The many faces of Schnorr

Victor Shoup
DFINITY

October 6, 2023

Abstract

Recently, a number of highly optimized threshold signing protocols for Schnorr signatures have been proposed. A key feature of these protocols is that they produce so-called "presignatures" in a "offline" phase (using a relatively heavyweight, high-latency subprotocol), which are then consumed in an "online" phase to generate signatures (using a relatively lightweight, low-latency subprotocol). The idea is to build up a large cache of presignatures in periods of low demand, so as to be able to quickly respond to bursts of signing requests in periods of high demand. Unfortunately, it is well known that using such presignatures naively leads to subexponential attacks. Thus, any protocols based on presignatures must mitigate against these attacks.

One such notable protocol is FROST, which provides security even with an unlimited number of presignatures; moreover, assuming unused presignatures are available, signing requests can be processed concurrently with minimal latency. Unfortunately, FROST is not a robust protocol, at least in the asynchronous communication model (arguably the most realistic model for such a protocol). Indeed, a single corrupt party can prevent any signatures from being produced. Recently, a protocol called ROAST was developed to remedy this situation. Unfortunately, ROAST is significantly less efficient that FROST (each signing request runs many instances of FROST concurrently).

A more recent protocol is SPRINT, which provides robustness without synchrony assumptions, and actually provides better throughput than FROST. Unfortunately, SPRINT is only secure in very restricted modes of operation. Specifically, to avoid a subexponential attack, only a limited number of presignatures may be produced in advance of signing requests, which somewhat defeats the purpose of presignatures.

Our main new result is to show how to securely combine the techniques used in FROST and SPRINT, allowing one to build a threshold Schnorr signing protocol that (i) is secure and robust without synchrony assumptions (like SPRINT), (ii) provides security even with an unlimited number of presignatures, and (assuming unused presignatures are available) signing requests can be processed concurrently with minimal latency (like FROST), (iii) achieves high throughput (like SPRINT), and (iv) achieves optimal resilience.

Besides achieving this particular technical result, one of our main goals in this paper is to provide a *unifying framework* in order to better understand the techniques used in various protocols. To that end, we attempt to isolate and abstract the main ideas of each protocol, stripping away superfluous details, so that these ideas can be more readily combined and implemented in different ways. More specifically, we generally avoid talking about distributed protocols at all, and rather, we examine the security of the ordinary, non-threshold Schnorr scheme in "enhanced" attack modes that correspond to attacks on various types of threshold signing protocols.

Another one of our goals is to carry out a security analysis of these enhanced attack modes in the Generic Group Model (GGM), sometimes in conjunction with the Random Oracle Model (ROM). Despite the limitations of these models, we feel that giving security proofs in the GGM or GGM+ROM provides useful insight into the concrete security of the various enhanced attack modes we consider.

1 Introduction

1.1 Background

Recently, a number of highly optimized threshold signing protocols for Schnorr signatures have been proposed. Threshold signing protocols are useful in that they can provide both *security* and *robustness*, even if some of the parties on the signing committee are corrupt — *security*, in the sense that even if a bounded number parties are corrupt, they cannot forge a signature, and *robustness*, in the sense that even if a bounded number parties are corrupt, they cannot stop the honest parties from producing signatures.

Recall that for the Schnorr signature scheme, the public key is of the form $\mathcal{D} = d\mathcal{G}$, where $d \in \mathbb{Z}_q$ is the secret key and $\mathcal{G}$ is a generator for a group E of prime order q (which we write here using additive notation to reflect the fact that E is nowadays typically an elliptic curve). A signature on a message m is a pair $(\mathcal{R}, z) \in E \times \mathbb{Z}_q$, where $z\mathcal{G} = \mathcal{R} + h\mathcal{D}$ and $h \in \mathbb{Z}_q$ is a hash of $\mathcal{R}$ and m (and typically $\mathcal{D}$ as well). To generate such a signature in the non-threshold setting, the signer generates $r \in \mathbb{Z}_q$ at random, computes $\mathcal{R} \leftarrow r\mathcal{G}$ and $z \leftarrow r + hd$, and outputs the signature $(\mathcal{R}, z)$.

In the threshold setting, we have n parties on a signing committee, some of which may be corrupt, and a certain threshold number of parties is needed to sign a message (and assuming this threshold is high enough, some honest party must actually participate in signing the message). The usual technique used is Shamir secret sharing, so that each of the n parties obtains $\mathcal{D}$ and its share of d. To generate a public-key/secret-key pair, some kind of distributed key generation (DKG) protocol must be executed, which can be rather expensive.

To sign an individual message, in principle, the same DKG protocol could be used to generate the "ephemeral" public-key/secret-key pair $(\mathcal{R}, r)$, where each party obtains $\mathcal{R}$ and its share of r. Once this is done, each party can locally compute its share of the signature (since this is a linear operation), and then these shares can be revealed and combined to form a signature.

The problem with this approach is that an expensive DKG protocol must be run for each signing operation. To reduce this cost, a few optimizations have been considered.

One obvious optimization follows from the observation that the "ephemeral" public-key/secret-key pair $(\mathcal{R}, r)$ is completely independent of the message to be signed. Therefore, we could potentially use an "offline/online" strategy, in which we generate such ephemeral key pairs in an offline fashion, building up a cache of them in advance of actual signing requests. In this context, such an ephemeral public key is called a **presignature**. The idea is to build up a large cache of presignatures in periods of low demand, so as to be able to quickly respond to bursts of signing requests in periods of high demand. Note, however, that while computing presignatures in this way can improve *latency*, it does not improve *throughput*.

Unfortunately, using presignatures naively in this way breaks the security of Schnorr signatures. Indeed, the usual proof of security of ordinary, non-threshold Schnorr signatures relies in an essential way on the fact that the randomly generated group element $\mathcal{R}$ is not revealed before the request to sign m is given. Moreover, this is not just an artifact of the proof: there are actual subexponential attacks on signing protocols that use presignatures in this way [DEF+18] (which we review below).

To mitigate against these presignature attacks, the FROST protocol [KG20, CKM21] was introduced. FROST provides security even with an unlimited number of presignatures; moreover, assuming unused presignatures are available, signing requests can be processed concurrently with minimal latency. However, FROST is not a robust protocol. Indeed, a single corrupt party can prevent any signatures from being produced. Nevertheless, FROST does enjoy a property called *identifiable abort*, which allows misbehaving parties that prevent protocol termination to be identified and removed from the signing committee. The use of *identifiable aborts* in the context of threshold signatures is also found in the work of [GG20]. However, the notion of *identifiable aborts* only makes sense in a *synchronous communication setting*. Indeed, in an *asynchronous communication setting*, it is impossible to tell the difference between a party that is misbehaving by staying silent and a party that is just slow or temporarily disconnected from the rest of the parties. Thus, at least in an asynchronous communication setting, FROST does not provide robustness. This makes FROST unusable in distributed systems for which both security and robustness are required without synchrony assumptions. Indeed, for a protocol with parties distributed around the globe, synchrony assumptions seem quite unrealistic.

This limitation of FROST was highlighted in [RRJ$^+$22], who propose a new protocol called ROAST. To obtain robustness without synchrony assumptions, the ROAST protocol uses FROST (or any protocol with similar security properties) as a subprotocol, running it concurrently $O(n)$ times per signing request. Thus, while ROAST achieves robustness without synchrony assumptions, this comes at a significant performance cost.

More recently, the SPRINT protocol [BHK$^+$23] was proposed, which aims to achieve security and robustness without synchrony assumptions, and to do so while actually providing *better throughput* than FROST by using improved presignature generation protocols based on batch randomness extraction techniques (an idea that goes back to [HN06]). While SPRINT does achieve this goal, it is only secure in very restricted modes of operation. Specifically, only a limited number of presignatures may be generated in advance of signing requests, which somewhat defeats the purpose of presignatures. Indeed, the security theorem in [BHK$^+$23] only applies to a chosen message attack in which a single, fixed-size batch of presignatures is generated, which are subsequently used to sign a corresponding batch of messages. As we discuss below in Section 4.1, if many such batches of presignatures are generated in advance, the same subexponential attacks mentioned above can be used on SPRINT.

1.2 Our contributions

On a purely technical level, our main new result is to show how the batch randomness extraction technique used in SPRINT can be securely combined with the main technical idea of FROST for making presignatures safe, thus allowing one to build a threshold Schnorr signing protocol that

- is secure and robust without synchrony assumptions (like SPRINT),
- provides security even with an unlimited number of presignatures, and (assuming unused presignatures are available) signing requests can be processed concurrently with minimal latency (like FROST),

- achieves high throughput (like SPRINT), and
- achieves optimal resilience (i.e., tolerates up to $f < n/3$ corrupt parties).

Note that one variant of SPRINT also considers so-called "packed" secret sharing, which can give even higher throughput at the cost of suboptimal resilience.[1] We do not consider this type of protocol here, although our techniques and analysis may well apply.

