Half-Aggregation of Schnorr Signatures with Tight Reductions

Yanbo Chen and Yunlei Zhao

Fudan University
{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, like blockchain systems, network routing, e-voting, and certificate chains. 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. [8] 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. Chalkias et al. also presented an aggregate scheme for Schnorr signatures whose security can be tightly reduced to the security of Schnorr signatures in the random oracle model (ROM). However, the scheme is rather expensive and does not achieve half-aggregation. It is a fundamental question whether there exists half-aggregation of Schnorr signatures with tight reduction in the ROM, of both theoretical and practical interests.

This work's contributions are threefold. We first give a tight security proof for the scheme in [8] 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. [6]. 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, like blockchain and e-voting.

Lysyanskaya et al. [16] proposed a useful variant of aggregate signatures, *sequential aggregate signatures* (SAS). In an SAS scheme, the signatures can 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 and routing protocols. They also have applications in blockchain systems, e.g., in Bitcoin's lightning network. In these scenarios, the signatures are produced and passed in order, and a signer always knows the previous aggregation.

Boneh et al. pointed out in their seminal work [6] 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 schemes are based on bilinear maps [4, 6, 14, 15] and trapdoor permutations [16]. Some designs also require the synchronized model [1, 12]. Constructing AS 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. [8] 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 significantly reduces the storage at basically no cost. Hence, it is still very useful and timely as Schnorr signature will be enforced in Bitcoin (and many other blockchain systems) in November of 2021 with the Taproot update.

We observe two problems of ASchnorr. In [8], 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 pa-

rameter. 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. 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 when aggregating Schnorr signatures.

1.1 Contribution

The contributions of this work are threefold. For the first problem of ASchnorr, 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) [10]. The AGM is a weaker ideal model than the generic group model (GGM) [17, 22]. 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 to security proofs for cryptographic schemes [2, 3, 11, 19].

For the second problem, 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). Compared to tightASchnorr proposed in [8], 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. The key point is that SASchnorr uses an aggregating method completely different from the other schemes. In particular, the aggregation does not rely on group homomorphism, and the verifier can recovers all the individual signatures.

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 for permissionless blockchain systems in general, but it is useful in many other applications like network routing, certificate chains, and Bitcoin's lighting network. 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.

1.2 Overview of Constructions

Here we give the main ideas of our three schemes, ASchnorr, IASchnorr, and SASchnorr.

ASchnorr. It is a natural idea to half-aggregate Schnorr signatures utilizing the property of group exponentiation. Take the case of aggregating two signatures as an example. Suppose we have two Schnorr signatures $\sigma_1 = (R_1, s_1)$ and $\sigma_2 = (R_2, s_2)$, respectively on messages m_1 and m_2 under public keys X_1 and X_2. Let $c_1 = \mathsf{H}_1(R_1, m_1)$ and $c_2 = \mathsf{H}_1(R_2, m_2)$. It must hold that $g^{s_1} = R_1 X_1^{c_1}$ and $g^{s_2} = R_2 X_2^{c_2}$ for these two signatures to be valid. The most direct way to half-aggregate them is to compress s_1 and s_2 into $\tilde{s} = s_1 + s_2$. It thus holds that $g^{\tilde{s}} = R_1 X_1^{c_1} R_2 X_2^{c_2}$. However, this aggregating method is insecure, for the challenge of one signature, say c_1, does not depend on the commitment of another signature, namely R_2. A forger may not know the discrete logarithm of X_1 and hence can not produce the response s_1, i.e., the discrete logarithm of $R_1 X_1^{c_1}$, but it can easily eliminate this term with R_2. It lets $R_2 = g^{r_2}/(R_1 X_1^{c_1})$ instead of normally g^{r_2} and thus only needs to produce a normal signature under X_2 to forge the aggregation.

The solution in ASchnorr is using "outer coefficients" depending on all individual signatures. Let

$$L = \{(R_1, X_1, m_1), (R_2, X_2, m_2)\},$$

$a_1 = \mathsf{H}_2(L, 1)$, and $a_2 = \mathsf{H}_2(L, 2)$. The aggregated response becomes $\tilde{s} = a_1 s_1 + a_2 s_2$, and the aggregate verification becomes checking whether $g^{\tilde{s}} = (R_1 X_1^{c_1})^{a_1} (R_2 X_2^{c_2})^{a_2}$. The term $X_1^{c_1 a_1}$ depends on the second signature and the corresponding message and public key, so we prevent the above attack.

This aggregating method is indeed provably secure if H_2 is modeled as a random oracle. The random oracle H_2 provides a forking point so that the reduction can rewind a forger to extract a forged individual signature. However, the rewinding-based proof in the ROM suffers from a quadratic loss. In the AGM+ROM, the reduction goes relatively straightforward. It can extract a forged signature running the forger once and hence achieve a tight bound.

IASchnorr. Things seem to go well so far. However, as we have explained, incremental aggregation is desired in practice. An aggregate scheme that supports incremental aggregation does not require the aggregator to store whole individual signatures when waiting for unreceived signatures, so it saves the storage and balances the time overhead. Now let us evaluate how ASchnorr supports incremental aggregation. Continue our two-signature example. Suppose we are going to add one more signature, $\sigma_3 = (R_3, s_3)$, on message m_3 under public key X_3, into the existing aggregation. We can use the same aggregate method, treating the aggregation as a normal signature. We let

$$L' = \{L, (R_3, X_3, m_3)\},$$

$a'_1 = \mathsf{H}_2(L', 1)$, and $a'_2 = \mathsf{H}_2(L', 2)$. The new aggregated response is thus $\tilde{s}' = a'_1\tilde{s} + a'_2 s_3$.

The first problem is ambiguity. The same three signatures may be compressed into different aggregations, depending on whether the third signature is aggregated together with the first two signatures or incrementally. To make an aggregate signature correctly verifiable, we also need information about how it is aggregated.

The second problem is redundant operations. We need three coefficients, respectively for s_1, s_2, and s_3, but we compute four hash values to obtain $a_1a'_1$, $a_2a'_1$, and a'_2 as those coefficients. Generally, whenever we do an incremental aggregating, the hash value for the pre-existing aggregation is redundant and brings extra multiplications.

We may prevent the redundant operations when using ASchnorr to do incremental aggregating by omitting the hashing for the pre-existing aggregation, but the ambiguity is intrinsic for ASchnorr. When we use ASchnorr to aggregate σ_1 and σ_2, the outer coefficient a_2 is $\mathsf{H}_2(L_2, 2)$ with

$$L_2 = \{(R_1, X_1, m_1), (R_2, X_2, m_2)\}.$$

When we aggregate σ_1, σ_2, and σ_3, we have a different coefficient $a'_2 = \mathsf{H}_2(L_3, 2)$ with

$$L_3 = \{(R_1, X_1, m_1), (R_2, X_2, m_2), (R_3, X_3, m_3)\}.$$

We can thus see the essential problem: different ways of aggregation yield different coefficients a_2 and a'_2. Hence, we need to know whether the third signature is incrementally aggregated, otherwise we cannot always verify it correctly.

