multiplicative group. Let $X_1, \ldots, X_n$ be group elements provided to the adversary as inputs or from oracles. For any group element Y it outputs or queries to oracles, it also gives a representation of Y, i.e., a vector $(\alpha_1, \ldots, \alpha_n)$ satisfying that $Y = \prod_{i=1}^{n} X_i^{\alpha_i}$.

2.4 Schnorr Signatures

In Fig. 1, we present Schnorr signature scheme in its traditional (c, s)-format [22], while nowadays it is also common to deploy its (R, s) variant on elliptic groups. For instance, the (R, s) version of Schnorr signature scheme was standardized as EdDSA [4]. Bitcoin also chose the (R, s) version [24].

Schnorr signature in the (c, s)-format is more compact than its (R, s)-format over integer groups, but the difference is relatively small over elliptic curve groups [14]. Practical elliptic curves of order p with $\log p = 2\lambda$ can offer approximately λ bits of security, and the points over these groups can be represented by about 2λ bits (to be precise, $2\lambda + 1$ bits). In practice, it is safer to use a hash function H_1 with 2λ-bit outputs for λ-bit security, which avoids some subtle fragile caused by shorter hashes. For example, such a full-length hash function can computationally bind the signature to the corresponding message, which is increasingly important for applications like blockchain [6]. In short, the point R, the scalar s, and the hash value c all have a size of about 2λ bits.

$\mathsf{Agg}(\{(\mathsf{pk}_1, m_1, \sigma_1), \ldots, (\mathsf{pk}_n, m_n, \sigma_n)\})$

for $i = 1, \ldots, n$ **do**
  $X_i := \mathsf{pk}_i$
  $(c_i, s_i) := \sigma_i$
  $R_i := g^{s_i}/X^{c_i}$
$L := \{(R_1, X_1, m_1), \ldots, (R_n, X_n, m_n)\}$
for $i = 1, \ldots, n$ **do**
  $a_i := \mathsf{H}_2(L, i)$
$\tilde{s} := \sum_{i=1}^{n} a_i s_i$
return $\tilde{\sigma} := (\{R_1, \ldots, R_n\}, \tilde{s})$

$\mathsf{AggVf}(\{(\mathsf{pk}_1, m_1), \ldots, (\mathsf{pk}_n, m_n)\}, \tilde{\sigma})$

for $i = 1, \ldots, n$ **do**
  $X_i := \mathsf{pk}_i$
$(\{R_1, \ldots, R_n\}, \tilde{s}) := \tilde{\sigma}$
$L := \{(R_1, X_1, m_1), \ldots, (R_n, X_n, m_n)\}$
for $i = 1, \ldots, n$ **do**
  $c_i := \mathsf{H}_1(R_i, m_i)$
  $a_i := \mathsf{H}_2(L, i)$
return $[\![g^{\tilde{s}} = \prod_{i=1}^{n} (R_i X_i^{c_i})^{a_i}]\!]$

Fig. 2. Description of ASchnorr. The cyclic group $\mathbb{G}$, of order p with a generator g, and the hash functions H_1 and H_2 are scheme-level parameters. The range of H_2 is denoted by $\mathcal{H}_2$.

3 Half-Aggregation of Schnorr Signatures, Revisited

3.1 Scheme Description

In this section, we analyze the security of ASchnorr, the half-aggregate scheme for Schnorr signatures in [6], in the AGM+ROM. Figure 2 describes the scheme, with a slight difference that we consider individual signatures in the (c, s)-format, as presented in Fig. 1. H_1 and H_2 are hash functions to $\mathbb{Z}_p$, and we use $\mathcal{H}_2$ to denote the range of H_2.

On signatures (c_1, s_1), ..., (c_n, s_n), respectively on messages m_1, ..., m_n under public keys $X_1, \ldots, X_n$, algorithm Agg recovers $R_1, \ldots, R_n$. Then it computes n coefficients a_1, ..., a_n and aggregates the responses into $\tilde{s} = \sum_{i=1}^{n} a_i s_i$. In the aggregate signature, R_1, ..., R_n replace c_1, ..., c_n. To verify an aggregate signature, algorithm Vf also computes these coefficients and checks whether $g^{\tilde{s}} = \prod_{i=1}^{n} (R_i X_i^{c_i})^{a_i}$, where $c_i = \mathsf{H}_1(R_i, m_i)$.

The scheme certainly works well with signatures in the (R, s)-format. Actually, the two formats are mathematically equivalent, and the scheme essentially recovers R and aggregates the (R, s) signatures.

Let λ be the security parameter. As discussed in Sect. 2.4, we consider the case that R, c, and s are all approximately 2λ-bit long. Then n individual signatures are about $2n \cdot 2\lambda$ bits of total length, while the aggregation of these signatures is only about $(n + 1) \cdot 2\lambda$ bits of length as discussed above. In this case, ASchnorr compresses the signatures to roughly half the original size. For instance, consider the widely applied curve secp256k1 and the hash function SHA256 for about 128 bits of security. The order p of secp256k1 is of 32 bytes. A point over secp256k1 is usually represented by 32 bytes (represent the x-ordinate) plus one more bit (indicating the sign of its y-coordinate). SHA256 gives 32-byte outputs.

Then $\mathsf{ASchnorr}$ compresses n individual signatures, of $64n$ byte length, into $(32(n+1)+n/8)$ byte length.

One needs to compute $2n+1$ exponentiations to verify an aggregate signature, while it takes 2 exponentiations to verify each individual signature. Although we do not reduce the number of exponentiations, AggVf can be significantly speed up by applying the simultaneous exponentiation techniques [8,13,15]. It was estimated in [25] that verifying an aggregate signature using simultaneous exponentiation techniques is about 72% faster than sequentially verifying the individual signatures. The benchmarks in [6] also support this estimation. Note that $\mathsf{ASchnorr}$ only reduces the verification time of signatures in the (c,s)-format, because (R,s) signatures support batch verification, which is also essentially computing $2n+1$ simultaneous exponentiations.

3.2 Security in the AGM+ROM

We prove the CK-AEUF-CMA security of $\mathsf{ASchnorr}$ with a tight bound in the AGM+ROM, where the hash function H_2 is modeled as a random oracle, based on the EUF-CMA security of $\mathsf{Schnorr}$. In comparison, the bound in the ROM suffers from a quadratic loss, for the proof is based on rewinding. In the CK-AEUF-CMA game against $\mathsf{ASchnorr}$, a forger has accesses to a signing oracle $\textsc{Sign}$ and the random oracle H_2.