Besides achieving this particular technical result, one of our main goals in this paper is to provide a *unifying framework* in order to better understand the techniques used in various papers. Indeed, the analyses in the papers [KG20, CKM21, BHK$^+$23] are quite targeted to very specific protocols (although [CKM21] makes some attempt to be a bit more modular), and it is not clear how ideas from one protocol can be used in a different context. Here, we attempt to isolate and abstract the main ideas of each protocol, stripping away superfluous details, so that these ideas can be more readily combined and implemented in different ways. Indeed, our approach is much more like that of [GS21], in that we try to avoid talking about distributed protocols at all, and rather, we examine the security of the ordinary, non-threshold Schnorr scheme in "enhanced" attack modes that correspond to attacks on various types of threshold signing protocols (which may use presignatures, for example) — the details of these threshold protocols do not matter that much, so long as they are designed in a reasonably modular way so as to satisfy certain natural security properties. Because of this, our results can be used to easily analyze protocols that work very differently from SPRINT, such as those in the more recent work of [GS23].[2]

Another one of our goals to carry out a security analysis of these enhanced attack modes in the Generic Group Model (GGM). Such an analysis in the GGM has already been done by [NSW09] for the basic attack mode on Schnorr, but not for any of the enhanced attack modes we consider here. The analysis in [NSW09] proves the security of Schnorr for the basic attack mode in the GGM under specific preimage assumptions on the underlying hash function. In fact, we reprove the results in [NSW09]. Our main reason for this is that we want to establish a general framework for proving results on various Schnorr attack modes. This framework is very similar to that introduced in [GS21], in which at attack in the GGM is reduced to a purely "symbolic" attack that allows for a much more modular and intuitive security analysis. We actually prove a bit more than what is proved in [NSW09], observing that if we use both the GGM and Random Oracle Model (ROM), where the hash function is modeled as a random oracle, we get a very tight security bound: any adversary that makes at most N oracle queries (to either the signing, group, or random oracles) forges a signature with probability $O(N^2/q + N/M)$. Here, M is the size of the output space of the hash function. Note that this tight security bound is not new: it was proved already in [BL19].

We feel that giving security proofs in the GGM or GGM+ROM provides useful insight into the practical security of the various enhanced attack modes we consider. For example,

[1] "Packed" secret sharing is a technique introduced in [FY92], in which many secrets are packed into a single Shamir secret sharing, and is not to be confused with "batched" secret sharing, in which many independent Shamir secret sharings are generated concurrently, as in [DN07], for example. Indeed, protocols that implement the strategies we outline here may very well use "batched" secret sharing.

[2] The original version of this paper preceded the paper [GS23], but has been subsequently updated to take advantage of some of the concepts discussed in that paper.

the attack modes that correspond to FROST and SPRINT have been analyzed in the literature in the ROM, with reductions to the one-more discrete logarithm problem (for FROST) or the discrete logarithm problem (for SPRINT). However, these reductions all go via the so-called "forking lemma" [PS96], which yields very "loose" security reductions. Even though security proofs in the GGM or GGM+ROM have limitations in the generality of attacks they consider, they also have value by giving a better understanding of concrete security against the types of attacks that are arguably most likely to be carried out in practice.

We give security proofs in the GGM+ROM for a number of enhanced attack modes, including those corresponding to FROST, SPRINT, and our new technique that combines the best of both FROST and SPRINT. In all cases, we find that the adversary's forging probability is still $O(N^2/q + N/M)$, just as for the basic attack mode. For several enhanced attack modes, we also give security proofs in the GGM under specific preimage assumptions on the hash function. Note that our analysis of our new technique combining FROST and SPRINT (which is covered in Section 4.3) is only done in the GGM+ROM. We speculate that a security proof in the ROM via a reduction to the one-more discrete logarithm problem should be possible — we leave that as an open problem.

In addition to all of the above, we also model **additive key derivation**. Here, when the adversary makes a signing query, he additionally specifies an additive tweak $e \in \mathbb{Z}_q$ to derive the effective public key as $\mathcal{D}' := \mathcal{D} + e\mathcal{G}$. This corresponds to using a scheme like BIP32 [Wui20] to derive subkeys from a master key. This type of key derivation is especially important in a threshold setting, as there is a significant cost to maintaining a secret key — for example, it will likely need to be reshared with regular frequency, both to achieve proactive security and to support membership changes to the signing committee. With additive key derivation, a signing committee can just maintain a single master key, and derive subkeys as necessary on behalf of individual external users (or "smart contracts" in a blockchain setting). Moreover, because of the simple additive nature of the key derivation, it is generally trivial to deal with these derived keys in a distributed computation. Not surprisingly, including the (effective) public key in the hash used to derive h is necessary and sufficient to obtain security proofs for all of the attack modes we consider.

2 Preliminaries

2.1 Schnorr Signatures

From now on, we consider the Schnorr signature scheme over an elliptic curve. Let E be an elliptic curve defined over $\mathbb{Z}_p$ and generated by a point $\mathcal{G}$ of prime order q, and let E^* be the set of points (x, y) on the curve excluding the point at infinity $\mathcal{O}$.

The secret key for ECDSA is a random $d \in \mathbb{Z}_q$, the public key is $\mathcal{D} = d\mathcal{G} \in E$. The scheme makes use of a hash function $H : \{0,1\}^* \to \mathbb{Z}_q$. The signing and verification algorithms are shown in Figure 1. Here, we assume a serialization function

$$\langle \cdot \rangle : E \to \{0,1\}^*$$

that is prefix-free and is 1-1 (as well as easy to compute and to invert).

```
Sign message m:
    $r \xleftarrow{\$} \mathbb{Z}_q,\ \mathcal{R} \leftarrow r\mathcal{G} \in E$
    $h \leftarrow H(\langle\mathcal{D}\rangle \,\|\, \langle\mathcal{R}\rangle \,\|\, m) \in \mathbb{Z}_q$
    $z \leftarrow r + hd$
    return the signature $(\mathcal{R}, z)$

Verify signature $(\mathcal{R}, z) \in E \times \mathbb{Z}_q$ on m:
    $h \leftarrow H(\langle\mathcal{D}\rangle \,\|\, \langle\mathcal{R}\rangle \,\|\, m) \in \mathbb{Z}_q$
    check that $z\mathcal{G} = \mathcal{R} + h\mathcal{D}$
```

Figure 1: Schnorr signing and verification algorithms

NOTE: The scheme presented in Figure 1 does not quite fully capture either BIP340 (the bitcoin version of Schnorr) or EdDSA — each have there own quirks. However, it seems reasonable to speculate that all of the results proved here can easily be adapted to those particular schemes.

2.2 Enhanced attack modes

In the basic attack game for signatures, the adversary makes a series of signing queries and then must forge a signature on some message that was not submitted as a signing query. This attack game needs to be modified in order to model attacks that can be carried out in the threshold setting. There are three variations to consider:

Presignatures. Here, the adversary instructs the challenger to generate presignatures $\mathcal{R}_1, \mathcal{R}_2, \ldots$, which are random elements of E that are given to the adversary. In a signing query, the adversary specifies the index k of an unused presignature and a message m_k; the challenger then signs m_k using $\mathcal{R}_k$.

This models the situation in the threshold setting where we do the expensive presignature computation in advance using a secure DKG protocol. Any secure DKG protocol may be used. For example, [GS22] provides fairly efficient DKG protocols that are secure and robust, with optimal resilience, in the asynchronous setting.

Biased presignatures. Here, when the adversary makes a signing query, in addition to specifying k and m_k, the adversary specifies a "bias" $(u_k, u'_k) \in \mathbb{Z}_q^* \times \mathbb{Z}_q$; the challenger then signs m_k using $\mathcal{R}'_k := u_k\mathcal{R}_k + u'_k\mathcal{G}$.

This models a common situation in the threshold setting where we utilize a simple DKG protocol in which each party securely distributes shares of an ephemeral secret key and publishes the corresponding ephemeral public key, after which a collection of these ephemeral keys is agreed upon and added together to obtain a presignature. This protocol is not a secure DKG, as the adversary can bias the result. Indeed, the adversary may use the values of the ephemeral public keys to influence the choice of his own secret keys and the choice of ephemeral keys to include in the agreed-upon collection.

This type of biasing was discussed in [GJKR07] in the synchronous communication setting, and in [GS22] in the asynchronous communication setting. In [GS22] it was shown, by means of a random self-reduction, that the effects of this biasing can be simply modeled as we have here. See also [GS23] for more context.

Additive key derivation. Here, when the adversary makes a signing query, he additionally specifies an additive tweak $e_k \in \mathbb{Z}_q$ to derive the effective public key as $\mathcal{D}'_k := \mathcal{D} + e_k\mathcal{G}$. With this modification, the notion of a forgery must also be appropriately modified, so that the forgery includes a tweak $e^* \in \mathbb{Z}_q$ in addition to a message m^*, and the forgery counts so long as $(m^*, e^*) \neq (m_k, e_k)$ for any (m_k, e_k) submitted as part of a signing query.

This corresponds to using a scheme like BIP32 [Wui20] to derive subkeys from a master key.

Additive key derivation can be considered either by itself, or in combination of one of the two variants above.