In IASchnorr, we solve the problems by letting the outer coefficients only depend on earlier signatures, rather than all individual signatures. Precisely, let

$$L_i = \{(R_1, X_1, m_1), \ldots, (R_i, X_i, m_i)\},$$

and the i-th coefficient becomes $a_i = \mathsf{H}_2(L_i)$. Note that now a_2 is always $\mathsf{H}_2(L_2)$, no matter whether more signatures are aggregated. When we add more signatures into a pre-existing aggregation, we only compute the outer coefficients for the new ones.

In IASchnorr, a signature only influences the coefficients for itself and those signatures aggregated later. Compared with ASchnorr, this seems a security loss, but we can prove IASchnorr is actually as secure as ASchnorr in the AGM+ROM.

SASchnorr. What prevents ASchnorr's security to tightly reduce to Schnorr's security in the ROM? An important observation is that ASchnorr compresses the response parts of individual signatures *unrecoverably*. The aggregate verifier can not see the whole individual signatures. The reduction can extract an individual signature in the ROM, but only by rewinding, which makes the bound untight. Furthermore, it takes many times of rewinding to extract an individual signature when trying proving IASchnorr's security in the ROM, making the security loss unacceptably big. In light of this, we consider a completely different aggregating method that aggregates the signatures recoverably. It can be implemented sequentially, so we obtain an SAS scheme SASchnorr as a result.

In SASchnorr, we aggregate the commitments instead of the responses. When the i-th signer produces a signature, it already knows an existing aggregation that contains an aggregated commitment $\tilde{R}_{i-1}$ and the response from the last signer s_{i-1}. Instead of hashing its own commitment R_i to get the challenge, the signer hashes the new aggregated commitment $\tilde{R}_i = \tilde{R}_{i-1} \cdot R_i$ together with s_{i-1}. To verify the sequential aggregate signature, the verifier computes the hash value c_i and gets $R_i = g^{s_i}/X_i^{c_i}$. It then recovers the earlier aggregation $\tilde{R}_{i-1} = \tilde{R}_i/R_i$ and iteratively runs the procedure. Compared with aggregating responses, the verifier here can recover all the individual signatures.

We can indeed prove that SASchnorr is as secure as Schnorr in only the ROM. In the textbook security proof for Schnorr, the random oracle $\mathsf{H}(R, m)$ provides a forking point that "anchors" R in two executions of the forgery. For SASchnorr, we make an interesting argument that hashing $\tilde{R}_i$ and s_{i-1} is also sufficient to anchor R_i.

We summarize our schemes' performance in Table 1. Our results, including the new security result for ASchnorr and new schemes IASchnorr and SASchnorr, are shown in bold. We refer to Schnorr and TightASchnorr for comparison.

1.3 Related Work

The associations and differences between aggregate signatures and multi-signatures are worth mentioning, especially considering the great research interest in Schnorr-based multi-signatures these years [2, 9, 18, 19]. A multi-signature (MS) scheme allows multiple signers to produce a signature on a message interactively. All the signers authenticate the message, while the storage and verification overhead is as little as an individual signature.

We stress that the real-time interactivity of multi-signatures is an important difference from aggregate signatures. In an AS scheme, the signers normally produce individual signatures without knowing the existence of each other, and then an unspecified aggregator aggregates those signatures. In an SAS scheme, the current signer receives the earlier aggregation, but the communication is not real-time. AS can be viewed as a very restricted version of multi-signature.

The non-interactivity of AS makes it difficult to compress multiple signatures into constant size, especially for signatures following the Fiat-Shamir paradigm. In a Schnorr-based multi-signature scheme, the signers agree on a common commitment and then respond to a common challenge, which can only be realized

Scheme	AS?	SAS?	Size	Security	
				ROM	AGM+ROM
Schnorr	×	×	$2n \cdot 2\lambda$		
ASchnorr	√	×	$(n+1)2\lambda$	$\mathcal{O}(\sqrt{\varepsilon_{\mathrm{Sch}}})$	$\mathcal{O}(\boldsymbol{\varepsilon}_{\mathrm{Sch}})$
TightASchnorr	√	×	$(n+r)2\lambda + rl$	$\mathcal{O}(\varepsilon_{\mathrm{Sch}})$	
IASchnorr	**√**	**√**	$\boldsymbol{(n+1)2\lambda}$		$\mathcal{O}(\boldsymbol{\varepsilon}_{\mathrm{Sch}})$
SASchnorr	×	**√**	$\boldsymbol{(n+1)2\lambda}$	$\mathcal{O}(\boldsymbol{\varepsilon}_{\mathrm{Sch}})$	

Table 1. A summarization of our results. We show whether the schemes are AS schemes or SAS schemes, and we regard IAS schemes as both. We compare these schemes in the size of the aggregation of n individual signatures and their security in ROM and AGM+ROM, separately. Here λ is the security parameter. We assume these schemes work on a group where the representation of elements is compact (such as elliptic curve groups), so both the commitment and the response of a Schnorr signature are of about 2λ bits. Note that TightASchnorr not only suffers from larger aggregation sizes, but also has more expensive aggregating and verification procedure. In particular, TightASchnorr has scheme parameters r and l, which are typically set to $r = \mathcal{O}(\lambda/\log\lambda)$ and $l = \mathcal{O}(\log\lambda)$. Its aggregation (resp. verification) time is longer by a factor $r2^l$ (resp. r). We refer to non-aggregate scheme Schnorr as a baseline. We suppose Schnorr is $(t, \varepsilon_{\mathrm{Sch}})$-secure, and we show the upper bounds of the success probabilities breaking these schemes in the ROM or the AGM+ROM in t time. In particular, $\mathcal{O}(\sqrt{\varepsilon_{\mathrm{Sch}}})$ means the security reduction is untight, while $\mathcal{O}(\varepsilon_{\mathrm{Sch}})$ means it is tight.

with real-time interactions. Note that even with real-time interactions, compressing multiple signatures into constant size is not easy. In Schnorr-based MS schemes [2, 19], a signer has to broadcast more than a normal Schnorr signature to other signers.

If we allow real-time interactions for AS schemes, we will get the notion of *interactive* aggregate signatures. Interactive AS schemes are generalized from multi-signature schemes by allowing the signers to sign different messages. Bellare and Neven [5] pointed out that we can trivially transform a MS scheme into an interactive AS scheme. In particular, we make the signers exchange their messages and let the concatenation of those messages be the actual message to be signed. This idea is recently used to construct a lattice-based interactive AS scheme [7].

2 Preliminaries

Notation. If x and y are variables, $y := x$ denotes that we assign x's value to y. If y is a variable and S is a set, $y \leftarrow\!\$\, S$ denotes that we uniformly choose an element from S at random and assign it to y. If S is a set, $|S|$ denotes the order of S. By $\log x$ we mean the length of x's bit-representation (so it is always an integer).