Theorem 1. *If there exists a forger that $(t, q_{\mathsf{H}_2}, q_{\mathsf{S}}, \varepsilon)$-breaks the CK-AEUF-CMA security of $\mathsf{ASchnorr}$ in the AGM+ROM with H_2 modeled as a random oracle, then there exists an algorithm that $(t', q_{\mathsf{S}}, \varepsilon')$-breaks the EUF-CMA security of $\mathsf{Schnorr}$ with $t' = \mathcal{O}(t)$ and $\varepsilon' \geq \varepsilon - (q_{\mathsf{H}_2}+1)/|\mathcal{H}_2|$.*

Proof. Suppose that $\mathcal{F}$ is the forger that $(t, q_{\mathsf{H}_2}, q_{\mathsf{S}}, \varepsilon)$-breaks the CK-AEUF-CMA security of $\mathsf{ASchnorr}$. We construct an algorithm $\mathcal{A}$ to break the EUF-CMA security of $\mathsf{Schnorr}$. On the target public key X^*, $\mathcal{A}$ first initializes an empty table $T[\cdot,\cdot]$ for simulating random oracle H_2. After that, it runs $\mathcal{F}$ with X^* as the target public key. Algorithm $\mathcal{A}$ handles queries from $\mathcal{F}$ as follows:

- Signing queries. On query $\textsc{Sign}(m)$ from $\mathcal{F}$, $\mathcal{A}$ queries m to its own signing oracle in the EUF-CMA game. It receives a signature (c,s) and returns it to $\mathcal{F}$.
- H_2 queries. On query $\mathsf{H}_2(L,k)$ from $\mathcal{F}$, $\mathcal{A}$ assigns $T[L,k] \leftarrow\!\!\$\, \mathcal{H}_2$ if $T[L,k]$ is undefined and then returns $T[L,k]$ to $\mathcal{F}$.

As the AGM requires, whenever $\mathcal{F}$ queries or outputs a group element, it should also provide the representation of the element as a product of those elements given to it, i.e., the generator g and the target public key X^*. We assume $\mathcal{A}$ never gets two different representations of the same element, otherwise it can directly compute the discrete logarithm of X^*.

At last, $\mathcal{F}$ outputs a forged aggregate signature $(\{R_1,\ldots,R_n\},\tilde{s})$ together with the corresponding messages $m_1, \ldots, m_n$ and public keys $X_1, \ldots, X_n$ it

chooses. $\mathcal{F}$ also outputs a representation of each group element. Precisely, $\mathcal{A}$ gets $2n$ pairs $(\alpha_{1,1}, \beta_{1,1}), \ldots, (\alpha_{1,n}, \beta_{1,n}), (\alpha_{2,1}, \beta_{2,1}), \ldots, (\alpha_{2,n}, \beta_{2,n})$ satisfying

$$R_i = g^{\alpha_{1,i}} X^{*\beta_{1,i}} \quad \text{and} \quad X_i = g^{\alpha_{2,i}} X^{*\beta_{2,i}}$$

for $i = 1, \ldots, n$. Let $c_i = \mathsf{H}_1(R_i, m_i)$ for $i = 1, \ldots, n$.

If there exists k such that $X_k = X^*$ and $\beta_{1,k} + c_k \beta_{2,k} = 0$, which we call Case 1, then we have

$$R_k X^{*c_k} = g^{\alpha_{1,k} + c_k \alpha_{2,k}}.$$

Thus, $\mathcal{A}$ obtains a forged Schnorr signature $(R_k, \alpha_{1,k} + c_k \alpha_{2,k})$ on m_k and wins the EUF-CMA game if m_k is fresh (i.e., has not been queried to the signing oracle).

We further consider the case that $\beta_{1,k} + c_k \beta_{2,k} \neq 0$ for all k that satisfies $X_k = X^*$, which we call Case 2. Let $L = \{(R_1, X_1, m_1), \ldots, (R_n, X_n, m_n)\}$ and $a_i = \mathsf{H}_2(L, i)$ for $i = 1, \ldots, n$. It must hold that $g^{\tilde{s}} = \prod_{i=1}^{n} (R_i X_i^{c_i})^{a_i}$ for $\mathcal{F}$ to win. Let

$$\alpha^* = \sum_{i=1}^{n} a_i (\alpha_{1,i} + c_i \alpha_{2,i}) \quad \text{and} \quad \beta^* = \sum_{i=1}^{n} a_i (\beta_{1,i} + c_i \beta_{2,i}).$$

It can be verified that $\prod_{i=1}^{n} (R_i X_i^{c_i})^{a_i} = g^{\alpha^*} X^{*\beta^*}$, and consequently $g^{\tilde{s}} = g^{\alpha^*} X^{*\beta^*}$. Therefore, $\mathcal{A}$ can extract the discrete logarithm $(\tilde{s} - \alpha^*)/\beta^*$ of X^* as long as $\beta^* \neq 0$. It can further produce signatures on any message it chooses and certainly win the EUF-CMA game.

To be exact, to compute α^* and β^*, $\mathcal{A}$ only needs to let $a_i = T[L, i]$ for every i satisfying $R_i X_i^{c_i} \neq 1_{\mathbb{G}}$, where $1_{\mathbb{G}}$ denotes the identity in $\mathbb{G}$ (otherwise a_i is irrelevant). If anyone of those items is undefined, then $\mathcal{A}$ just aborts. Here we show that $\mathcal{F}$ is almost impossible to win in such a case. Suppose $T[L, i]$ was undefined when $\mathcal{F}$ decides its forgery. We regard it as the last one to be determined. Since $R_i X_i^{c_i} \neq 1_{\mathbb{G}}$, there is at most one value of $a_i = T[L, i]$ in $\mathcal{H}_2$ can make $g^{\tilde{s}} = \prod_{i=1}^{n} (R_i X_i^{c_i})^{a_i}$. Hence, $\mathcal{F}$ wins with probability at most $1/|\mathcal{H}_2|$.

Now let us show that $\mathcal{A}$ can win the EUF-CMA game in one of the above two cases (by extracting a forged signature or directly computing the discrete logarithm of X^*) with high probability. To do so, we define an event $\mathsf{AggElim}$, explain how it relates to $\mathcal{A}$'s winning, and bound its probability.

We say $\mathsf{AggElim}$ occurs if there exists $L = \{(R_1, X_1, m_1), \ldots, (R_n, X_n, m_n)\}$ satisfying the following conditions:

- Let $\mathcal{I}$ be the set of those i satisfying $R_i X_i^{c_i} \neq 1_{\mathbb{G}}$, where $c_i = \mathsf{H}_1(R_i, m_i)$. The condition is that for each $i \in \mathcal{I}$, $T[L, i]$ has been defined.
- Let $(\alpha_{1,i}, \beta_{1,i})$ and $(\alpha_{2,i}, \beta_{2,i})$ be the representations of R_i and X_i respectively. Let $\beta_i^* = \beta_{1,i} + c_i \beta_{2,i}$. The condition is that there exists $k \in \mathcal{I}$ such that $\beta_k^* \neq 0$.
- $\sum_{i \in \mathcal{I}} T[L, i] \beta_i^* = 0$.