2.3 Proof techniques and known attacks

In the usual analysis of Schnorr, we model H as a random oracle. The main idea of the security proof is to reduce an attack on the signature scheme to an attack on the interactive identification scheme. In the latter attack, the adversary, playing the role of prover, may initiate many conversations with the challenger, who is playing the role of verifier. The adversary wins the attack game if he can make any of these verifiers accept.[3] To carry out this reduction, we program the random oracle, which allows us to (a) simulate signing queries and (b) translate the random challenges in the identification attack game into random oracle outputs in the signature attack game.

To simulate signing queries, when we get a message m to sign, we generate $z, h \in \mathbb{Z}_q$ at random, compute $\mathcal{R} \leftarrow z\mathcal{G} - h\mathcal{D}$, and program the random oracle representing H so that $H(\langle\mathcal{D}\rangle \parallel \langle\mathcal{R}\rangle \parallel m) := h$. This simulation fails only if $H(\langle\mathcal{D}\rangle \parallel \langle\mathcal{R}\rangle \parallel m)$ was already defined, which happens only with negligible probability since $\mathcal{R}$ is chosen after m is specified.

With presignatures, the above proof falls apart, precisely because $\mathcal{R}$ is chosen and given to the adversary before the adversary specifies m. Indeed, as is well known [DEF+18], there are attacks. Suppose the adversary is given presignatures $\mathcal{R}_1, \ldots, \mathcal{R}_K$. The adversary sets

$$\mathcal{R}^* := \sum_{k \in [K]} \mathcal{R}_k,$$

and attempts to find messages $m^*, m_1, \ldots, m_K$ such that

$$H(\langle\mathcal{D}\rangle \parallel \langle\mathcal{R}^*\rangle \parallel m^*) = \sum_{k \in [K]} H(\langle\mathcal{D}\rangle \parallel \langle\mathcal{R}_k\rangle \parallel m_k).$$

This is an instance of the $(K+1)$-sum problem, a generalization of the Birthday Problem studied by Wagner [Wag02]. Indeed, the adversary can generate $(K+1)$ lists of random numbers, where the first list is obtained by computing $H(\langle\mathcal{D}\rangle \parallel \langle\mathcal{R}^*\rangle \parallel m^*)$ for various messages m^*, the second by computing $H(\langle\mathcal{D}\rangle \parallel \langle\mathcal{R}_1\rangle \parallel m_1)$ for various messages m_1, and so on. This can generally be done much faster than the time $O(\sqrt{q})$ needed to break the discrete logarithm problem in E. Once this is done, the adversary can obtain signatures $(\mathcal{R}_k, z_k)$ on m_k for $k \in [K]$. From this, the adversary can compute $z^* \leftarrow \sum_{k \in [K]} z_k$ so that $(\mathcal{R}^*, z^*)$ is a valid signature on m^*.

[3] One can then reduce the security of the interactive identification scheme to the hardness of the discrete logarithm using the "forking lemma".

2.4 Re-randomized presignatures

One mitigation to this security weakness is to use **re-randomized presignatures**, just as in [GS21]. The idea is that a random tweak $\delta_k \in \mathbb{Z}_k$ is chosen by the challenger after the signing request is made, so that the original presignature $\mathcal{R}_k$ is replaced by the effective presignature $\mathcal{R}'_k := \mathcal{R}_k + \delta_k \mathcal{G}$. The value δ_k is given to the adversary to model that once chosen by the system it is publicly known. The same mitigation can be applied to biased presignatures: the effective presignature is then $\mathcal{R}'_k := u_k \mathcal{R}_k + (u'_k + \delta_k)\mathcal{G}$.

To implement this technique in a threshold setting, some type of "Random Beacon" must be used. A Random Beacon is a mechanism for obtaining public random values that remain hidden and unpredictable until a time determined by the protocol. For example, a Random Beacon can be efficiently implemented using a threshold BLS signature scheme [BLS01, Bol03]. Alternatively, it may be implemented by simply opening a previously secret-shared random value. Since the re-randomization is linear (and public), in terms of working with linear secret sharing, the impact is negligible. Depending on the details of the system, obtaining the value δ_k from the Random Beacon may result in some additional latency — but not necessarily so. For example, on a distributed system such as the Internet Computer [DFI22], signing requests must go through a consensus mechanism, which itself may be implemented so that it uses a threshold BLS signature to achieve finalization; that very same threshold BLS signature can be used to derive δ_k.

Let us reconsider the proof of security with this mitigation. We will consider the re-randomized biased presignature setting (which includes the re-randomized presignature setting as a special case where $u_k = 1$ and $u'_k = 0$). We will also combine this with additive key derivation. Again, the main part of the proof is to simulate signing queries by programming the random oracle representing H. The simulator generates the presignature $\mathcal{R}_k$ as

$$\mathcal{R}_k \leftarrow \zeta_k \mathcal{G} - \eta_k \mathcal{D},$$

where $\zeta_k, \eta_k \in \mathbb{Z}_q$ are chosen at random. At a later time, the adversary makes a corresponding signing query, where he specifies a message m_k an additive key tweak $e_k \in \mathbb{Z}_q$, and an presignature tweak $(u_k, u'_k) \in \mathbb{Z}_q^* \times \mathbb{Z}_q$. So the effective public key is $\mathcal{D}'_k := \mathcal{D} + e_k \mathcal{G}$, the effective presignature (used in the actual signature) is $\mathcal{R}'_k := u_k \mathcal{R} + (u'_k + \delta_k)\mathcal{G}$ and the resulting signature is $(\mathcal{R}'_k, z_k)$, where

$$z_k \mathcal{G} = \mathcal{R}'_k + h_k \mathcal{D}'_k = (u_k \mathcal{R}_k + (u'_k + \delta_k)\mathcal{G}) + h_k(\mathcal{D}_k + e_k \mathcal{G}),$$

which is equivalent to

$$\underbrace{u_k^{-1}(z_k - u'_k - \delta_k - e_k h_k)}_{=\zeta_k} \mathcal{G} = \mathcal{R}_k + \underbrace{u_k^{-1} h_k}_{=\eta_k} \mathcal{D}.$$

So the simulator can simply compute

$$h_k \leftarrow u_k \eta_k$$

and

$$z_k \leftarrow u_k \zeta_k + u'_k + \delta_k + e_k h_k,$$

and then program the random oracle so that $H(\langle \mathcal{D}'_k \rangle \parallel \langle \mathcal{R}'_k \rangle \parallel m_k) \coloneqq h_k$. Because δ_k is chosen only after the adversary makes the signing request, the input is unlikely to have been used before and the programming of the oracle will fail only with negligible probability.

We give an alternative proof of security of re-randomized presignatures in the generic group model, below in Section 3.2.

2.4.1 Batch re-randomization

Another variation worth considering is an attack game in which the adversary may submit a batch of signing queries, and a single random tweak $\delta \in \mathbb{Z}_q$ is used to update all the corresponding presignatures in the batch. That is, the adversary submits several signing queries $m_{k_1}, m_{k_2}, \ldots$ in a batch, which are paired with presignatures $\mathcal{R}_{k_1}, \mathcal{R}_{k_2}, \ldots$, and the effective presignatures are then computed as $\mathcal{R}'_{k_1} \coloneqq \mathcal{R}_{k_1} + \delta\mathcal{G}$, $\mathcal{R}'_{k_2} \coloneqq \mathcal{R}_{k_2} + \delta\mathcal{G}$, and so on.

This attack mode corresponds to a setting where a threshold signing protocol has signing requests coming in so fast that it makes sense to process these signing requests in batches, so as to amortize the cost of generating δ and computing $\delta\mathcal{G}$ over the size of the batch.

One can easily verify that the above security proof extends to cover batch re-randomization.

2.5 Re-randomizing presignatures via hashing

The FROST [KG20] and FROST2 protocols [CKM21] use a hash function to derive the re-randomization tweak and uses a second random group element as a part of the presignature. We abstract away the details of that protocol in a way that is still useful in the context of a threshold Schnorr scheme built using robust MPC primitives, such as in [GS22]. To this end, a presignature consists of a pair of random group elements $(\mathcal{R}_k, \mathcal{S}_k)$. To sign a message m_k, the effective presignature (used in the actual signature) is

$$\mathcal{R}'_k \coloneqq \mathcal{R}_k + \delta_k \mathcal{S}_k,$$

where

$$\delta_k \coloneqq \Delta(\langle \mathcal{D} \rangle \parallel \langle \mathcal{R}_k \rangle \parallel \langle \mathcal{S}_k \rangle \parallel \langle k \rangle \parallel m_k).$$

Here, Δ is a hash function whose output space is $\mathbb{Z}_q$. Note that $\mathcal{R}_k$ and $\mathcal{S}_k$ could be biased presignatures.

The main advantage of this approach to re-randomizing presignatures is that in the threshold setting, we do not need a Random Beacon, as in Section 2.4.

The FROST2 protocol was analyzed in [CKM21] in the random oracle model, giving a reduction to one-more discrete log (OMDL). Below in Section 3.3, we gave an analysis of the above abstract variant in the generic group model (where we also model the hash functions as random oracles). We believe this is useful because (a) the reduction to OMDL is extremely loose and our analysis here gives what is probably a more realistic bound on the effectiveness of any generic attacks, and (b) working in the generic group model allows us to examine further variants more quickly and easily.