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 (correct) 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 $\mathsf{IncrAgg}$. Algorithm $\mathsf{IncrAgg}$ 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. [6] brought the existential unforgeability under chosen-message attacks (EUF-CMA) [13] of normal signature schemes into the context of aggregate signatures. In particular, they defined the EUF-CMA security of AS schemes in the *aggregate chosen-key model*. We abbreviate the EUF-CMA security in this model as CK-AEUF-CMA. In the CK-AEUF-CMA game, a forger $\mathcal{F}$ is given a target public key, and its goal is to forge an aggregate signature under a list of public keys including the target one. The forger can choose the messages to sign and all the public keys except the target one. Precisely, 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 $\mathcal{F}$ is provided the access to a signing oracle. It can adaptively requests signatures under pk^* on messages of its choice. Precisely, on $\mathcal{F}$'s query m, the signing oracle returns σ given by $\mathsf{Sign}(\mathsf{sk}^*, m)$, where sk^* is the secret key corresponding to pk^*.

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. We say a forger $\mathcal{F}$ $(t, q_{\mathsf{H}_1}, \ldots, q_{\mathsf{H}_l}, q_{\mathsf{S}}, N, \varepsilon)$-breaks the CK-AEUF-CMA security of an aggregate

signature scheme AS in the ROM if: $\mathcal{F}$ runs in time at most t; $\mathcal{F}$ makes at 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 AS; $\mathcal{F}$ makes at most q_{S} queries to the signing oracle; $\mathcal{F}$ gives a forged aggregate signature of length at most N; and $\mathcal{F}$ wins the CK-AEUF-CMA game with probability at least ε. In particular, the security results for AS schemes in this work are independent of N, so we simply say the forger $(t, q_{\mathsf{H}_1}, \ldots, q_{\mathsf{H}_l}, q_{\mathsf{S}}, \varepsilon)$-breaks the security.

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 $m_1, \ldots, m_{n-1}$ and public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_{n-1}$, 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.

Similar to the CK-AEUF-CMA security of AS schemes, Lysyanskaya et al. [16] defined the EUF-CMA security in the *sequential aggregate chosen-key model* of SAS schemes, abbreviated as CK-SAEUF-CMA. 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 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 [10]. 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) [17, 22] 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 $Y = \prod_{i=1}^{n} X_i^{\alpha_i}$.

3 Half-Aggregation of Schnorr Signatures, Revisited

3.1 Scheme Description

In this section, we analyze the security of $\mathsf{ASchnorr}$, the half-aggregate scheme for Schnorr signatures in [8], in the AGM+ROM. Fig. 1 describes the scheme, where H_1 and H_2 are hash functions on $\mathbb{Z}_p$. We use $\mathcal{H}_2$ to denote the range of H_2. The first three algorithms KGen, Sign, and Vf exactly constitutes $\mathsf{Schnorr}$.

On signatures $(R_1, s_1), \ldots, (R_n, s_n)$, respectively on messages $m_1, \ldots, m_n$ under public keys $X_1, \ldots, X_n$, algorithm Agg computes n coefficients $a_1, \ldots, a_n$ and aggregates the responses into $\tilde{s} = \sum_{i=1}^{n} a_i s_i$. To be precise, Agg lets

$$L = \{(R_1, X_1, m_1), \ldots, (R_n, X_n, m_n)\}$$

and computes hash values $a_i = \mathsf{H}_2(L, i)$ for $i = 1, \ldots, n$ as the coefficients. 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 $c_i = \mathsf{H}_1(R_i, m_i)$.

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 a random oracle H_2. We will briefly discuss the case when H_1 is also modeled as a random oracle after the proof.

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

$$\varepsilon' \geq \varepsilon - \frac{q_{\mathsf{H}_2} + 1}{|\mathcal{H}_2|}.$$

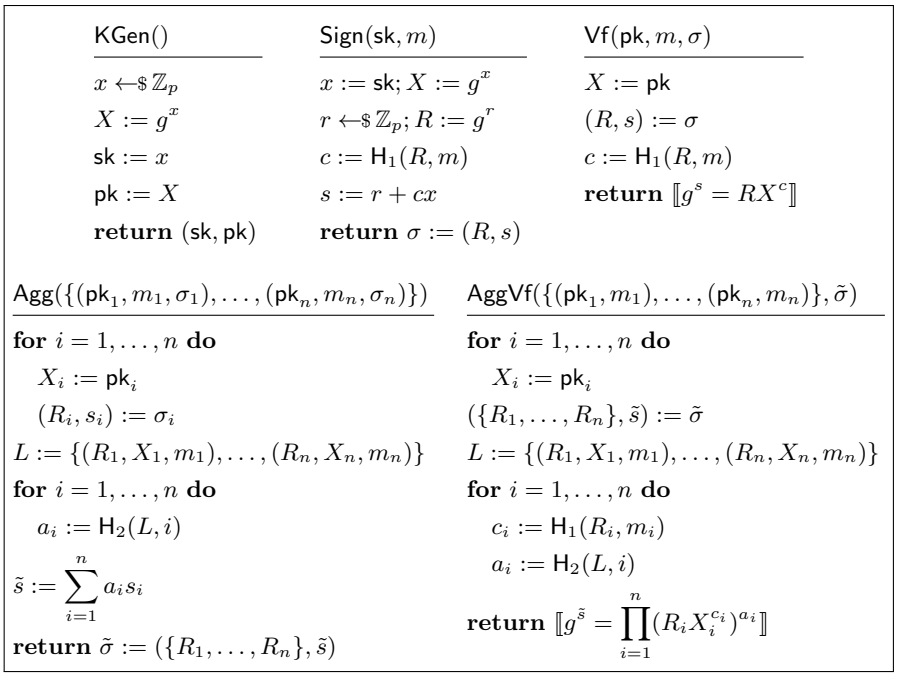

Fig. 1. Description of $\mathsf{ASchnorr}$. The cyclic group $\mathbb{G}$, of order p with a generator g, and the hash functions H_1 and H_2 are scheme-level parameters. The range of H_2 is denoted by $\mathcal{H}_2$.