If $\mathcal{F}$ wins, but $\mathcal{A}$ does not go to Case 1 (and then extract a forged signature), does not abort because of undefined table items, and also does not win in Case 2

(by directly computing the discrete logarithm of X^*), then $\mathsf{AggElim}$ must happen. Conditioned on $\mathcal{F}$'s winning, there must exist k such that $X^* = X_k$ and m_k is fresh. Since $\mathcal{A}$ does not go to Case 1, it must hold that $\beta_{i,k} + c_k\beta_{2,k} \neq 0$, and hence the second condition holds. That $\mathcal{A}$ does not abort because of undefined table items implies the first condition. Recall the definition of β^*, and we can see if $\mathcal{A}$ does not win in Case 2 (only when $\beta^* = 0$), the third condition holds.

Among those $i \in \mathcal{I}$ satisfying $\beta_i^* \neq 0$, consider the last a_i to be determined. There is at most one value from $\mathcal{H}_2$ can make L satisfy the third condition. This means $\mathsf{AggElim}$ happens with probability at most $1/|\mathcal{H}_2|$ for each L occurring in table T. The total probability of $\mathsf{AggElim}$ is thus upper bounded by $q_{\mathsf{H}_2}/|\mathcal{H}_2|$.

Now we can bound the probability ε' that $\mathcal{A}$ wins the EUF-CMA game. Consider the case $\mathcal{F}$ wins. We assume $\mathcal{A}$ does not win in the first way. It then loses the game only if it aborts for undefined table items or $\beta^* = 0$, respectively bounded by $1/|\mathcal{H}_2|$ and $q_{\mathsf{H}_2}/|\mathcal{H}_2|$. Therefore, we have $\varepsilon' \geq \varepsilon - (q_{\mathsf{H}_2} + 1)/|\mathcal{H}_2|$.

It remains to bound the running time t' of $\mathcal{A}$. Except the running time t of $\mathcal{F}$, there are two significant parts of t' we need to consider: maintaining table T and handling H_2 queries and the final forgery. Assuming that a table operation takes constant time, the first part takes $\mathcal{O}(q_{\mathsf{H}_2})$ which is also $\mathcal{O}(t)$. The second part of time is $\mathcal{O}(t)$, for the forger needs to write the queries and the forgery all. In conclusion, we have $t' \leq \mathcal{O}(t)$. □

Remark 1 (Security of aggregating signatures in the (R, s)-format). Compared to signatures in the (R, s)-format, the (c, s)-format slightly simplifies the proof in the AGM, since forger $\mathcal{F}$ does not get other group elements than g and X^*. Nevertheless, it is easy to adapt our proof to (R, s) signatures. Let $(\hat{R}_j, \hat{s}_j)$ be the signature that $\mathcal{F}$ receives in the j-th signing query $\textsc{Sign}(\hat{m}_j)$. In addition to g and X^*, it can also use $\hat{R}_1, \ldots, \hat{R}_{q_\mathsf{S}}$ to represent the group elements it queries or outputs. Let $\hat{c}_j = \mathsf{H}_1(\hat{R}_j, \hat{m}_j)$. By the validity of signatures, $\hat{R}_j = g^{\hat{s}_j}/X^{*\hat{c}_j}$ for $j = 1, \ldots, q_\mathsf{S}$. Hence, since $\mathcal{A}$ knows how to represent each $\hat{R}_j$ with g and X^*, it essentially gets a representation with g and X^* of each group element that $\mathcal{F}$ queries or outputs. The rest of the proof remains.

4 Incremental Aggregation of Schnorr Signatures

4.1 Scheme Description

In the real world, it is common that we need to store more signatures after we have produced an aggregation, which leads to the demand for incremental aggregation. However, $\mathsf{ASchnorr}$ does not support incremental aggregation well. The procedure of incremental aggregating is not explicitly defined at the scheme level. We can implement it by treating a pre-existing aggregation as a normal signature, but this causes ambiguity and redundant operations. We can omit some redundant computations, but the ambiguity comes from the scheme intrinsically. Following $\mathsf{ASchnorr}$, if we aggregate n' signatures, we will compute coefficients $a_1, \ldots, a_{n'}$. If we aggregate the first n signatures among them, we will compute

$\mathsf{IncrAgg}(L, \tilde{\sigma}, L')$

// $L = \{(\mathsf{pk}_1, m_1), \ldots, (\mathsf{pk}_n, m_n)\}$
// $L' = \{(\mathsf{pk}_{n+1}, m_{n+1}, \sigma_{n+1}), \backslash\backslash$
// $\quad \ldots, (\mathsf{pk}_{n'}, m_{n'}, \sigma_{n'})\}$
for $i = 1, \ldots, n'$ **do** $X_i := \mathsf{pk}_i$
$(\{R_1, \ldots, R_n\}, \tilde{s}) := \tilde{\sigma}$
for $i = n+1, \ldots, n'$ **do**
  $(c_i, s_i) := \sigma_i$
  $R_i := g^{s_i} / X_i^{c_i}$
  $L_i := \{(R_1, X_1, m_1), \ldots, (R_i, X_i, m_i)\}$
  $a_i := \mathsf{H}_2(L_i)$
$\tilde{s}' := \tilde{s} + \sum_{i=n+1}^{n'} a_i s_i$
return $\tilde{\sigma}' := (\{R_1, \ldots, R_{n'}\}, \tilde{s}')$

$\mathsf{AggVf}(\{(\mathsf{pk}_1, m_1), \ldots, (\mathsf{pk}_n, m_n)\}, \tilde{\sigma})$

for $i = 1, \ldots, n$ **do** $X_i := \mathsf{pk}_i$
$(\{R_1, \ldots, R_n\}, \tilde{s}) := \tilde{\sigma}$
$a_1 := 1; \quad c_1 := \mathsf{H}_1(R_1, m_1)$
for $i = 2, \ldots, n$ **do**
  $c_i := \mathsf{H}_1(R_i, m_i)$
  $L_i := \{(R_1, X_1, m_1), \ldots, (R_i, X_i, m_i)\}$
  $a_i := \mathsf{H}_2(L_i)$
return $[\![g^{\tilde{s}} = \prod_{i=1}^{n} (R_i X_i^{c_i})^{a_i}]\!]$

Fig. 3. Algorithms $\mathsf{IncrAgg}$ and AggVf of $\mathsf{IASchnorr}$. The cyclic group $\mathbb{G}$, of order p with a generator g, and the hash functions H_1 and H_2 are scheme-level parameters. The range of H_2 is denoted by $\mathcal{H}_2$. The aggregate algorithm Agg is a special case of $\mathsf{IncrAgg}$.

coefficients $a'_1, \ldots, a'_n$. The scheme-level ambiguity is reflected by that usually $a_i \neq a'_i$ for $i = 1, \ldots, n$. Hence, a verifier has to know whether the first n signatures are aggregated first, or aggregated together with the others. Otherwise, it can not correctly verify the aggregation of the n' signatures.