NOTE: If we instead derive $\mathcal{R}'_k \coloneqq \mathcal{R}_k + \delta_k \mathcal{G}$, where

$$\delta_k \coloneqq \Delta(\langle \mathcal{D} \rangle \parallel \langle \mathcal{R}_k \rangle \parallel \langle k \rangle \parallel m_k),$$

one can carry out essentially the same attack as in Section 2.3. Indeed, suppose the adversary is given presignatures $\mathcal{R}_1, \ldots, \mathcal{R}_K$. For $k \in [K]$, define

$$\delta_k(m) \coloneqq \Delta(\langle\mathcal{D}\rangle \parallel \langle\mathcal{R}_k\rangle \parallel \langle k\rangle \parallel m)$$

and

$$h_k(m) \coloneqq H(\langle\mathcal{D}\rangle \parallel \langle\mathcal{R}_k + \delta_k(m_k)\mathcal{G}\rangle \parallel m).$$

The adversary sets

$$\mathcal{R}^* \coloneqq \sum_{k\in[K]} \mathcal{R}_k,$$

and tries to find messages $m^*, m_1, \ldots, m_K$ such that

$$H(\langle\mathcal{D}\rangle \parallel \langle\mathcal{R}^*\rangle \parallel m^*) = \sum_{k\in[K]} h_k(m_k).$$

This can again be done by solving an instance of an instance of the $(K+1)$-sum problem. Once this is done, the adversary asks for signatures $(\mathcal{R}_k + \delta_k(m_k)\mathcal{G}, z_k)$ on m_k for $k \in [K]$, computes

$$z^* \leftarrow \sum_{k\in[K]} (z_k - \delta_k(m_k)),$$

and outputs the forgery $(\mathcal{R}^*, z^*)$ on m^*.

3 Generic Group Model analysis

In the above analysis, we give a reduction to breaking the interactive Schnorr identification scheme. That security property can be reduced to the DL problem via a "forking lemma" argument. This gives a very "loose" reduction. An alternative approach is to carry out an analysis in the Generic Group Model (GGM). For the basic Schnorr attack game, this has already been done in [NSW09]. However, we want to extend this to various extended attack games.

3.1 Analysis of basic attack

Our approach and proof technique will be as in [GS21].

3.1.1 The EC-GGM

We review the EC-GGM (Elliptic Curve Generic Group Model), introduced in [GS21]. We assume an elliptic curve E is defined by an equation $y^2 = F(x)$ over $\mathbb{Z}_p$ and that the curve contains q points including the point at infinity $\mathcal{O}$. Here, p and q are odd primes. Let E^* be the set of non-zero points (excluding the point at infinity) on the curve, i.e., $(x, y) \in \mathbb{Z}_p \times \mathbb{Z}_p$ that satisfy $y^2 = F(x)$. From now on, we shall not be making any use of the usual group law for E, but simply treat E as a set; however, for a point $\mathcal{P} = (x, y) \in E^*$, we write $-\mathcal{P}$ to denote the point $(x, -y) \in E^*$.

An **encoding function** for E is a function

$$\pi : \mathbb{Z}_q \mapsto E$$

that is

- injective,
- **identity preserving**, meaning that $\pi(0) = \mathcal{O}$, and
- **inverse preserving**, meaning that for all $i \in \mathbb{Z}_q$, $\pi(-i) = -\pi(i)$.

In the EC-GGM, parties know E and interact with a **group oracle** $\mathcal{O}_{\text{grp}}$ that works as follows:

- $\mathcal{O}_{\text{grp}}$ on initialization chooses an encoding function π at random from the set of all encoding functions
- $\mathcal{O}_{\text{grp}}$ responds to two types of queries:
  - $(\texttt{map}, i)$, where $i \in \mathbb{Z}_q$:
    - return $\pi(i)$ *// models computing* $i\mathcal{G}$
  - $(\texttt{add}, \mathcal{P}_1, \mathcal{P}_2, c_1, c_2)$, where $\mathcal{P}_1, \mathcal{P}_2 \in E$ and $c_1, c_2 \in \mathbb{Z}_q$:
    - return $\pi\big(\ c_1\pi^{-1}(\mathcal{P}_1) + c_2\pi^{-1}(\mathcal{P}_2)\ \big)$ *// models computing* $c_1\mathcal{P}_1 + c_2\mathcal{P}_2$

NOTES:

1. The intuition is that the random choice of encoding function hides relations between group elements.
2. However, to make things more realistic, the encodings themselves have the same format as in a concrete elliptic curve, even though we do not at all use the group law of an elliptic curve.
3. Also to make things more realistic, the trivial relationship between a point and its inverse (that they share the same x-coordinate) is preserved.
4. Our model only captures the situation of elliptic curves over $\mathbb{Z}_p$ of prime order and cofactor 1. This is sufficient for many settings, and it covers all of the "`secp`" curves in [Cer10].
5. We have enhanced slightly the EC-GCM model from [GS21]: in that paper, the `add` query only supports coefficients $c_1 = c_2 = 1$. This "enhanced `add` query" only strengthens the model and brings it more in line with other formulations of the GGM (such as [Zha22]).

3.1.2 Modeling the attack on Schnorr in the EC-GCM

In the EC-GGM model, the generator $\mathcal{G}$ is encoded as $\pi(1)$ and the public key $\mathcal{D}$ is encoded as $\pi(d)$ for randomly chosen $d \in \mathbb{Z}_q$. These encodings of $\mathcal{G}$ and $\mathcal{D}$ are given to the adversary at the start of the signing attack game.

The adversary then makes a sequence of queries to both the group and signing oracles. The signing oracle on a message m itself works as usual, generating $r \in \mathbb{Z}_q$ at random, but it uses the group oracle to compute the encoding of $\mathcal{R} = r\mathcal{G}$. After that, the signing oracle computes $h \leftarrow H(\langle\mathcal{D}\rangle \parallel \langle\mathcal{R}\rangle \parallel m) \in \mathbb{Z}_q$ and $z \leftarrow r + hd$, and then gives the signature $(\mathcal{R}, z)$ to the adversary.

At the end of the signing attack game, the adversary outputs a forgery $(\mathcal{R}^*, z^*)$ on a message m^*. The signature is then verified using the verification algorithm, computing $h^* \leftarrow H(\langle\mathcal{D}\rangle \parallel \langle\mathcal{R}^*\rangle \parallel m^*) \in \mathbb{Z}_q$ and checking that $z^*\mathcal{G} = \mathcal{R}^* + h^*\mathcal{D}$ using the group oracle. WLOG, we may assume that the adversary has already performed this check and made

```
1. Initialization:
    (a) $\pi \leftarrow \{(0, \mathcal{O})\}$.
    (b) $d \xleftarrow{\$} \mathbb{Z}_q$
    (c) invoke $(\texttt{map}, 1)$ to obtain $\mathcal{G}$
    (d) invoke $(\texttt{map}, d)$ to obtain $\mathcal{D}$
    (e) return $(\mathcal{G}, \mathcal{D})$
2. To process a group oracle query $(\texttt{map}, i)$:
    (a) if $i \notin Domain(\pi)$:
        i. $\mathcal{P} \xleftarrow{\$} E$;
           while $\mathcal{P} \in Range(\pi)$ do: $\mathcal{P} \xleftarrow{\$} E$
        ii. add $(-i, -\mathcal{P})$ and $(i, \mathcal{P})$ to $\pi$
    (b) return $\pi(i)$
3. To process a group oracle query
   $(\texttt{add}, \mathcal{P}_1, \mathcal{P}_2, c_1, c_2)$:
    (a) for $j = 1, 2$: if $\mathcal{P}_j \notin Range(\pi)$:
        i. $i \xleftarrow{\$} \mathbb{Z}_q$;
           while $i \in Domain(\pi)$ do: $i \xleftarrow{\$} \mathbb{Z}_q$
        ii. add $(-i, -\mathcal{P}_j)$ and $(i, \mathcal{P}_j)$ to $\pi$
    (b) invoke $(\texttt{map}, c_1\pi^{-1}(\mathcal{P}_1) + c_2\pi^{-1}(\mathcal{P}_2))$
        and return the result
4. To process a request to sign $m$:
    (a) $r \xleftarrow{\$} \mathbb{Z}_q$
    (b) invoke $(\texttt{map}, r)$ to get $\mathcal{R}$
    (c) $h \leftarrow H(\langle\mathcal{D}\rangle \parallel \langle\mathcal{R}\rangle \parallel m) \in \mathbb{Z}_q$
    (d) $z \leftarrow r + hd$
    (e) return $(\mathcal{R}, z)$
```

Figure 2: Lazy-Sim

the corresponding calls to the group oracle. The adversary wins the signing attack game if $(\mathcal{R}^*, z^*)$ is a valid signature on m^* and m^* was not submitted as an input to the signing oracle.

We let N_{sig} be a bound on the number of signing queries made by the adversary, and N_{grp} be a bound on the number of group oracle queries made by the adversary. For simplicity, we assume that N_{grp} includes the group oracle queries made in the initialization step and in the verification step of the adversary's forgery attempt. We let N be a bound on the number of group oracle and signing queries made by during the attack. Later in the paper, we will consider scenarios where the adversary also makes queries to a random oracle, and in these scenarios, N will also bound the number of random oracle queries as well.