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 the j-th signing query $\textsc{Sign}(m)$ from $\mathcal{F}$, $\mathcal{A}$ queries m to the signing oracle in the EUF-CMA game it is playing. It then receives a signature $(\hat{R}_j, \hat{s}_j)$ on m under X^*. Let $\hat{c}_j = \mathsf{H}_1(\hat{R}_j, m)$. It holds that $\hat{R}_j = g^{\hat{s}_j} / X^{*\hat{c}_j}$ from the validity of the signature. Algorithm $\mathcal{A}$ records the pair $(\hat{s}_j, \hat{c}_j)$ and returns $(\hat{R}_j, \hat{s}_j)$ 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, the target public key X^*, and the commitment parts $\hat{R}_1, \ldots, \hat{R}_{q_{\mathsf{S}}}$ of the signatures returned by the signing oracle. Meanwhile, $\mathcal{A}$ can also represent those commitments by g and X^*. Thus, $\mathcal{A}$ can represent every group element $\mathcal{F}$ queries or outputs by g and X^*. Precisely, when

$\mathcal{F}$ queries or outputs a group element, it additionally provides $(\alpha, \beta, \gamma_1, \ldots, \gamma_{q_S})$ such that the element equals $g^{\alpha} X^{*\beta} \prod_{j=1}^{q_S} \hat{R}_j^{\gamma_j}$. Let

$$\alpha' = \alpha + \sum_{j=1}^{q_S} \gamma_j \hat{s}_j \quad \text{and} \quad \beta' = \beta - \sum_{j=1}^{q_S} \gamma_j \hat{c}_j.$$

It can be verified that the element also equals $g^{\alpha'} X^{*\beta'}$, which $\mathcal{A}$ can compute from the pairs $(\hat{s}_1, \hat{c}_1)$, ..., $(\hat{s}_{q_S}, \hat{c}_{q_S})$ it recorded.[1] 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, ..., m_n and public keys X_1, ..., X_n it chooses. Note that $\mathcal{A}$ can represent all the group elements $\mathcal{F}$ outputs by g and X^*. Precisely, $\mathcal{A}$ knows $2n$ pairs $(\alpha_{1,1}, \beta_{1,1})$, ..., $(\alpha_{1,n}, \beta_{1,n})$, $(\alpha_{2,1}, \beta_{2,1})$, ..., $(\alpha_{2,n}, \beta_{2,n})$ 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 $X_k = X^*$ and $\beta_{1,k} + c_k \beta_{2,k} = 0$, 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 $\mathcal{A}$ does not win in the above way. 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 $R_i X_i^{c_i} \neq g^0$ (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

[1] Forger $\mathcal{F}$ may have not received $\hat{R}_j$ when it queries the element. In this case we can regard γ_j as always 0, and the undefined $\hat{R}_j$, $\hat{c}_j$, and $\hat{s}_j$ are irrelevant.

regard it as the last one to be determined. Since $R_i X_i^{c_i} \neq g^0$, 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 ways (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 $R_i X_i^{c_i} \neq g^0$, 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 win in the first way (by extracting a forged signature), does not abort because of undefined table items, and also does not win in the second way (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 win in the first way, it must hold that $\beta_{i,k} + c_k \beta_{2,k} \neq 0$, and hence the second condition is true. 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 the second way (only when $\beta^* = 0$), the third condition is true.

Among those $i \in \mathcal{I}$ satisfying $\beta_i^* \neq 0$, consider the last one 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 (running time of $\mathcal{A}$). The second part of this time can be bounded by $\mathcal{O}(q_{\mathsf{H}_2} N)$, and the third part can be bounded by $\mathcal{O}(q_{\mathsf{S}} N)$, where N is the maximum length of each H_2 query and the final forgery. However, we deliberately avoid doing so. While q_{H_2}, q_{S}, and N are all bounded by t, there is no guarantee that their products are still smaller than t. Therefore, our bound may

be asymptotically better. In the ROM, the reduction's running time is approximately bounded by $2Nt$ [8]. We can see that we remove not only the quadratic loss but also a factor N in the AGM.

Remark 2 (if H_1 is also modeled as a random oracle). The proof keeps basically unchanged when the EUF-CMA game is also in the ROM, but we should consider the number of H_1 queries. If $\mathcal{F}$ queries H_1 at most q_{H_1} times, then $\mathcal{A}$ queries H_1 at most $q_{\mathsf{H}_1} + q_{\mathsf{S}}$ times. It makes one extra query for each signing query from $\mathcal{F}$ to obtain the pair $(\hat{s}_j, \hat{c}_j)$ and no extra query for the forgery. We have an argument analogous to the one for H_2. Precisely, in the final forgery, if $\mathcal{A}$ has never queried $\mathsf{H}_1(R_i, m_i)$ for any $i \in \{1, \ldots, n\}$, then $\mathcal{F}$ is virtually impossible to win (unless $X_i = g^0$, in which case $\mathsf{H}_1(R_i, m_i)$ is irrelevant).

Remark 3 (if $0 \notin \mathcal{H}_2$). We can obtain a slightly better bound if $0 \notin \mathcal{H}_2$. Recall the three conditions of event $\mathsf{AggElim}$. Since there exists $\beta_k^* \neq 0$, if $T[L, i]$ can not be 0, then the condition $\sum_{i \in \mathcal{I}} T[L, i]\beta_i^* = 0$ can only be true with $\mathcal{I}$ of size at least 2. This means that event $\mathsf{AggElim}$ happens with probability at most $1/|\mathcal{H}_2|$ for each two H_2 queries. It follows that

$$\varepsilon' \geq \varepsilon - \frac{q_{\mathsf{H}_2}/2 + 1}{|\mathcal{H}_2|}.$$

On the other hand, the original bound $\varepsilon' \geq \varepsilon - (q_{\mathsf{H}_2} + 1)/|\mathcal{H}_2|$ is closely matched by a forger that expects to eliminate X^* with 0 from H_2.

Remark 4 (fix $a_1 = 1$). We can slightly optimize $\mathsf{ASchnorr}$ by fixing $a_1 = 1$. This may reduce the verification time a bit and make the verification consistent for individual signatures and aggregate signatures. Namely, we can omit algorithm Vf and use AggVf to verify an individual signature as a special case of aggregate signatures. The security proof will be almost unchanged. A subtle difference is that when we fix $a_1 = 1$, we can not obtain the above improved bound even though $0 \notin \mathcal{H}_2$. The forger can expect to eliminate X^* with an L of length 2, while it only needs one H_2 query. Note that this small optimization is also secure without the AGM.[2]

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 produce an aggregation, which leads to the demand for incremental aggregation.

[2] Since we will not go deep in the security of $\mathsf{ASchnorr}$ in the ROM, we briefly explain this here. In the ROM, $\mathcal{A}$ runs $\mathcal{F}$ twice. If $X_k = X^*$ and m_k is fresh, it makes a_k different in the two executions of $\mathcal{F}$ to extract a forged individual signature on m_k. While we fix $a_1 = 1$ in the scheme, $\mathcal{A}$ still keeps an internal a_1 but divides $a_2, \ldots, a_n$ all by a_1 as the returned values from H_2. Thus, $\mathcal{A}$ can still "make a_1 different in two executions".

However, 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 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 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 IASchnorr, described in Fig. 2. 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 Schnorr (i.e., algorithms KGen, Sign, and Vf) is unchanged, so Fig. 2 only describes algorithms IncrAgg and AggVf. The aggregate algorithm Agg can be seen as a special case of IncrAgg when $n = 0$.

4.2 Security

We prove almost the same security result for IASchnorr as ASchnorr in the AGM+ROM.