For such a problem, we provide a modified scheme $\mathsf{IASchnorr}$, described in Fig. 3. See how we remove the ambiguity: the coefficient a_i for the i-th signature only depends on the first i signatures. As a result, whether the first n signatures are aggregated first or together with the others does not affect the value of the coefficients. The normal scheme $\mathsf{Schnorr}$ (i.e., algorithms KGen, Sign, and Vf) is unchanged, so Fig. 3 only describes algorithms $\mathsf{IncrAgg}$ and AggVf. The aggregate algorithm Agg can be seen as a special case of $\mathsf{IncrAgg}$ when $n = 0$.

4.2 Security

We can easily prove almost the same security result for $\mathsf{IASchnorr}$ as $\mathsf{ASchnorr}$ in the AGM+ROM. The proof is very similar to the one of Theorem 1, so we defer it to the full version of this paper [7].

Theorem 2. *If there exists a forger that* $(t, q_{\mathsf{H}_2}, q_{\mathsf{S}}, \varepsilon)$*-breaks the CK-AEUF-CMA security of aggregate signature scheme* $\mathsf{IASchnorr}$ *in the AGM+ROM with* H_2 *modeled as a random oracle, then there exists an algorithm that* $(t', q_{\mathsf{S}}, \varepsilon')$*-breaks the EUF-CMA security of* $\mathsf{Schnorr}$ *with* $t' = \mathcal{O}(t)$ *and* $\varepsilon' \geq \varepsilon - (q_{\mathsf{H}_2} + 1)/|\mathcal{H}_2|$.

$\mathsf{SeqSign}(L, \tilde{\sigma}_{n-1}, \mathsf{sk}_n, m_n)$	$\mathsf{Vf}(\{(\mathsf{pk}_1, m_1), \ldots, (\mathsf{pk}_n, m_n)\}, \tilde{\sigma}_n)$
$/\!/\ L = \{(\mathsf{pk}_1, m_1), \ldots, (\mathsf{pk}_{n-1}, m_{n-1})\}$	**for** $i = 1, \ldots, n$ **do** $\quad X_i := \mathsf{pk}_i$
for $i = 1, \ldots, n-1$ **do** $\quad X_i := \mathsf{pk}_i$	$(\tilde{R}_n, \{s_1, \ldots, s_n\}) := \tilde{\sigma}_n$
$(\tilde{R}_{n-1}, \{s_1, \ldots, s_{n-1}\}) := \tilde{\sigma}_{n-1}$	$c_n := \mathsf{H}(\tilde{R}_n, X_n, m_n, s_{n-1}, n)$
$x_n := \mathsf{sk}_n; \quad X_n := g^{x_n}$	**if** $n = 1$ **then** $\quad$ **return** $[\![g^{s_1} = \tilde{R}_1 X_1^{c_1}]\!]$
$r_n \leftarrow\!\$\, \mathbb{Z}_p; \quad R_n := g^{r_n}$	**else**
$\tilde{R}_n = \tilde{R}_{n-1} \cdot R_n$	$R_n := g^{s_n} / X_n^{c_n}$
$c_n := \mathsf{H}(\tilde{R}_n, X_n, m_n, s_{n-1}, n)$	$\tilde{R}_{n-1} := \tilde{R}_n / R_n$
$s_n := r_n + c_n x_n$	$\tilde{\sigma}_{n-1} := (\tilde{R}_{n-1}, \{s_1, \ldots, s_{n-1}\})$
return $\tilde{\sigma}_n := (\tilde{R}_n, \{s_1, \ldots, s_n\})$	**return** $\mathsf{Vf}(L', \tilde{\sigma}_{n-1})$
	$/\!/\ L' = \{(\mathsf{pk}_1, m_1), \ldots, (\mathsf{pk}_{n-1}, m_{n-1})\}$

Fig. 4. Description of SASchnorr. The cyclic group $\mathbb{G}$, of order p with a generator g, and the hash function H are scheme-level parameters. The range of H is denoted by $\mathcal{H}$. The key generation algorithm KGen is the same as Schnorr's, as described in Fig. 2. We define s_0 as always 0.

5 Sequential Aggregation of Schnorr Signatures with Tight Reduction in the ROM

5.1 Scheme Description

We describe SASchnorr in Fig. 4. The aggregation is implemented in a very different way in SASchnorr compared with the other schemes: we aggregate the commitment parts of the individual signatures rather than the response parts. Provided an pre-existing sequential aggregate signature $(\tilde{R}_{n-1}, \{s_1, \ldots, s_{n-1}\})$ on messages $m_1, \ldots, m_{n-1}$ under public keys $X_1, \ldots, X_{n-1}$, what the signer does in SeqSign is basically producing a normal Schnorr signature. The difference is what it hashes to get its challenge c_n. Instead of its own commitment $R_n = g^{r_n}$, it hashes the aggregate commitment $\tilde{R}_n = \tilde{R}_{n-1} \cdot R_n$. It additionally hashes its public key X_n, the response s_{n-1} from the last signer, and the current length n.

To verify an aggregate signature, the verifier sequentially recovers the individual commitments from the n-th to the first one. Provided the aggregation of j commitments $\tilde{R}_j$, the verifier can compute c_j. It then obtains R_j, the j-th individual commitment, by $R_j = g^{s_j} / X_j^{c_j}$. After that, it knows $\tilde{R}_{j-1}$ and iteratively continues the procedure.

As ASchnorr and IASchnorr do, SASchnorr achieves about half aggregation in elliptic curve groups. On the other hand, the verification in SASchnorr is similar to a sequence of individual verification. We cannot use simultaneous multiplication techniques to accelerate the verification.

Note that in Fig. 4, we minimize what the signer needs to hash. As a result, many inputs are irrelevant to the signing procedure. There are some potential

optimizations can be made in practice. For example, consider the scenario where a fixed destination is public known to all signers, and they do not care the validity of the partial aggregations. A signer can choose to not pass redundant information to the next one. Instead, the j-th signer can pass only $\tilde{R}_j$ and s_j to the next signer and directly pass X_j, m_j, and s_j to the destination. Thus, the total communication complexity is significantly reduced.

For consistency, we require the first signer also hashes s_0, which we define as 0, and the current length 1. This can be omitted without ambiguity.

5.2 A New Security Model for SAS Schemes

Rather than the security model presented in [19] and Sect. 2.2 for SAS schemes, we analyze the security of SASchnorr in a modified model. There is no essential difference between them. We just adapt the model to fit the fact that in our scheme SASchnorr, algorithm SeqSign takes fewer inputs than the general syntax of SAS schemes defined in [19] and Sect. 2.2. Hence, we still use the term CK-SAEUF-CMA to denote the security notion.

Note that the signature produced by SeqSign only depends on x_n, m_n, and part of $\tilde{\sigma}_{n-1}$, i.e., $\tilde{R}_{n-1}$ and s_{n-1}. We only take them as the arguments of the signing oracle. Precisely, the adversary can query $\textsc{Sign}(\tilde{R}_{n-1}, s_{n-1}, m_n, n)$ and receive $(\tilde{R}_n, s_n)$.