A lazy simulation of the signature attack game. Instead of choosing the encoding function π at random at the beginning of the attack game, we can lazily construct π a bit at a time. That is, we represent π as a set of pairs $(i, \mathcal{P})$ which grows over time — such a pair $(i, \mathcal{P})$ represents the relation $\pi(i) = \mathcal{P}$. Here, we give the entire logic for both the group and signing oracles in the forgery attack game. Figure 2 gives the details of **Lazy-Sim**. This is essentially the same as the lazy simulator in Fig. 2 in [GS21], except for the logic for processing signing requests, which has been changed to Schnorr signatures instead of ECDSA signatures (and the "enhanced `add` queries")

This lazy simulation is perfectly faithful. Specifically, the advantage of any adversary in the signature attack game using this lazy simulation of the group oracle is identical to that using the group oracle as originally defined.

A symbolic simulation of the signature attack game. We now define a **symbolic** simulation of the attack game. The essential difference in this game is that $Domain(\pi)$ will now consist of polynomials of the form $a + b\texttt{D}$, where $a, b \in \mathbb{Z}_q$ and $\texttt{D}$ is a variable (or indeterminant). Here, $\texttt{D}$ symbolically represents the value of d. Note that π will otherwise still satisfy all of the requirements of an encoding function. Figure 3 gives the details of **Symbolic-Sym**. This is essentially the same as the lazy simulator in Fig. 3 in [GS21],

```
1. Initialization:
   (a) $\pi \leftarrow \{(0, \mathcal{O})\}$.
   (b) invoke $(\texttt{map}, 1)$ to obtain $\mathcal{G}$
   (c) invoke $(\texttt{map}, \texttt{D})$ to obtain $\mathcal{D}$
   (d) return $(\mathcal{G}, \mathcal{D})$
2. To process a group oracle query $(\texttt{map}, i)$:
   (a) if $i \notin \mathit{Domain}(\pi)$:
       i. $\mathcal{P} \xleftarrow{\$} E$;
          if $\mathcal{P} \in \mathit{Range}(\pi)$ then abort
       ii. add $(-i, -\mathcal{P})$ and $(i, \mathcal{P})$ to $\pi$
   (b) return $\pi(i)$
3. To process a group oracle query
   $(\texttt{add}, \mathcal{P}_1, \mathcal{P}_2, c_1, c_2)$:
   (a) for $j = 1, 2$: if $\mathcal{P}_j \notin \mathit{Range}(\pi)$:
       i. $i \xleftarrow{\$} \mathbb{Z}_q$;
          if $i \in \mathit{Domain}(\pi)$ then abort
       ii. add $(-i, -\mathcal{P}_j)$ and $(i, \mathcal{P}_j)$ to $\pi$
   (b) invoke $(\texttt{map}, c_1\pi^{-1}(\mathcal{P}_1) + c_2\pi^{-1}(\mathcal{P}_2))$
       and return the result
4. To process a request to sign $m$:
   (a) $\mathcal{R} \xleftarrow{\$} E$, $z \xleftarrow{\$} \mathbb{Z}_q$
   (b) $h \leftarrow H(\langle\mathcal{D}\rangle \parallel \langle\mathcal{R}\rangle \parallel m) \in \mathbb{Z}_q$
   (c) $r \leftarrow z - h\texttt{D}$
   (d) if $r \in \mathit{Domain}(\pi)$ or $\mathcal{R} \in \mathit{Range}(\pi)$
       then abort
   (e) add $(-r, -\mathcal{R})$ and $(r, \mathcal{R})$ to $\pi$
   (f) return $(\mathcal{R}, z)$
```

Figure 3: Symbolic-Sim

except for the logic for processing signing requests (and the "enhanced `add` queries").

Essentially, the signing oracle in the symbolic simulation (i) chooses $\mathcal{R} \in E$ and $z \in \mathbb{Z}_q$ at random, (ii) sets $r \leftarrow z - h\texttt{D}$, where $h = H(\langle\mathcal{D}\rangle \parallel \langle\mathcal{R}\rangle \parallel m)$, (iii) "programs" π so that $\pi(r) = \mathcal{R}$, and (iv) returns the signature $(\mathcal{R}, z)$.

The following lemma is fairly straightforward, and may be proved along the same lines as Lemma 1 in [GS21].

Lemma 1. *The difference between the adversary's forging advantage in the Lazy-Sim and Symbolic-Sim games in Figures 2 and 3 is $O(N^2/q)$.*

Proof. See Appendix A below. □

Indeed, throughout this paper, we hew closely to the general strategies developed in [GS21], one of which is to carry out the generic group analysis in a modular fashion, moving first from the real attack to a symbolic simulation of the attack, and then to finish off the analysis in this symbolic simulation. The move from real attack to symbolic simulation is usually straightforward and fairly mechanical, and allows us to then focus fashion on the "meat" of the proof in a more intuitive fashion.

3.1.3 Proving security in the EC-GGM

By virtue of Lemma 1, it suffices to prove the security of Schnorr in the Symbolic-Sim game. We do this by reducing the security to specific preimage resistance security properties of H.

Assume the adversary's forgery is the signature $(\mathcal{R}^*, z^*)$ on the message m^*. Suppose $\pi^{-1}(\mathcal{R}^*) = a + b\texttt{D}$. By the verification equation, we must also have $\pi^{-1}(\mathcal{R}^*) = z^* - h^*\texttt{D}$. Let $h^* := H(\langle\mathcal{D}\rangle \parallel \langle\mathcal{R}^*\rangle \parallel m^*)$. Then we must have $a = z^*$ and $b = -h^*$.

Type I forgery: $\mathcal{R}^* = \pm\mathcal{R}$ for some $\mathcal{R}$ output by the signing oracle.

Let $\mathcal{R}^* = \epsilon\mathcal{R}$, with $\epsilon \in \{\pm 1\}$. Suppose m was the input to signing oracle that produced the signature $(\mathcal{R}, z)$, and let $h := H(\langle\mathcal{D}\rangle \parallel \langle\mathcal{R}\rangle \parallel m)$. Then we must have

$$z^* - h^*\mathtt{D} = \epsilon(z - h\mathtt{D}).$$

In particular, $h^* = \epsilon h$.

In this case, the adversary must essentially win a UOWHF-like attack on H, which we call **Preimage Attack I** — details below.

Type II forgery: not type I and $h^* \neq 0$.

Since $b = -h^* \neq 0$, the group element $\mathcal{R}^*$ was generated at random as the result of a group oracle query made by the adversary. (Note that the assumption $h^* \neq 0$ is used here to rule out the possibility of $\mathcal{R}^*$ being cooked up directly by the adversary, which is allowed in the EC-GGM model.)

So in this case, the adversary must essentially win a certain type of preimage attack game on H, which we call **Preimage Attack II** — details below.

Type III forgery: not type I and $h^* = 0$.

In this case, the adversary must find a preimage of zero under H, which we call **Preimage Attack III**. (Note this case does not arise in the analysis of [NSW09] because they do not allow access to π^{-1} as is done in the EC-GCM.)

3.1.4 Preimage attack games

In the above analysis, we sketched a reduction to various preimage attacks on H. Here, we state these preimage attacks in more detail. (Preimage Attacks I and II are stated in greater generality than what we need here to cover other situations we will encounter later.)

Preimage Attack I on H.

- For $k = 1, 2, \ldots$, the adversary makes a *challenge query*, giving $(m_k, \mathcal{D}'_k)$ to challenger, who responds with random $\mathcal{R}_k$.

  Let $h^*_k = H(\langle\mathcal{D}'_k\rangle \parallel \langle\mathcal{R}_k\rangle \parallel m_k)$.

- To win, the adversary outputs k, $(m^*, \mathcal{D}^*) \neq (m_k, \mathcal{D}'_k)$, and $\epsilon \in \{\pm 1\}$ such that

$$H(\langle\mathcal{D}^*\rangle \parallel \langle\epsilon\mathcal{R}_k\rangle \parallel m^*) = \epsilon h^*_k.$$

Preimage Attack II on H.

- For $i = 1, 2, \ldots$, the adversary makes a *challenge query*, giving h^*_i to challenger, who responds with random $\mathcal{R}^*_i$.

- To win, the adversary outputs i, $(m^*, \mathcal{D}^*)$, and $\epsilon \in \{\pm 1\}$ such that

$$H(\langle\mathcal{D}^*\rangle \parallel \langle\epsilon\mathcal{R}^*_i\rangle \parallel m^*) = \epsilon h^*_i.$$

Preimage Attack III on H. To win, the adversary outputs a bit string x such that $H(x) = 0$.