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* 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 Schnorr with*

$$t' = \mathcal{O}(t)$$

and

$$\varepsilon' \geq \varepsilon - \frac{q_{\mathsf{H}_2} + 1}{|\mathcal{H}_2|}.$$

Proof. This proof is almost identical to the proof of Theorem 1 up to event AggElim. In this proof, we say event AggElim happens if there exists

$$L_n = \{(R_1, X_1, m_1), \ldots, (R_n, X_n, m_n)\}$$

satisfying the following conditions:

- Let $\mathcal{I}$ be the set of those $i \in \{1, \ldots, n\}$ satisfying $R_i X_i^{c_i} \neq g^0$, where $c_i = \mathsf{H}_1(R_i, m_i)$. Let

$$L_i = \{(R_1, X_1, m_1), \ldots, (R_i, X_i, m_i)\}.$$

  The condition is that for each $i \in \mathcal{I}$, $T[L_i]$ has been defined.

$\mathsf{IncrAgg}(\{(\mathsf{pk}_1, m_1), \ldots, (\mathsf{pk}_n, m_n)\}, \tilde{\sigma}, \{(\mathsf{pk}_{n+1}, m_{n+1}, \sigma_{n+1}), \ldots, (\mathsf{pk}_{n'}, m_{n'}, \sigma_{n'})\})$

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**
  $(R_i, s_i) := \sigma_i$
  $L_i := \{(R_1, X_1, m_1), \ldots, (R_i, X_i, m_i)\}$
  $a_i := \mathsf{H}_2(L_i)$
$\tilde{s}' := \tilde{s} + \sum_{i=n+1}^{n'} a_i s_i$
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** $X_i := \mathsf{pk}_i$
$(\{R_1, \ldots, R_n\}, \tilde{s}) := \tilde{\sigma}$
$a_1 := 1$; $c_1 := \mathsf{H}_1(R_1, m_1)$
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. 2. Algorithms IncrAgg and AggVf of 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 IncrAgg.

- 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 $\beta_k^* \neq 0$.
- $\sum_{i \in \mathcal{I}} T[L_i] \beta_i^* = 0$.

Same as in the proof of Theorem 1, if $\mathcal{F}$ wins, but $\mathcal{A}$ does not win by extracting a forged signature, does not abort because of undefined table items, and also does not win by directly computing the discrete logarithm of X^*, then AggElim must happen.

AggElim happens with probability at most $1/|\mathcal{H}_2|$ for each L_n occurring in table T. The total probability of AggElim is thus upper bounded by $q_{\mathsf{H}_2}/|\mathcal{H}_2|$. The probability $\mathcal{A}$ aborts for undefined table items is also at most $1/|\mathcal{H}_2|$. It

follows that

$$\varepsilon' \geq \varepsilon - \frac{q_{\mathsf{H}_2} + 1}{|\mathcal{H}_2|}.$$

We can bound the running time of $\mathcal{A}$ as in the proof of Theorem 1. □

Remark 5 (comparison). The security loss of IASchnorr compared with ASchnorr is subtle, since Theorem 1 and Theorem 2 give the same bound. Here we explain the differences.

For ASchnorr, the probability bound $\varepsilon' \geq \varepsilon - (q_{\mathsf{H}} + 1)/|\mathcal{H}_2|$ can only be matched by a forger that tries to obtain 0 from H_2. This can be easily prevented by a range $\mathcal{H}_2$ excluding 0. The forger thus has to eliminate X^* with a list of at least two individual signatures. Hence, we achieve a better bound $\varepsilon' \geq \varepsilon - (q_{\mathsf{H}}/2 + 1)/|\mathcal{H}_2|$.

In contrast, for IASchnorr, a forger can match the bound even if $0 \notin \mathcal{H}_2$. When a forger fails to eliminate X^* with L_n, it can just append L_n by one more individual signature or replace the last one in it. In ASchnorr the forger needs one H_2 query for each signature in the modified list, while in IASchnorr it only needs one more query for another opportunity to eliminate X^*. We hence can not obtain the improved bound for IASchnorr.

Remark 6 (another perspective). Here we provide another perspective to understand IASchnorr. We can consider we get IASchnorr by modifying ASchnorr in two steps. First, we fix $a_1 = 1$. Second, we require the aggregation to be performed one by one. The first modification removes redundant operations, and the second removes ambiguity. From this perspective, we can better understand the security of IASchnorr. The first modification keeps the security in the ROM, as we discussed at the end of Section 3. The second modification is a benefit the AGM brings. In the AGM+ROM, the security of ASchnorr is tight, which permits the one-by-one aggregation. By contrast, in merely the ROM, one-by-one aggregation forces the reduction to rewind the forger at most n times for an aggregation of n signatures and hence exponentially expands the security loss to an unacceptable level.

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

So far, we see neither a tightly secure half-aggregate scheme for Schnorr signatures nor a (maybe non-tightly) provably secure sequential half-aggregate scheme for Schnorr signatures in the ROM without the AGM. By contrast, ASchnorr is non-tightly secure in the ROM. ASchnorr and IASchnorr both lack security proofs in the ROM when serving as an SAS scheme.

In this section, we present SASchnorr, an SAS scheme based on Schnorr signatures, and reduces its security tightly to that of Schnorr in the ROM. We also propose a new security model for SASchnorr together with justification. SASchnorr is provably secure in both the new security model and in the original model given in [16].

5.1 Scheme Discription

$\mathsf{SeqSign}(\{(\mathsf{pk}_1, m_1), \ldots, (\mathsf{pk}_{n-1}, m_{n-1})\}, \tilde{\sigma}_{n-1}, \mathsf{sk}_n, m_n)$

for $i = 1, \ldots, n-1$ **do** $X_i := \mathsf{pk}_i$
$(\tilde{R}_{n-1}, \{s_1, \ldots, s_{n-1}\}) := \tilde{\sigma}_{n-1}$
$x_n := \mathsf{sk}_n$; $X_n := g^{x_n}$
$r_n \leftarrow\$ \mathbb{Z}_p$; $R_n := g^{r_n}$
$\tilde{R}_n = \tilde{R}_{n-1} \cdot R_n$
$c_n := \mathsf{H}(\tilde{R}_n, X_n, m_n, s_{n-1}, n)$
$s_n := r_n + c_n x_n$
return $\tilde{\sigma}_n := (\tilde{R}_n, \{s_1, \ldots, s_n\})$

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

for $i = 1, \ldots, n$ **do** $X_i := \mathsf{pk}_i$
$(\tilde{R}_n, \{s_1, \ldots, s_n\}) := \tilde{\sigma}_n$
$c_n := \mathsf{H}(\tilde{R}_n, X_n, m_n, s_{n-1}, n)$
if $n = 1$ **then**
  return $[\![g^{s_1} = \tilde{R}_1 X_1^{c_1}]\!]$
else
  $R_n := g^{s_n} / X_n^{c_n}$
  $\tilde{R}_{n-1} := \tilde{R}_n / R_n$
  $\tilde{\sigma}_{n-1} := (\tilde{R}_{n-1}, \{s_1, \ldots, s_{n-1}\})$
  return $\mathsf{Vf}(\{(\mathsf{pk}_1, m_1), \ldots, (\mathsf{pk}_{n-1}, m_{n-1})\}, \tilde{\sigma}_{n-1})$

Fig. 3. 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. 1. We define s_0 as always 0.