The adversary's goal is to forge an aggregation $(\tilde{R}_n, \{s_1, \ldots, s_n\})$ on/under corresponding messages/public keys $m_1, \ldots, m_n$, $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$ on its choice. The adversary is said to win if the forgery is valid, and it has not queried $\textsc{Sign}(\cdot, s_{k-1}, m_k, k)$ for some k such that $X_k = X^*$, where X^* is the target public key.

We make some comparisons between the new security model and the original model defined in [19]. On the one hand, the adversary does not need to give a valid aggregation in order to request a subsequent aggregation. Specifically, the signing oracle cannot verify the validity of the previous aggregation, as it doesn't know the corresponding public keys and messages. In this aspect, our model allows for a more powerful adversary. On the other hand, the success conditions of the adversary in our model are also adjusted according to the change of the signing oracle, which makes our model incomparable with the original one.

We underline that the reason why we introduce the new model is not that we cannot achieve security in the original one. Actually, simpler designs can already make the scheme secure in the original model. If we require each signer to verify the previous aggregation, or we let c_n be instead $\mathsf{H}(\tilde{R}_n, X_1, \ldots, X_n, m_1, \ldots, m_n)$, then our scheme can be proved secure in the original model. We introduce our new security model for SAS schemes to show the possibility of signing without knowing so much information. This feature allows essential bandwidth/storage saving.

See the full version of this paper [7] for more discussion. In the full version, we explain our model as a result of a three-step modification on the original model, among which two steps strengthen the model and one weakens it. We also explain how to prove the security of our scheme (with minor modifications as mentioned above) in the original model.

5.3 Security

We prove that the security of SASchnorr reduces to the EUF-CMA security of Schnorr in the ROM, with only an additive security loss. Note that we can directly reduce the security of SASchnorr to the DLP based on the forking lemma, but we intentionally avoid doing so. Improving the proof techniques and finding tighter bounds for Schnorr signatures in the ROM are popular research topics, and some great results were achieved in a recent work [21]. We prove a relatively modular result which is compatible with any previous or future improvements on the security results for Schnorr.

Theorem 3. *If there exists a forger that* $(t, q_{\mathsf{H}}, q_{\mathsf{S}}, N, \varepsilon)$*-breaks the CK-SAEUF-CMA security of* SASchnorr *in the ROM, then there exists an algorithm that* $(t', q_{\mathsf{H}} + q_{\mathsf{S}}, q_{\mathsf{S}}, \varepsilon')$*-breaks the EUF-CMA security of* Schnorr *in the ROM, with*

$$t' \leq t + 2Nt_{\exp} + \mathcal{O}\left(q_{\mathsf{S}} + q_{\mathsf{H}}\right)$$

and

$$\varepsilon' \geq \varepsilon - \frac{(q_{\mathsf{H}} + q_{\mathsf{S}})(q_{\mathsf{H}} + 3q_{\mathsf{S}})}{2p} - \frac{(q_{\mathsf{H}} + q_{\mathsf{S}} + 1)^2 + 1}{2|\mathcal{H}|},$$

where $t_{\exp}$ *is the time of an exponentiation in* $\mathbb{G}$.

We give some intuition before the actual proof. Let X^* be the target public key. In a valid forgery, there must exist a $k \in \{1, \ldots, n\}$ such that $X_k = X^*$, and it holds that $R_k X^{*c_k} = g^{s_k}$. The equality is in form of the verification of an individual signature, so intuitively, we would like to take (R_k, s_k) as a forged Schnorr signature.

Let H and H' denote the random oracles in the CK-SAEUF-CMA game against SASchnorr and the EUF-CMA game against Schnorr, respectively. For the reduction to win the latter game, it should hold that $c_k = \mathsf{H}'(R_k, m^*)$ for some m^*. On the other hand, $c_k = \mathsf{H}(\tilde{R}_k, m_k, X^*, s_{k-1}, k)$ in the former game. Therefore, to use (R_k, s_k) as its own forgery, the reduction has to find out R_k when handling the forger's hash query, given only $\tilde{R}_k$.

The key point is to retrieve $\tilde{R}_{k-1}$ with s_{k-1} (and then obtain $R_k = \tilde{R}_k / \tilde{R}_{k-1}$). We do so by setting the exponent of the expected response s_n as the index of each query $\mathsf{H}(\tilde{R}_n, m_n, X_n, s_{n-1}, n)$. It takes most of our effort to show this works. Simply speaking, we present a mathematical induction: we can retrieve unique $\tilde{R}_1$ with s_1; given that we can retrieve $\tilde{R}_{i-1}$ with s_{i-1}, we can successfully figure out $R_i = \tilde{R}_i / \tilde{R}_{i-1}$ and set the index of query $\mathsf{H}(\tilde{R}_i, m_i, X_i, s_{i-1}, i)$, and thus we can retrieve $\tilde{R}_i$ with s_i.

Following Fig. 4, we define s_0 as always 0. Moreover, we define $\tilde{R}_0$ as $1_{\mathbb{G}}$, which simplifies the discussion a bit.

Proof. (Theorem 3). Suppose $\mathcal{F}$ is the forger that breaks the CK-SAEUF-CMA security of SASchnorr. We construct an algorithm $\mathcal{A}$ that breaks the EUF-CMA security of Schnorr. In the EUF-CMA game, it has access to a signing oracle $\textsc{Sign}'$, and the hash function is modeled as a random oracle H'.

On target public key X^*, algorithm $\mathcal{A}$ first initializes an empty table $T[\cdot,\cdot,\cdot,\cdot,\cdot]$ for simulating the random oracle H. Each table item may have an index $I[\cdot,\cdot,\cdot,\cdot,\cdot]$ which is a group element. For any group element, $\mathcal{A}$ can efficiently locate the table item with an index equal to the element. Algorithm $\mathcal{A}$ runs $\mathcal{F}$ with the same target public key. It handles queries from $\mathcal{F}$ as follows:

- Hash queries. On a hash query $\mathsf{H}(\tilde{R}_j, X_j, m_j, s_{j-1}, j)$, algorithm $\mathcal{A}$ returns $T[\tilde{R}_j, X_j, m_j, s_{j-1}, j]$. If the item is undefined, $\mathcal{A}$ first defines it as follows. Algorithm $\mathcal{A}$ checks the following two conditions:
  C1 $X_j = X^*$;
  C2 $j = 1$; or among all defined items with the last arguments being $j-1$, there exists a unique one $T[\tilde{R}_{j-1}, X_{j-1}, m_{j-1}, s_{j-2}, j-1]$ whose index is $g^{s_{j-1}}$.
  If C2 is not true, $\mathcal{A}$ assigns $c \leftarrow\!\$\, \mathcal{H}$ to $T[\tilde{R}_j, X_j, m_j, s_{j-1}, j]$. If only C2 is true, $\mathcal{A}$ additionally sets the index
  $$I[\tilde{R}_j, X_j, m_j, s_{j-1}, j] = (\tilde{R}_j / \tilde{R}_{j-1}) X_j^c.$$
  If both conditions hold, $\mathcal{A}$ instead assigns $c = \mathsf{H}'(\tilde{R}_j/\tilde{R}_{j-1}, m^*)$, with m^* uniformly chosen from $\{0,1\}^{\log p}$, to $T[\tilde{R}_j, X_j, m_j, s_{j-1}, j]$. It sets the index in the same way. We say $\mathcal{A}$ *retrieves* $\tilde{R}_{j-1}$ here.
- Signing queries. To answer a signing query $\textsc{Sign}(\tilde{R}_{n-1}, s_{n-1}, m_n, n)$, $\mathcal{A}$ uniformly chooses m^* from $\{0,1\}^{\log p}$ and queries m^* to $\textsc{Sign}'$. It receives a Schnorr signature (R, s) on m^* under X^*. Let $\tilde{R}_n = \tilde{R}_{n-1} \cdot R$. Algorithm $\mathcal{A}$ aborts if $T[\tilde{R}_n, X^*, m_n, s_{n-1}, n]$ has been defined. Otherwise, $\mathcal{A}$ assigns $\mathsf{H}'(R, m^*)$ to $T[\tilde{R}_n, X^*, m_n, s_{n-1}, n]$. It returns $(\tilde{R}_n, s)$ to $\mathcal{F}$. It also checks condition C2 defined above and sets index $I[\tilde{R}_n, X^*, m_n, s_{n-1}, n] = g^s$ if C2 is true.

At last, $\mathcal{F}$ outputs a forgery with messages and public keys it chooses:

$$\{(X_1, m_1), \ldots, (X_n, m_n)\}, (\tilde{R}_n, \{s_1, \ldots, s_n\}).$$

Algorithm $\mathcal{A}$ runs the verification procedure. Namely, for $i = n, \ldots, 2$, it lets $c_i = T[\tilde{R}_i, X_i, m_i, s_{i-1}, i]$ and then computes $\tilde{R}_{i-1} = \tilde{R}_i / (g^{s_i} / X_i^{c_i})$. It finally lets $c_1 = T[\tilde{R}_1, X_1, m_1, 0, 1]$ and determines whether the forgery is valid by checking whether $g^{s_1} = \tilde{R}_1 X_1^{c_1}$. In this verification procedure, $\mathcal{A}$ aborts if it meets an undefined table item. This behavior is different from the verification algorithm Vf, since the item would be defined now if we run Vf. However, we will later show that the forgery is unlikely to be valid with such an undefined table item.

There must exist $k \in \{1, \ldots, n\}$ such that $X_k = X^*$, and $\mathcal{F}$ has not queried $\textsc{Sign}(\cdot, s_{k-1}, m_k, k)$ for $\mathcal{F}$ to win the CK-SAEUF-CMA game. From the forgery's validity, we know

$$(\tilde{R}_k / \tilde{R}_{k-1}) X^{*c_k} = g^{s_k},$$

where $c_k = T[\tilde{R}_k, X^*, m_k, s_{k-1}, k]$. If $c_k = \mathsf{H}'(\tilde{R}_k/\tilde{R}_{k-1}, m^*)$, and m^* is fresh in the EUF-CMA game (i.e., has not been queried to the signing oracle $\textsc{Sign}'$),

then $\mathcal{A}$ wins the game with a forged signature $(\tilde{R}_k/\tilde{R}_{k-1}, s_k)$ on message m^*. Our main task below is to prove this is exactly the case with high probability, guaranteed by how $\mathcal{A}$ handles the queries from $\mathcal{F}$.

To do so, we consider a list of events. We define them, explain how they relate to $\mathcal{A}$'s winning, and bound their probabilities. They are defined as follows:

E1 Algorithm $\mathcal{A}$ aborts when handling a signing query. We also use $\mathsf{SimFail}$ to denote this event.
E2 Algorithm $\mathcal{A}$ chooses some duplicate random messages from $\{0,1\}^{\log p}$. We also use MsgCol to denote this event.
E3 Forger $\mathcal{F}$ succeeds. We also use $\mathsf{Acc}_{\mathcal{F}}$ to denote this event.
E4 Algorithm $\mathcal{A}$ meets an undefined table item in the above verification procedure we described. We also use UnDef to denote this event.
E5 When $T[\tilde{R}_k, X^*, m_k, s_{k-1}, k]$ was defined, condition C2 was not true, or the aggregate commitment that $\mathcal{A}$ retrieved was not $\tilde{R}_{k-1}$.

As long as E3 happens while E2, E4, and E5 do not happen, $\mathcal{A}$ finds a forged Schnorr signature $(\tilde{R}_k/\tilde{R}_{k-1}, s_k)$ on a fresh message m^* (for the EUF-CMA game) and wins. Excluding E2 guarantees the freshness of m^*, excluding E4 guarantees $\mathcal{A}$ does not abort in the verification procedure, and excluding E5 guarantees c_k was indeed set to $\mathsf{H}'(\tilde{R}_k/\tilde{R}_{k-1}, m^*)$. If E1 and E2 do not happen, then the simulated game is identical to the real CK-SAEUF-CMA game, and we know E3 happens with probability at least ε on such a condition. Here we also exclude E2 to avoid one hash value $\mathsf{H}'(R, m^*)$ being assigned to different table items. Below we separately consider the probabilities of these events.

E1 For every signing query from $\mathcal{F}$, $\tilde{R}_n$ to be returned is uniformly distributed on a set of order p. This is because $\tilde{R}_n = \tilde{R}_{n-1} \cdot R$ with R uniformly distributed on $\mathbb{G}$, since R is the commitment of a $\mathsf{Schnorr}$ signature from $\textsc{Sign}'$. This $\tilde{R}_n$ may collide with the at most $q_{\mathsf{H}} + q_{\mathsf{S}}$ aggregate commitments occurring in T. Hence, $\mathsf{SimFail}$ happens in every signing query with probability at most $(q_{\mathsf{H}} + q_{\mathsf{S}})/p$. In total, we have $\Pr[\mathsf{SimFail}] \leq q_{\mathsf{S}}(q_{\mathsf{H}} + q_{\mathsf{S}})/p$.

E2 Algorithm $\mathcal{A}$ needs to choose at most one message from $\{0,1\}^{\log p}$ for every signing query and hash query from $\mathcal{F}$. The total number of the chosen messages is bounded by $q_{\mathsf{H}} + q_{\mathsf{S}}$, and it follows that $\Pr[\mathsf{MsgCol}] \leq (q_{\mathsf{H}} + q_{\mathsf{S}})^2/(2p)$.

E4 To bound the probability of this event, we need Lemma 4 below. Note that when one verifies a forgery with Vf, all the recursive calls return equal values. Hence, as long as $\mathcal{A}$ meets an undefined table item, the probability of the whole forgery's validity is bounded by $(q_{\mathsf{H}} + q_{\mathsf{S}} + 1)/|\mathcal{H}|$. Namely, we have $\Pr[\mathsf{Acc}_{\mathcal{F}} \mid \mathsf{UnDefAcc}_{\mathcal{F}}] \leq (q_{\mathsf{H}} + q_{\mathsf{S}} + 1)/|\mathcal{H}|$.