3.1.5 Concrete security bounds

We can derive concrete security bounds for the analysis in Section 3.1.3. If an adversary $\mathcal{A}$ has an advantage $\aleph$ in forging a signature, then

$$\aleph = O(N^2/q + \aleph_{\mathrm{I}} + \aleph_{\mathrm{II}} + \aleph_{\mathrm{III}}). \tag{1}$$

Here, $\aleph_X$ is the advantage of an adversary $\mathcal{A}_X$ in winning Preimage Attack X, for $X \in \{\mathrm{I}, \mathrm{II}, \mathrm{III}\}$. Each $\mathcal{A}_X$ has roughly the same running time as $\mathcal{A}$. Moreover, $\mathcal{A}_{\mathrm{I}}$ makes at most N_{sig} challenge queries and $\mathcal{A}_{\mathrm{II}}$ makes at most N_{grp} challenge queries. This follows from Lemma 1, together with the analysis in Section 3.1.3.

Suppose we model H as a random oracle (as well as working in the EC-GGM). In this paper, we generally assume that the output space of H is $\mathbb{Z}_q$. However, it is useful to consider a smaller output space as well, as this can be used to generate shorter signatures (using the standard technique where a signature consists of (h, z), rather than $(\mathcal{R}, z)$).

So suppose H has an output space of size M. Also, assume that N also bounds the number of queries made by $\mathcal{A}$ to the random oracle representing H. Note that this also bounds the number of random oracle queries made by each $\mathcal{A}_{\mathrm{X}}$. Then (1) implies

$$\aleph = O(N^2/q + N/M). \tag{2}$$

Indeed, consider an adversary that carries out Preimage Attack I or II and makes at most N_{h} random oracle queries and N_{ch} challenge queries. Then such an adversary wins this attack with probability at most $O(N_{\mathrm{ch}}^2/q + N_{\mathrm{h}}/M)$. The term $O(N_{\mathrm{ch}}^2/q)$ bounds the probability that there are collisions among the any of the N_{ch} random group elements generated by the challenger. Similarly, an adversary that carries out Preimage Attack III and makes at most N_{h} random oracle queries wins this attack with probability $O(N_{\mathrm{h}}/M)$. The bound (2) immediately follows.

The above analysis is similar to that in [NSW09], except that they consider preimage attacks with only a single challenge, and then make a "guessing" argument to complete the reduction to the hardness of winning such a single-challenge preimage attack. This leads to somewhat artificially pessimistic security bounds. Note that a similar security bound was proved already in [BL19].

3.2 Analysis of attack with re-randomized presignatures

We assume unbiased presignatures and no key derivation, but with presignatures re-randomized (as in Section 2.4). Figure 4 gives the detailed logic of **Lazy-Sim**, which is a lazy simulation of the forgery attack game. Figure 5 gives the detailed logic of **Symbolic-Sim**, which is a symbolic simulation of the forgery attack game. Our approach for designing the symbolic simulation in this setting is similar to that [GS21], in which each presignature $\mathcal{R}_k$ corresponds to a variable $\mathtt{R}_k$, meaning that $\pi(\mathtt{R}_k) = \mathcal{R}_k$. When such a presignature is

```
1. Initialization:
   (a) $\pi \leftarrow \{(0, \mathcal{O})\}$.
   (b) $d \xleftarrow{\$} \mathbb{Z}_q$
   (c) invoke $(\texttt{map}, 1)$ to obtain $\mathcal{G}$
   (d) invoke $(\texttt{map}, d)$ to obtain $\mathcal{D}$
   (e) $k \leftarrow 0$; $K \leftarrow \emptyset$
   (f) return $(\mathcal{G}, \mathcal{D})$
2. To process a group oracle query $(\texttt{map}, i)$:
   (a) if $i \notin Domain(\pi)$:
         i. $\mathcal{P} \xleftarrow{\$} E$;
            while $\mathcal{P} \in Range(\pi)$ do: $\mathcal{P} \xleftarrow{\$} E$
        ii. add $(-i, -\mathcal{P})$ and $(i, \mathcal{P})$ to $\pi$
   (b) return $\pi(i)$
3. To process a group oracle query
   $(\texttt{add}, \mathcal{P}_1, \mathcal{P}_2, c_1, c_2)$:
   (a) for $j = 1, 2$: if $\mathcal{P}_j \notin Range(\pi)$:
         i. $i \xleftarrow{\$} \mathbb{Z}_q$;
            while $i \in Domain(\pi)$ do: $i \xleftarrow{\$} \mathbb{Z}_q$
        ii. add $(-i, -\mathcal{P}_j)$ and $(i, \mathcal{P}_j)$ to $\pi$
   (b) invoke $(\texttt{map}, c_1\pi^{-1}(\mathcal{P}_1) + c_2\pi^{-1}(\mathcal{P}_2))$
       and return the result
4. To process a presignature request:
   (a) $k \leftarrow k + 1$; $K \leftarrow K \cup \{k\}$
   (b) $r_k \xleftarrow{\$} \mathbb{Z}_q$
   (c) invoke $(\texttt{map}, r_k)$ to get $\mathcal{R}_k$
   (d) return $\mathcal{R}_k$
5. To process a request to sign $m_k$ using pre-
   signature number $k \in K$:
   (a) $\delta_k \xleftarrow{\$} \mathbb{Z}_q$; $r'_k \leftarrow r_k + \delta_k$
   (b) invoke $(\texttt{map}, r'_k)$ to get $\mathcal{R}'_k$
   (c) $h_k \leftarrow H(\langle\mathcal{D}\rangle \parallel \langle\mathcal{R}'_k\rangle \parallel m_k) \in \mathbb{Z}_q$
   (d) $z_k \leftarrow r_k + \delta_k + h_k d$
   (e) $K \leftarrow K \setminus \{k\}$; return $(\mathcal{R}'_k, z_k, \delta_k)$
```

Figure 4: Lazy-Sim

used to process a signing request, we generate $\delta_k, z_k \in \mathbb{Z}_q$ and $\mathcal{R}'_k \in E$ at random, compute the hash $h_k \in \mathbb{Z}_q$, program π so that $\pi(\mathtt{R}_k + \delta_k) = \mathcal{R}'_k$, and then substitute

$$\mathtt{R}_k \mapsto z_k - \delta_k - h_k\mathtt{D}$$

throughout $Domain(\pi)$. This same "substitution strategy" for dealing with presignatures in the GGM was used extensively in [GS21], and works equally well here. The following lemma is fairly straightforward, and may be proved along the same lines as Lemma 2 in [GS21].

Lemma 2. *The difference between the adversary's forging advantage in the Lazy-Sim and Symbolic-Sim games in Figures 4 and 5 is $O(N^2/q)$.*

Proof. See Appendix B below. □

By virtue of this lemma, it suffuces to analyze the forgery attack in the symbolic simulation.

Type I forgery: $\mathcal{R}^* = \pm\mathcal{R}$ for some $\mathcal{R}$ output by signing oracle.

This is handled exactly the same as Type I in the basic attack in Section 3.1.3.

Type II forgery: not type I and $h^* \neq 0$.

By some simple case analysis, we can assume that $\mathcal{R}^*$ was randomly generated in processing a group oracle query made by the adversary. Suppose that initially

$$\pi^{-1}(\mathcal{R}^*) = a + b\mathtt{D} + \sum_k c_k\mathtt{R}_k,$$

```
1. Initialization:
   (a) $\pi \leftarrow \{(0, \mathcal{O})\}$.
   (b) invoke $(\texttt{map}, 1)$ to obtain $\mathcal{G}$
   (c) invoke $(\texttt{map}, \texttt{D})$ to obtain $\mathcal{D}$
   (d) $k \leftarrow 0$; $K \leftarrow \emptyset$
   (e) return $(\mathcal{G}, \mathcal{D})$
2. To process a group oracle query $(\texttt{map}, i)$:
   (a) if $i \notin \mathit{Domain}(\pi)$:
         i. $\mathcal{P} \xleftarrow{\$} E$; if $\mathcal{P} \in \mathit{Range}(\pi)$ then abort
         ii. add $(-i, -\mathcal{P})$ and $(i, \mathcal{P})$ to $\pi$
   (b) return $\pi(i)$
3. To process a group oracle query
   $(\texttt{add}, \mathcal{P}_1, \mathcal{P}_2, c_1, c_2)$:
   (a) for $j = 1, 2$: if $\mathcal{P}_j \notin \mathit{Range}(\pi)$:
         i. $i \xleftarrow{\$} \mathbb{Z}_q$; if $i \in \mathit{Domain}(\pi)$ then abort
         ii. add $(-i, -\mathcal{P}_j)$ and $(i, \mathcal{P}_j)$ to $\pi$
   (b) invoke $(\texttt{map}, c_1\pi^{-1}(\mathcal{P}_1) + c_2\pi^{-1}(\mathcal{P}_2))$
       and return the result
4. To process a presignature request:
   (a) $k \leftarrow k + 1$; $K \leftarrow K \cup \{k\}$
   (b) invoke $(\texttt{map}, \texttt{R}_k)$ to get $\mathcal{R}_k$
   (c) return $\mathcal{R}_k$
5. To process a request to sign $m_k$ using presignature number $k \in K$:
   (a) $\delta_k \xleftarrow{\$} \mathbb{Z}_q$; $r'_k \leftarrow \texttt{R}_k + \delta_k$
   (b) $\mathcal{R}'_k \xleftarrow{\$} E$
   (c) if $r'_k \in \mathit{Domain}(\pi)$ or $\mathcal{R}'_k \in \mathit{Range}(\pi)$
       then abort
   (d) add $(r'_k, \mathcal{R}'_k)$ and $(-r'_k, -\mathcal{R}'_k)$ to $\pi$
   (e) $h_k \leftarrow H(\langle\mathcal{D}\rangle \parallel \langle\mathcal{R}'_k\rangle \parallel m_k) \in \mathbb{Z}_q$
   (f) $z_k \xleftarrow{\$} \mathbb{Z}_q$;
       substitute $\texttt{R}_k \mapsto z_k - \delta_k - h_k\texttt{D}$
       throughout $\mathit{Domain}(\pi)$ and abort if
       $\mathit{Domain}(\pi)$ “collapses” (i.e., two distinct elements of $\mathit{Domain}(\pi)$ become
       equal after the substitution)
   (g) $K \leftarrow K \setminus \{k\}$; return $(\mathcal{R}'_k, z_k, \delta_k)$
```