We describe SASchnorr in Fig. 3. 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. In Fig. 3, we write algorithm Vf in a recursive way. The verification of an n-long aggregation is just computing $\tilde{R}_{n-1}$ and then recursively calling Vf on the $(n-1)$-long prefix.

Note that a signer can simply hash all relevant things, namely

$$\{(X_1, m_1), \ldots, (X_n, m_n)\}, (\tilde{R}_n, \{s_1, \ldots, s_{n-1}\})$$

to get its challenge c_n. In Fig. 3, 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 signers can keep their messages secret between each other, and 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 [16] for SAS schemes, we analyze the security of SASchnorr in a new model. Note that the signature produced by $\mathsf{SeqSign}$ 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.

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 aggreation, 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. See more detailed discussion in Remark 9. 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.

We make some comparisons between our new security model and the original model defined in [16]. 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, our model is incomparable with the original model in the success condition of attack.

In more details, we start with the original model and consider our model as the result of a three-step modification, where two steps strengthen the model while one weakens it. In the original model, the adversary makes a signing query in the form of $\textsc{Sign}(\{(X_1, m_1), \ldots, (X_{n-1}, m_{n-1})\}, m_n, \tilde{\sigma}_{n-1})$, where $\tilde{\sigma}_{n-1} = (\tilde{R}_{n-1}, s_1, \ldots, s_{n-1})$, in order to obtain a subsequent signature $\tilde{\sigma}_n$. The oracle can verify $\tilde{\sigma}_{n-1}$ and return $\tilde{\sigma}_n$ only if $\tilde{\sigma}_{n-1}$ is valid. The first step is to forbid the oracle to do so, which strengthens the model.

However, after removing the verification of intermediate aggregation, the signing oracle's behaviors only depend on $\tilde{R}_{n-1}$, s_{n-1}, and m_n. This gives rise to a type of trivial attack as follows: querying $\textsc{Sign}((X'_1, m'_1), m_2, \sigma_1)$ with arbitrary m'_1 and X'_1 to get a forged aggregate signature on m_1, m_2 under X_1, X^*. Hence, our second step is to remove the redundant arguments of the signing oracle (i.e., $X_1, \ldots, X_{n-1}, m_1, \ldots, m_{n-1}$, and $s_1, \ldots, s_{n-2}$), leaving only $\tilde{R}_n$, s_{n-1}, m_n, and n. The success condition thus requires the adversary not to query $\textsc{Sign}(\cdot, \cdot, m_k, k)$ for some k such that $X_k = X^*$. This makes the condition stricter and excludes the above trivial forgeries.

Finally, the third step loosens the success condition in another dimension. We can allow the adversary to query $\textsc{Sign}(\cdot, s_{k-1}, m_k, k)$ with s_{k-1} that is different from the one appeared in the forgery. This strengthens the model in a way that the original model does not consider. In particular, the original model requires the set of $X_1, \ldots, X_{k-1}, m_1, \ldots, m_k$ has not been queried, no matter whether $\tilde{\sigma}_{k-1}$ queried together is different from the one corresponds to the final forgery. By contrast, some differences in $\tilde{\sigma}_{k-1}$ can make the forgery non-trivial in our 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 [5,20], 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*

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

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. 3, we define s_0 as always 0. Moreover, we define $\tilde{R}_0$ as g^0, 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 $\Pr[\mathsf{Acc}_\mathcal{F} \mid \mathsf{UnDef}] \leq (q_\mathsf{H} + q_\mathsf{S} + 1)/|\mathcal{H}|$.

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*. Lemmas 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 $\Pr[\mathsf{BadOrder} \mid \neg\mathsf{Dup}] \leq (\sum_{i=1}^{\infty} q_i q_{i+1})/|\mathcal{H}|$.

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 with 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

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

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

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

We defer the proofs of Lemmas 4 to 6 to the end of this section.

Remark 7 (use bit-wise XOR instead of multiplication). Note that in SASchnorr we do not rely on group homomorphism in the verification (opposite to the case in ASchnorr and IASchnorr). Instead, all the individual commitments will be recovered, and we verify an equality for each of them. The method of aggregating the commitments is only required to be reversible. Hence, we can actually use bit-wise XOR rather than group multiplication to aggregate the commitments, which will slightly optimize the scheme.

Remark 8 (binding security). The work [8] introduced the binding security of AS schemes. It means that an aggregate signature is bound to a set of public keys and messages, in the sense that it is hard to find two sets of public keys and messages w.r.t. the same valid aggreate signature. As the related public keys and messages are all hashed to generate the coefficients in ASchnorr and IASchnorr, the two scheme obviously satisfy the binding security. Here, we clarify that SASchnorr also enjoys the binding security, though with only partial information being hashed. Specifically, Lemma 5 and Lemma 6 actually imply the binding security. Lemma 6 guarantees that the table items corresponding to a valid aggregation were defined in order, so every such item has its index.

Therefore, a valid signature for two different sets of public keys and messages means that the Dup event defined in Lemma 5 happens somewhere among those items.

Remark 9 (Security in the original security model). As mentioned, if we let the signer verify the previous aggregation σ_{k-1} (with $k \geq 2$),[3] then our scheme is also provably secure in the original security model [16]. In the security proof, we need the table item $T[\tilde{R}_k, X^*, m_k, s_{k-1}, k]$ (corresponding to the forgery) not to be assigned in a signing query. With the security result we already have, we only need to consider one more case in the original model: $\mathcal{F}$ has made a signing query with previous public keys and messages (X_1, m_1), ..., (X_{k-1}, m_{k-1}) and aggregation $(\tilde{R}_{k-1}, \{s_1, \ldots, s_{k-1}\})$ on m_k where $T[\tilde{R}_k, X^*, m_k, s_{k-1}, k]$ was assigned; and $\mathcal{F}$ outputs a forgery with different (X'_1, m'_1), ..., (X'_{k-1}, m'_{k-1}). This immediately contradicts the binding security just clarified above.

There is another way to make the scheme secure in the original model: simply hashing all the previous public keys and messages to generate the challenge, i.e., letting $c_n = \mathsf{H}(\tilde{R}_n, X_1, \ldots, X_n, m_1, \ldots, m_n)$. In this case, the signer does not need to verify the previous aggregation any longer. As required in the original model, the set of public keys and messages corresponding to the forgery has not been queried to the signing oracle. This immediately implies the table item corresponding to the forgery was not assigned in a signing query.

Proofs of Lemmas 4 to 6.

Proof (Lemma 4). Let $f(j)$ be the bound for a fixed j. Namely, for any forgery of length j, if $T[\tilde{R}_j, X_j, m_j, s_{j-1}, j]$ is undefined, algorithm Vf determines the forgery to be valid with probability at most $f(j)$. It is sufficient for us to show that $(q_{\mathsf{H}} + q_{\mathsf{S}} + 1)/|H|$ is the common bound of $f(j)$ for all $j \in \mathbb{N}$.