Lemma 4. *For any* $\{(X_1, m_1), \ldots, (X_j, m_j)\}, (\tilde{R}_j, \{s_1, \ldots, s_j\})$*, if table item* $T[\tilde{R}_j, X_j, m_j, s_{j-1}, j]$ *is undefined, then the probability that*

$$\mathsf{Vf}(\{(X_1, m_1), \ldots, (X_j, m_j)\}, (\tilde{R}_j, \{s_1, \ldots, s_j\})) = 1$$

is upper-bounded by $(q_{\mathsf{H}} + q_{\mathsf{S}} + 1)/|\mathcal{H}|$.

E5 We consider condition C2 in two aspects. First, it requires that there *exists* an item $T[\tilde{R}_{j-1}, X_{j-1}, m_{j-1}, s_{j-2}, j-1]$ with index being $g^{s_{j-1}}$. Second, it requires the item to be *unique*. Lemma 5 and 6 relate to the uniqueness and existence requirements respectively.

Lemma 5. *Let q_j be the number of defined entries in T with the last argument being j. Define* Dup *as the event that there exist two different table items*

$$T[\tilde{R}_j, X_j, m_j, s_{j-1}, j] \quad \textit{and} \quad T[\tilde{R}'_j, X'_j, m'_j, s'_{j-1}, j]$$

with the last arguments being equal, such that

$$I[\tilde{R}_j, X_j, m_j, s_{j-1}, j] = I[\tilde{R}'_j, X'_j, m'_j, s'_{j-1}, j].$$

It holds that $\Pr[\mathsf{Dup}] \leq (\sum_{i=1}^{\infty} q_i^2)/(2|\mathcal{H}|)$.

Lemma 6. *Let q_j be the number of defined entries in T with the last argument being j. Define* BadOrder *as the event that there exists a* valid chain *in T, namely a set of items*

$$c_1 = T[\tilde{R}_1, X_1, m_1, 0, 1], \ldots, c_j = T[\tilde{R}_j, X_j, m_j, s_{j-1}, j]$$

satisfying $(\tilde{R}_i/\tilde{R}_{i-1})X_j^{c_i} = g^{s_i}$ for $i = 1, \ldots, j-1$, while these items were not defined in order. It holds that $\Pr[\mathsf{BadOrder} \mid \neg\mathsf{DupBadOrder}] \leq (\sum_{i=1}^{\infty} q_i q_{i+1})/|\mathcal{H}|$.

We now show the link between E5 and these two lemmas. For a valid chain described in Lemma 6, suppose the items in it are defined in order. We use an induction to show the following statement is true for every item in the chain if Dup does not happen: for the item $T[\tilde{R}_i, X_i, m_i, s_{i-1}, i]$ in the chain, condition C2 was true when it is defined, and $\mathcal{A}$ exactly retrieved $\tilde{R}_{i-1}$ at that time.

For the first item in the chain, the statement is true directly from the definition of C2, and its index is g^{s_1} from the validity of the chain. Assume the statement is true for the $(i-1)$-th item, and its index is $g^{s_{i-1}}$. When the i-th item in the chain is going to be defined, the $(i-1)$-th has been defined. From the assumption, the index of the $(i-1)$-th item has been defined and equals $g^{s_{i-1}}$. That Dup does not happen guarantees there does not exist another item with the last argument being $i-1$ and equal index. Thus, condition C2 for the i-th item holds, and $\mathcal{A}$ retrieves $\tilde{R}_{i-1}$. The index of the i-th item is thus g^{s_i} from the validity of the chain. This means that the statement is true for the i-th item. By induction, the statement is true for every item in the chain.

Obviously, the forgery must correspond to a valid chain for it to be valid, conditioned on that E4 does not happen. The above statement means that E5 is impossible if none of BadOrder and Dup happen. The probability of E5 is hence bounded by

$$\begin{aligned}\Pr[\mathsf{Dup} \vee \mathsf{BadOrder}] &= \Pr[\mathsf{Dup}] + \Pr[\mathsf{BadOrder} \mid \neg\mathsf{Dup}] \\ &\leq \frac{\sum_{i=1}^{\infty} q_i^2 + \sum_{i=1}^{\infty} 2q_i q_{i+1}}{2|\mathcal{H}|} \\ &\leq \frac{(q_{\mathsf{H}} + q_{\mathsf{S}})^2}{2|\mathcal{H}|},\end{aligned}$$

where the last inequality follows from $q_{\mathsf{H}} + q_{\mathsf{S}} = \sum_{i=1}^{\infty} q_i$.

Put all these bounds together, and we have

$$\begin{aligned}
\varepsilon' &\geq \Pr[\mathsf{Acc}_{\mathcal{F}} \wedge \neg\mathsf{MsgCol} \wedge \neg\mathsf{UnDef} \wedge \neg\mathsf{Dup} \wedge \neg\mathsf{BadOrder}] \\
&\geq \Pr[\mathsf{Acc}_{\mathcal{F}} \wedge \neg\mathsf{MsgCol}] - \Pr[\mathsf{UnDef}] - \Pr[\mathsf{Dup} \vee \mathsf{BadOrder}] \\
&\geq \Pr[\mathsf{Acc}_{\mathcal{F}} \wedge \neg\mathsf{SimFail} \wedge \neg\mathsf{MsgCol}] \\
&\quad - \Pr[\mathsf{Acc}_{\mathcal{F}} \wedge \mathsf{UnDef}] - \Pr[\mathsf{Dup} \vee \mathsf{BadOrder}] \\
&\geq \Pr[\mathsf{Acc}_{\mathcal{F}} \mid \neg\mathsf{SimFail} \wedge \neg\mathsf{MsgColAcc}_{\mathcal{F}}] \cdot \Pr[\neg\mathsf{SimFail} \wedge \neg\mathsf{MsgCol}] \\
&\quad - \Pr[\mathsf{Acc}_{\mathcal{F}} \wedge \mathsf{UnDef}] - \Pr[\mathsf{Dup} \vee \mathsf{BadOrder}] \\
&\geq \Pr[\mathsf{Acc}_{\mathcal{F}} \mid \neg\mathsf{SimFail} \wedge \neg\mathsf{MsgColAcc}_{\mathcal{F}}] - \Pr[\mathsf{SimFail}] - \Pr[\mathsf{MsgCol}] \\
&\quad - \Pr[\mathsf{Acc}_{\mathcal{F}} \mid \mathsf{UnDefAcc}_{\mathcal{F}}] - \Pr[\mathsf{Dup} \vee \mathsf{BadOrder}] \\
&\geq \varepsilon - \frac{(q_{\mathsf{H}} + q_{\mathsf{S}})(q_{\mathsf{H}} + 3q_{\mathsf{S}})}{2p} - \frac{(q_{\mathsf{H}} + q_{\mathsf{S}} + 1)^2 + 1}{2|\mathcal{H}|}
\end{aligned}$$

It only remains for us to bound the running time of $\mathcal{A}$. We assume a table operation takes constant time with enough space and a hash table implemented properly. We also assume retrieving a table item as described in condition C2 also takes constant time with an index structure implemented properly. In total, the time $\mathcal{A}$ spends on handling queries from $\mathcal{F}$ and maintaining table T is bounded by $\mathcal{O}(q_{\mathsf{S}} + q_{\mathsf{H}})$.