Figure 5: Symbolic-Sim

where the c_k's are all nonzero. If the sum on k is empty, this can be handled the same as Type II in the basic attack in Section 3.1.3. Otherwise, in order for the forgery to be valid, the $\texttt{R}_k$ variables need to be eliminated by substitution, so as to end up with

$$\pi^{-1}(\mathcal{R}^*) = z^* - h^*\texttt{D}.$$

Suppose that all but one has been eliminated, say $\texttt{R}_\ell$, so that at that time,

$$\pi^{-1}(\mathcal{R}^*) = a' + b'\texttt{D} + c_\ell\texttt{R}_\ell.$$

The last substitution is $\texttt{R}_\ell \mapsto z_\ell - \delta_\ell - h_\ell\texttt{D}$, yielding

$$\pi^{-1}(\mathcal{R}^*) = \underbrace{\{a' + c_\ell(z_\ell - \delta_\ell)\}}_{=z^*} + \underbrace{\{b' - c_\ell h_\ell\}}_{=-h^*}\texttt{D}.$$

So in this case, the adversary can win a certain type of preimage attack game on H, which we call **Preimage Attack II′** — details below.

Type III forgery: not type I and $h^* = 0$.

This is handled exactly the same as Type III in the basic attack in Section 3.1.3.

3.2.1 Another preimage attack

We describe in more detail the preimage attack used in the above security analysis. This attack is stated in greater generality than what we need here to cover other situations we will encounter later.

Preimage Attack II$'$ on H.

- The challenger gives a collection $\{\mathcal{R}_i^*\}_{i=1}^{N_{\text{ch}}}$ of random challenges to the adversary (each $\mathcal{R}_i^*$ is a random element of E).
- For $k = 1, 2, \ldots$, the adversary submits a *completion query* to the challenger consisting of an index set $\mathcal{I}_k \subseteq \{1, \ldots, N_{\text{ch}}\}$ that is disjoint from $\mathcal{I}_1 \cup \cdots \cup \mathcal{I}_{k-1}$, along with $\mathcal{D}'_k$, m_k, and $\{(b_i, c_i)\}_{i \in \mathcal{I}_k}$, where each $(b_i, c_i) \in \mathbb{Z}_q \times \mathbb{Z}_q^*$.
  - The challenger generates $\mathcal{R}'_k$ at random and returns this to the adversary.
  - Let $h_k = H(\langle \mathcal{D}'_k \rangle \parallel \langle \mathcal{R}'_k \rangle \parallel m_k)$ and $h_i^* = b_i - c_i \cdot h_k$ for $i \in \mathcal{I}_k$.
- To win, the adversary outputs $i \in \mathcal{I}$, $(m^*, \mathcal{D}^*)$, and $\epsilon \in \{\pm 1\}$ such that

$$H(\langle \mathcal{D}^* \rangle \parallel \langle \epsilon \mathcal{R}_i^* \rangle \parallel m^*) = \epsilon h_i^*.$$

In the above attack game, the various $\mathcal{R}_i^*$ values correspond to outputs from the group oracle in the symbolic simulation of the signing attack, while the various $\mathcal{R}'_k$ values correspond to the outputs of the signing oracle. The kth completion query in the above attack game corresponds to the kth signing query in the symbolic simulation of the signing attack, and the set of indices $\mathcal{I}_k$ represents those group elements that were output by the group oracle whose last remaining presignature variable is being eliminated by substitution from this signing request.

3.2.2 Concrete security bounds

If an adversary $\mathcal{A}$ has an advantage $\aleph$ in forging a signature, then

$$\aleph = O(N^2/q + \aleph_{\text{I}} + \aleph_{\text{II}} + \aleph_{\text{II}'} + \aleph_{\text{III}}). \tag{3}$$

Here, $\aleph_X$ is the advantage of an adversary $\mathcal{A}_X$ in winning Preimage Attack X, for $X \in \{\text{I}, \text{II}, \text{II}', \text{III}\}$. Each $\mathcal{A}_X$ has roughly the same running time as $\mathcal{A}$, plus time $O(LN)$, where L is the maximum number of unused presignatures that are extant at any time. Moreover, $\mathcal{A}_{\text{I}}$ makes at most N_{sig} challenge queries, $\mathcal{A}_{\text{II}}$ makes at most N_{grp} challenge queries, and $\mathcal{A}_{\text{II}'}$ receives at most N_{grp} challenges and makes at most N_{sig} completion queries.

Now suppose we model H as a random oracle with an output space of size M. Also, assume that N also bounds the number of queries made by $\mathcal{A}$ to the random oracle representing H. Note that this also bounds the number of random oracle queries made by each $\mathcal{A}_{\text{X}}$. Then (3) implies

$$\aleph = O(N^2/q + N/M). \tag{4}$$

To see this, suppose that in Preimage Attack II$'$, the adversary receives N_{ch} random challenges, and makes at most N_{cmp} completion queries and at most N_{h} random oracle queries. Assume no collisions among the random challenges occur. This means that for a given random oracle query of the form

$$H(\langle \mathcal{D}^* \rangle \parallel \langle \epsilon \mathcal{R}_i^* \rangle \parallel m^*), \tag{5}$$

there is a unique index i and values b_i, c_i, the h_k such that

$$H(\langle \mathcal{D}^* \rangle \parallel \langle \epsilon \mathcal{R}_i^* \rangle \parallel m^*) = b_i - c_i \cdot h_k$$

must hold in order for the random oracle query (5) to lead to a win. Here, k is the index of the completion query which included i in $\mathcal{I}_k$. Moreover, assuming that $\mathcal{D}'_k \neq \pm \mathcal{R}_i^*$ and the adversary did not happen to query $H(\langle \mathcal{D}'_k \rangle \parallel \langle \mathcal{R}'_k \rangle \parallel m_k)$ before the kth completion query was made, the value $h_k \coloneqq H(\langle \mathcal{D}'_k \rangle \parallel \langle \mathcal{R}'_k \rangle \parallel m_k)$ is random and independent of $H(\langle \mathcal{D}^* \rangle \parallel \langle \epsilon \mathcal{R}_i^* \rangle \parallel m^*)$, b_i, and c_i, and so the random oracle query (5) leads to a win with probability at most $1/M$. From this, we see that the adversary wins the attack with probability at most

$$O((N_{\text{ch}} + N_{\text{cmp}} + N_{\text{h}})^2/q + N_{\text{h}}/M).$$

The bound (4) now follows.

3.2.3 Variations

If we use biased presignatures, then effectively $\mathcal{R}_k$ gets replaced by $u_k \mathcal{R}_k + u'_k \mathcal{G}$ just before signing a message, where $u_k \neq 0$ and u'_k are explicitly given by the adversary. So in the symbolic simulation, the signing oracle programs π so that $\pi(u_k \mathtt{R}_k + u'_k + \delta_k) = \mathcal{R}'_k$ and substitutes

$$\mathtt{R}_k \mapsto u_k^{-1}(z_k - u'_k - \delta_k - h_k \mathtt{D}).$$

The general argument does not really change at all. If we use additive key derivation, deriving $\mathcal{D}'_k \coloneqq \mathcal{D} + e_k \mathcal{G}$, then this substitution becomes

$$\mathtt{R}_k \mapsto u_k^{-1}(z_k - h_k e_k - u'_k - \delta_k - h_k \mathtt{D}).$$

The argument is also easily adapted to deal with batch re-randomization (see Section 2.4.1). The same concrete security bounds in Section 3.2.2 also hold here.

3.3 Re-randomizing presignatures via hashing

We now analyze the security of Schnorr signatures in the GGM with presignatures that are re-randomized via hashing, as discussed in Section 2.5. Here, we will model Δ as a random oracle with output space $\mathbb{Z}_q$. We will also model as H as random oracle.