First we consider $f(1)$. If $j = 1$, then Vf returns 1 *iff* $g^{s_1} = \tilde{R}_1 X_1^{c_1}$, with $c_1 = T[\tilde{R}_1, X_1, m_1, 0, 1]$. Here, c_1 is uniformly chosen from $\mathcal{H}$ after s_1, $\tilde{R}_1$, X_1 are fixed, and only at most one value of c_1 is sufficient. Hence, we have $f(1) \leq 1/|\mathcal{H}|$.

Now we consider $f(j)$ for $j \geq 2$. When $j \geq 2$,

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

iff

$$\mathsf{Vf}(\{(X_1, m_1), \ldots, (X_{j-1}, m_{j-1})\}, (\tilde{R}_{j-1}, \{s_1, \ldots, s_{j-1}\})) = 1,$$

where $\tilde{R}_{j-1} = \tilde{R}_j / (g^{s_j} / X_j^{c_j})$ for $c_j = T[\tilde{R}_j, X_j, m_j, s_{j-1}, j]$. Note that there are at most q_{j-1} items with the last argument being $j-1$ in T. Since c_j is uniformly chosen from $\mathcal{H}$ at last, $\tilde{R}_{j-1}$ is uniformly distributed on a set of size $|\mathcal{H}|$. Hence, the probability that $T[\tilde{R}_{j-1}, X_{j-1}, m_{j-1}, s_{j-2}, j-1]$ is defined is at

[3] This implies that, as in the traditional case of sequential aggregate signatures, the whole aggregation $(\tilde{R}_{k-1}, \{s_1, \ldots, s_{k-1}\})$ and all the related information $\{m_1, \ldots, m_{k-1}, \mathsf{pk}_1, \ldots, \mathsf{pk}_{k-1}\}$ need to be sent to the signer.

most $q_{j-1}/|\mathcal{H}|$. On the condition that it is undefined, the probability that Vf returns 1 is bounded by $f(j-1)$. Therefore, we have

$$f(j) \leq 1 - (1 - \frac{q_{j-1}}{|\mathcal{H}|})(1 - f(j-1)) \leq \frac{q_{j-1}}{|\mathcal{H}|} + f(j-1).$$

Combining with that $f(1) \leq 1/|\mathcal{H}|$, we have

$$f(j) \leq \frac{1 + \sum_{i=1}^{j-1} q_i}{|\mathcal{H}|} \leq \frac{q_{\mathsf{H}} + q_{\mathsf{S}} + 1}{|\mathcal{H}|}. \qquad \square$$

Proof (Lemma 5). Define $\mathsf{Dup(j)}$ as the event that there exist two different table items with the last arguments being both j and equal indices. It holds that

$$\Pr[\mathsf{Dup}] \leq \sum_{i=1}^{\infty} \Pr[\mathsf{Dup(i)}].$$

It is sufficient for us to separately bound the terms on the right.

First we consider $\mathsf{Dup(1)}$. For every table item $c = T[\tilde{R}_1, X_1, m_1, 0, 1]$, we have

$$I[\tilde{R}_1, X_1, m_1, 0, 1] = \tilde{R}_1 X_1^c.$$

If the item is defined when $\mathcal{A}$ handles a signing query, the index is uniformly distributed on $\mathbb{G}$. If it is defined when $\mathcal{A}$ handles a hash query, since c is uniformly chosen from $\mathcal{H}$ after $\tilde{R}_1$ and X_1 are fixed, the index is uniformly distributed on a set of size $|\mathcal{H}|$. Therefore, for every pair of items with the last arguments both being 1, their indices collide with probability at most $1/|\mathcal{H}|$. The number of such pairs is $q_1(q_1-1)/2$, and it follows that

$$\Pr[\mathsf{Dup(1)}] \leq \frac{q_1(q_1-1)}{2|\mathcal{H}|}.$$

Now consider $\mathsf{Dup(i)}$ for $i \geq 2$. For every table item $c = T[\tilde{R}_i, X_i, m_i, s_{i-1}, i]$ with $i \geq 2$, its index $I[\tilde{R}_i, X_i, m_i, s_{i-1}, i]$ can be defined only when there exists a unique item $T[\tilde{R}_{i-1}, X_{i-1}, m_{i-1}, s_{i-2}, i-1]$ with index being $g^{s_{i-1}}$. The index is then set to

$$I[\tilde{R}_i, X_i, m_i, s_{i-1}, i] = (\tilde{R}_i / \tilde{R}_{i-1}) X_i^c.$$

Similarly, the index is also distributed on $\mathbb{G}$ or a set of size $|\mathcal{H}|$. We thus have

$$\Pr[\mathsf{Dup(i)}] \leq \frac{q_i(q_i-1)}{2|\mathcal{H}|}.$$

Put the above results together, we have

$$\Pr[\mathsf{Dup}] \leq \sum_{i=1}^{\infty} \frac{q_i(q_i-1)}{2|\mathcal{H}|} \leq \frac{\sum_{i=1}^{\infty} q_i^2}{2|\mathcal{H}|}. \qquad \square$$

Proof (Lemma 6). Define $\mathsf{BadOrder(j)}$ as the event that there exists a valid chain of length j that is not defined in order. It holds that

$$\Pr[\mathsf{BadOrder} \mid \neg\mathsf{Dup}] \leq \Pr[\mathsf{BadOrder(2)} \mid \neg\mathsf{Dup}] + \sum_{i=3}^{\infty} \Pr[\mathsf{BadOrder(i)} \mid \neg\mathsf{Dup} \wedge \neg\mathsf{BadOrder(i-1)}].$$

We bound the terms on the right separately.

First we consider $\mathsf{BadOrder(2)}$. For every pair of items

$$c_1 = T[\tilde{R}_1, X_1, m_1, 0, 1] \quad \text{and} \quad c_2 = T[\tilde{R}_2, X_2, m_2, s_1, 2],$$

if they are not defined in order, i.e., c_1 is defined later, then $(\tilde{R}_2/\tilde{R}_1)X_1^{c_1}$ is uniformly distributed on $\mathbb{G}$ or a set of size $|\mathcal{H}|$ (depending on whether c_1 is defined in a signing query or a hash query, as in the proof of Lemma 5), and determined after s_1 is fixed. On the other hand, it is necessary that $g^{s_1} = (\tilde{R}_2/\tilde{R}_1)X_1^{c_1}$ for such a pair to be a valid chain of length 2. Hence, every pair that are not defined in order forms a 2-long valid chain with probability at most $1/|\mathcal{H}|$. There are at most $q_1 q_2$ such pairs in T, so in total we have

$$\Pr[\mathsf{BadOrder(2)} \mid \neg\mathsf{Dup}] \leq \frac{q_1 q_2}{|\mathcal{H}|}.$$

Now we consider $\mathsf{BadOrder(i)}$, conditioned on that $\mathsf{BadOrder(i-1)}$ does not happen. Consider a valid chain

$$\begin{aligned} c_1 &= T[\tilde{R}_1, X_1, m_1, 0, 1], \\ &\vdots \\ c_i &= T[\tilde{R}_i, X_i, m_i, s_{i-1}, i]. \end{aligned}$$

Since $\mathsf{BadOrder(i-1)}$ does not happen, we know $c_1, \ldots, c_{i-1}$ are defined in order. Hence, if the items in this chain are not defined in order, c_{i-1} must be the last one to be defined.