Note that $\mathcal{A}$ runs a verification procedure on $\mathcal{F}$'s forgery in order to obtain $\tilde{R}_k/\tilde{R}_{k-1}$, the commitment part of its own forged Schnorr signature. This takes at most $2N$ exponentiation operations. In conclusion, we have

$$t' \leq t + 2Nt_{\text{exp}} + \mathcal{O}(q_{\mathsf{S}} + q_{\mathsf{H}}).$$

□

We defer the proofs of Lemma 4 to 6 to the full version of this paper [7].

Acknowledgement. We are grateful to the anonymous reviewers of ESORICS 2022 for their insightful and very helpful comments, and particularly to Yvo Desmedt for shepherding and invaluable advice, which have significantly improved this work.

This work is supported by the following foundation items: The National Natural Science Foundation of China (No. U1536205, No. 61472084), The National Key Research and Development Program of China (No. 2017YFB0802000), Shanghai Innovation Action Project (No. 16DZ1100200), Shanghai Science and Technology Development Funds (No. 16JC1400801), Technical Standard Project of Shanghai Scientific and Technological Committee (No. 21DZ2200500), Shandong Provincial Key Research and Development Program (No. 2017CXG0701, No. 2018CXGC0701).

References

1. Ahn, J.H., Green, M., Hohenberger, S.: Synchronized aggregate signatures: new definitions, constructions and applications. In: Al-Shaer, E., Keromytis, A.D., Shmatikov, V. (eds.) ACM CCS 2010, pp. 473–484. ACM Press, October 2010. https://doi.org/10.1145/1866307.1866360

2. Kılınç Alper, H., Burdges, J.: Two-round trip Schnorr multi-signatures via delinearized witnesses. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part I. LNCS, vol. 12825, pp. 157–188. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84242-0_7
3. Bellare, M., Namprempre, C., Neven, G.: Unrestricted aggregate signatures. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 411–422. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73420-8_37
4. Bernstein, D.J., Duif, N., Lange, T., Schwabe, P., Yang, B.-Y.: High-speed high-security signatures. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 124–142. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23951-9_9
5. Boneh, D., Gentry, C., Lynn, B., Shacham, H.: Aggregate and verifiably encrypted signatures from bilinear maps. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 416–432. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-39200-9_26
6. Chalkias, K., Garillot, F., Kondi, Y., Nikolaenko, V.: Non-interactive half-aggregation of EdDSA and variants of Schnorr signatures. In: Paterson, K.G. (ed.) CT-RSA 2021. LNCS, vol. 12704, pp. 577–608. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-75539-3_24
7. Chen, Y., Zhao, Y.: Half-aggregation of Schnorr signatures with tight reductions. Cryptology ePrint Archive, Report 2022/222 (2022). https://ia.cr/2022/222
8. Dimitrov, V.S., Jullien, G.A., Miller, W.C.: Complexity and fast algorithms for multiexponentiations. IEEE Trans. Comput. **49**(2), 141–147 (2000)
9. Fuchsbauer, G., Kiltz, E., Loss, J.: The algebraic group model and its applications. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part II. LNCS, vol. 10992, pp. 33–62. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96881-0_2
10. Fuchsbauer, G., Plouviez, A., Seurin, Y.: Blind Schnorr signatures and signed ElGamal encryption in the algebraic group model. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part II. LNCS, vol. 12106, pp. 63–95. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45724-2_3
11. Gentry, C., Ramzan, Z.: Identity-based aggregate signatures. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) PKC 2006. LNCS, vol. 3958, pp. 257–273. Springer, Heidelberg (2006). https://doi.org/10.1007/11745853_17
12. Goldwasser, S., Micali, S., Rivest, R.L.: A digital signature scheme secure against adaptive chosen-message attacks. SIAM J. Comput. **17**(2), 281–308 (1988)
13. Gordon, D.M.: A survey of fast exponentiation methods. J. Algorithms **27**(1), 129–146 (1998)
14. Hankerson, D., Menezes, A.: Elliptic Curve Signature Schemes, pp. 1–3. Springer, Heidelberg (2019). https://doi.org/10.1007/978-3-642-27739-9_251-2
15. Hankerson, D., Menezes, A.J., Vanstone, S.: Guide to Elliptic Curve Cryptography. Springer, New York (2006)
16. Hartung, G., Kaidel, B., Koch, A., Koch, J., Rupp, A.: Fault-tolerant aggregate signatures. In: Cheng, C.-M., Chung, K.-M., Persiano, G., Yang, B.-Y. (eds.) PKC 2016, Part I. LNCS, vol. 9614, pp. 331–356. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49384-7_13
17. Lee, K., Lee, D.H., Yung, M.: Sequential aggregate signatures with short public keys: design, analysis and implementation studies. In: Kurosawa, K., Hanaoka, G. (eds.) PKC 2013. LNCS, vol. 7778, pp. 423–442. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36362-7_26

18. Lu, S., Ostrovsky, R., Sahai, A., Shacham, H., Waters, B.: Sequential aggregate signatures and multisignatures without random oracles. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 465–485. Springer, Heidelberg (2006). https://doi.org/10.1007/11761679_28
19. Lysyanskaya, A., Micali, S., Reyzin, L., Shacham, H.: Sequential aggregate signatures from trapdoor permutations. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 74–90. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24676-3_5
20. Maurer, U.: Abstract models of computation in cryptography. In: Smart, N.P. (ed.) Cryptography and Coding 2005. LNCS, vol. 3796, pp. 1–12. Springer, Heidelberg (2005). https://doi.org/10.1007/11586821_1
21. Rotem, L., Segev, G.: Tighter security for Schnorr identification and signatures: a high-moment forking lemma for Σ-protocols. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part I. LNCS, vol. 12825, pp. 222–250. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84242-0_9
22. Schnorr, C.P.: Efficient identification and signatures for smart cards. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 239–252. Springer, New York (1990). https://doi.org/10.1007/0-387-34805-0_22
23. Shoup, V.: Lower bounds for discrete logarithms and related problems. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 256–266. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-69053-0_18
24. Wuille, P., Nick, J., Ruffing, T.: Schnorr signatures for secp256k1. https://github.com/bitcoin/bips/blob/master/bip-0340.mediawiki
25. Zhao, Y.: Practical aggregate signature from general elliptic curves, and applications to blockchain. In: Galbraith, S.D., Russello, G., Susilo, W., Gollmann, D., Kirda, E., Liang, Z. (eds.) ASIACCS 19, pp. 529–538. ACM Press, July 2019. https://doi.org/10.1145/3321705.3329826