We consider every pair of items

$$c_{i-1} = T[\tilde{R}_{i-1}, X_{i-1}, m_{i-1}, s_{i-2}, i-1] \quad \text{and} \quad c_i = T[\tilde{R}_i, X_i, m_i, s_{i-1}, i]$$

with the last arguments being $i-1$ and i respectively. For them to be in a valid chain of length i, c_{i-1} must be the last item of a valid chain of length $i-1$. Conditioned on that $\mathsf{BadOrder(i-1)}$ does not happen, the $(i-1)$ items in the valid chain are defined in order. Moreover, since Dup does not happen, the indices of these $(i-1)$ items are also defined.[4] Hence, when c_{i-1} is going to be defined, there exists a unique item with an index being $g^{s_{i-2}}$, which is exactly

$$c_{i-2} = T[\tilde{R}_{i-2}, X_{i-2}, m_{i-2}, s_{i-3}, i-2],$$

[4] This can be precisely shown by the induction we make in the proof of Theorem 3.

the $(i-2)$-th item in the chain. For these items to form a valid chain, it must hold that

$$g^{s_{i-1}} = (\tilde{R}_{i-1}/\tilde{R}_{i-2})X_{i-1}^{c_{i-1}}.$$

On the other hand, $(\tilde{R}_{i-1}/\tilde{R}_{i-2})X_{i-1}^{c_{i-1}}$ is uniformly distributed on $\mathbb{G}$ or a set of size $|\mathcal{H}|$ and determined after s_{i-1} is fixed. Hence, this pair occurs in a i-long valid chain with probability at most $1/|\mathcal{H}|$. There are at most $q_{i-1}q_i$ such pairs. It follows that

$$\Pr[\mathsf{BadOrder(i)} \mid \neg\mathsf{Dup} \wedge \neg\mathsf{BadOrder(i-1)}] \leq \frac{q_{i-1}q_i}{|\mathcal{H}|}.$$

Put the above results together, and we have

$$\Pr[\mathsf{BadOrder} \mid \neg\mathsf{Dup}] \leq \frac{\sum_{i=1}^{\infty} q_i q_{i+1}}{|\mathcal{H}|}. \qquad \square$$

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 (Oct 2010). https://doi.org/10.1145/1866307.1866360
2. Alper, H.K., 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, Heidelberg, Virtual Event (Aug 2021). https://doi.org/10.1007/978-3-030-84242-0_7
3. Bellare, M., Dai, W.: Chain reductions for multi-signatures. Cryptology ePrint Archive, Report 2021/404 (2021), https://ia.cr/2021/404
4. Bellare, M., Namprempre, C., Neven, G.: Unrestricted aggregate signatures. In: Arge, L., Cachin, C., Jurdzinski, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 411–422. Springer, Heidelberg (Jul 2007). https://doi.org/10.1007/978-3-540-73420-8_37
5. Bellare, M., Neven, G.: Multi-signatures in the plain public-key model and a general forking lemma. In: Juels, A., Wright, R.N., De Capitani di Vimercati, S. (eds.) ACM CCS 2006. pp. 390–399. ACM Press (Oct / Nov 2006). https://doi.org/10.1145/1180405.1180453
6. 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 (May 2003). https://doi.org/10.1007/3-540-39200-9_26
7. Boneh, D., Kim, S.: One-time and interactive aggregate signatures from lattices (2021), https://crypto.stanford.edu/~skim13/agg_ots.pdf
8. 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, Heidelberg (May 2021). https://doi.org/10.1007/978-3-030-75539-3_24
9. Drijvers, M., Edalatnejad, K., Ford, B., Kiltz, E., Loss, J., Neven, G., Stepanovs, I.: On the security of two-round multi-signatures. In: 2019 IEEE Symposium on Security and Privacy. pp. 1084–1101. IEEE Computer Society Press (May 2019). https://doi.org/10.1109/SP.2019.00050

10. 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, Heidelberg (Aug 2018). https://doi.org/10.1007/978-3-319-96881-0_2
11. 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, Heidelberg (May 2020). https://doi.org/10.1007/978-3-030-45724-2_3
12. 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 (Apr 2006). https://doi.org/10.1007/11745853_17
13. Goldwasser, S., Micali, S., Rivest, R.L.: A digital signature scheme secure against adaptive chosen-message attacks. SIAM Journal on Computing **17**(2), 281–308 (Apr 1988)
14. 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 (Feb / Mar 2013). https://doi.org/10.1007/978-3-642-36362-7_26
15. 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 (May / Jun 2006). https://doi.org/10.1007/11761679_28
16. Lysyanskaya, A., Micali, S., Reyzin, L., Shacham, H.: Sequential aggregate signatures from trapdoor permutations. In: Cachin, C., Camenisch, J. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 74–90. Springer, Heidelberg (May 2004). https://doi.org/10.1007/978-3-540-24676-3_5
17. Maurer, U.M.: Abstract models of computation in cryptography (invited paper). In: Smart, N.P. (ed.) 10th IMA International Conference on Cryptography and Coding. LNCS, vol. 3796, pp. 1–12. Springer, Heidelberg (Dec 2005)
18. Maxwell, G., Poelstra, A., Seurin, Y., Wuille, P.: Simple schnorr multi-signatures with applications to bitcoin. Des. Codes Cryptogr. **87**(9), 2139–2164 (2019). https://doi.org/10.1007/s10623-019-00608-x
19. Nick, J., Ruffing, T., Seurin, Y.: MuSig2: Simple two-round Schnorr multi-signatures. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part I. LNCS, vol. 12825, pp. 189–221. Springer, Heidelberg, Virtual Event (Aug 2021). https://doi.org/10.1007/978-3-030-84242-0_8
20. Pointcheval, D., Stern, J.: Security proofs for signature schemes. In: Maurer, U.M. (ed.) EUROCRYPT'96. LNCS, vol. 1070, pp. 387–398. Springer, Heidelberg (May 1996). https://doi.org/10.1007/3-540-68339-9_33
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, Heidelberg, Virtual Event (Aug 2021). https://doi.org/10.1007/978-3-030-84242-0_9
22. Shoup, V.: Lower bounds for discrete logarithms and related problems. In: Fumy, W. (ed.) EUROCRYPT'97. LNCS, vol. 1233, pp. 256–266. Springer, Heidelberg (May 1997). https://doi.org/10.1007/3-540-69053-0_18