We will assume unbiased presignatures for now and later examine biased presignatures as well as additive key derivation. When a signing query on a message m_k is made that uses the presignature $(\mathcal{R}_k, \mathcal{S}_k)$, a preliminary computation

$$\begin{aligned} \delta_k &\leftarrow \Delta(\langle \mathcal{D} \rangle \parallel \langle \mathcal{R}_k \rangle \parallel \langle \mathcal{S}_k \rangle \parallel \langle k \rangle \parallel m_k), \\ \mathcal{R}'_k &\leftarrow \mathcal{R}_k + \delta_k \mathcal{S}_k, \\ h_k &\leftarrow H(\langle \mathcal{D} \rangle \parallel \langle \mathcal{R}'_k \rangle \parallel m_k) \end{aligned}$$

is made. WLOG, we can assume that the adversary has already computed these values himself before making the signing query. Moreover, to simplify the analysis, we make one more assumption about the adversary. Namely, whenever the adversary makes a random oracle query of the form

$$\delta_k \leftarrow \Delta(\langle \mathcal{D} \rangle \parallel \langle \mathcal{R}_k \rangle \parallel \langle \mathcal{S}_k \rangle \parallel \langle k \rangle \parallel m_k),$$

we assume it immediately makes a "special `add` query"

$$(\texttt{add}, \mathcal{R}_k, \mathcal{S}_k, 1, \delta_k)$$

to the group oracle to obtain the encoding of the group element $\mathcal{R}'_k := \mathcal{R}_k + \delta_k \mathcal{S}_k$. We may also assume that it then immediately makes the random oracle query

$$h_k \leftarrow H(\langle \mathcal{D} \rangle \parallel \langle \mathcal{R}'_k \rangle \parallel m_k).$$

So to model this situation in the symbolic simulation, we introduce variables $\mathtt{R}_k$ and $\mathtt{S}_k$, where $\pi(\mathtt{R}_k) = \mathcal{R}_k$ and $\pi(\mathtt{S}_k) = \mathcal{S}_k$. When a signing query as above is made, on a message m_k that uses the presignature $(\mathcal{R}_k, \mathcal{S}_k)$, the signing oracle generates z_k at random. Before returning the signature $(\mathcal{R}'_k, z_k)$, the signing oracle also substitutes

$$\mathtt{R}_k \mapsto z_k - \delta_k \mathtt{S}_k - h_k \mathtt{D} \tag{6}$$

throughout *Domain*(π). The symbolic simulation will "fail" if any of these substitutions cause *Domain*(π) to "collapse" (i.e., if two distinct elements of *Domain*(π) before the substitution become equal afterwards).

We leave it to the reader to verify that the analog of Lemma 2 above holds as well for this symbolic simulator.

As usual, suppose the forgery is a signature $(\mathcal{R}^*, z^*)$ on a message m^*. Note that the signing oracle does not generate any new group elements, so we do not categorize forgeries as we did before. We may assume that $\mathcal{R}^*$ was randomly generated by a group oracle query — otherwise, the adversary must essentially win Preimage Attack III on H (as in Section 3.1.4, but where H is modeled as a random oracle).

Suppose that initially

$$\pi^{-1}(\mathcal{R}^*) = a + b\mathtt{D} + \sum_k (c_k \mathtt{R}_k + d_k \mathtt{S}_k), \tag{7}$$

where each (c_k, d_k) is nonzero (as a pair). Note that the constants a, b, and c_k, d_k for all indices k are fixed before $\mathcal{R}^*$ is randomly generated. In order for this forgery to be valid, the $\mathtt{R}_k$ variables need to be eliminated by substitution, so as to end up with

$$\pi^{-1}(\mathcal{R}^*) = z^* - h^* \mathtt{D}.$$

In fact, after substitution, we have

$$\pi^{-1}(\mathcal{R}^*) = \underbrace{\{a + \sum_k c_k z_k\}}_{=z^*} + \underbrace{\{b - \sum_k c_k h_k\}}_{=-h^*} \mathtt{D} + \sum_k \underbrace{\{d_k - c_k \delta_k\}}_{=0} \mathtt{S}_k. \tag{8}$$

For the forgery to be valid, we must have $d_k - c_k\delta_k = 0$ for each index k. If $c_k = 0$, then $d_k = 0$ as well; moreover, since we are assuming that $(c_k, d_k) \neq (0, 0)$, this implies $c_k \neq 0$. In particular,

$$\delta_k = \frac{d_k}{c_k}$$

for each index k. This means that at the time we generate $\mathcal{R}^*$, we can inspect the queries to the random oracle Δ to find for each index k an input

$$(\langle \mathcal{D} \rangle \parallel \langle \mathcal{R}_k \rangle \parallel \langle \mathcal{S}_k \rangle \parallel \langle k \rangle \parallel m_k)$$

to Δ that yields the output d_k/c_k. If the forgery is to be valid, then except with negligible probability, the adversary must have already made such a query and it will be unique. Thus, at the time we generate $\mathcal{R}^*$ at random, the inputs to H that determine the h_k's have already been determined. More precisely, by our assumptions on the adversary, either

(i) this is a "special `add` query" as discussed above, or

(ii) all of the h_k's have already been computed.

In the first case, the adversary must essentially win Preimage Attack I on H, while in the second case, he must essentially win Preimage Attack II on H (as in Section 3.1.4, but where H is modeled as a random oracle).

Concrete security bounds. To make the above analysis concrete, we have to calculate the probability that the above inspection process fails. For it to fail, it means that either (a) the adversary finds a collision in Δ, or (b) for some $\mathcal{R}^*$ output by the group oracle, for each k in (7) for which the adversary has not already made a relevant query to Δ whose output hits d_k/c_k, the adversary must make such a query at a later time whose output (by pure luck) hits d_k/c_k. The probability that (a) or (b) occurs is at most $O(N^2/q)$ — more precisely, (a) occurs with probability $O(N_\mathrm{h}^2)$ and (b) occurs with probability $O(N_\mathrm{grp} N_\mathrm{h})$. From this, it follows that if H is modeled as a random oracle with an output space of size M, the adversary's forging advantage is $O(N^2/q + N/M)$.

3.3.1 Variations

If we use biased presignatures, then effectively $\mathcal{R}_k$ gets replaced by $\tilde{\mathcal{R}}_k := u_k\mathcal{R}_k + u'_k\mathcal{G}$ and $\mathcal{S}_k$ gets replaced by $\tilde{\mathcal{S}}_k := v_k\mathcal{S}_k + v'_k\mathcal{G}$, where $u_k \neq 0$, u'_k, $v_k \neq 0$, and v'_k are explicitly given by the adversary. The input to Δ used to derive δ_k is then

$$(\langle \mathcal{D} \rangle \parallel \langle \tilde{\mathcal{R}}_k \rangle \parallel \langle \tilde{\mathcal{S}}_k \rangle \parallel \langle k \rangle \parallel m_k) \tag{9}$$

The substitution (6) then becomes

$$\mathtt{R}_k \mapsto u_k^{-1}(z_k - u'_k - \delta_k v'_k - \delta_k v_k \mathtt{S}_k - h_k \mathtt{D}) \tag{10}$$

and (8) becomes

$$\pi^{-1}(\mathcal{R}^*) = \underbrace{\{a + \sum_k c_k u_k^{-1}(z_k - u'_k - \delta_k v_k)\}}_{=z^*} + \underbrace{\{b - \sum_k c_k u_k^{-1} h_k\}}_{=-h^*} \mathtt{D} + \sum_k \underbrace{\{d_k - c_k u_k^{-1} v_k \delta_k\}}_{=0} \mathtt{S}_k. \tag{11}$$

The main argument does not change too much. One thing we may have to adjust is that now we are inspecting the queries to Δ, looking for inputs that output

$$\delta_k = \frac{u_k}{v_k}\frac{d_k}{c_k}.$$

However, we have to look for such inputs *before* the signing request is made that determines u_k and v_k. Nevertheless, since the biased presignatures $\tilde{\mathcal{R}}$ and $\tilde{\mathcal{S}}$ are input to Δ, we can actually use the GGM to determine u_k and v_k. That is, we are looking for inputs to Δ of the form (9) such that

- $\pi^{-1}(\tilde{\mathcal{R}}) = (u_k \mathtt{R}_k + \cdots)$ and $\pi^{-1}(\tilde{\mathcal{S}}) = (v_k \mathtt{S}_k + \cdots)$, and
- the output is
$$\frac{u_k}{v_k}\frac{d_k}{c_k}.$$

Indeed, we can assume WLOG that for all queries to Δ, the corresponding group element encodings $\tilde{\mathcal{R}}$ and $\tilde{\mathcal{S}}$ are already in *Range*(π). Moreover, $\pi^{-1}(\tilde{\mathcal{R}})$ and $\pi^{-1}(\tilde{\mathcal{S}})$ need to be of this form if they are to be of the required form $u_k \mathtt{R}_k + u'_k$ and $v_k \mathtt{S}_k + v'_k$ at the time the actual signing request is made (none of the substitutions performed between now and then will affect the coefficients of $\mathtt{R}_k$ or $\mathtt{S}_k$).

If we use additive key derivation, deriving $\mathcal{D}'_k \coloneqq \mathcal{D} + e_k \mathcal{G}$, then we also need to include $\mathcal{D}'_k$ as input to Δ, in place of $\mathcal{D}$. The substitution (10) then becomes

$$\mathtt{R}_k \mapsto u_k^{-1}(z_k - u'_k - h_k e_k - \delta_k v'_k - \delta_k v_k \mathtt{S}_k - h_k \mathtt{D}) \tag{12}$$

and the constant term in (11) is adjusted accordingly. Again, the main argument does not change too much.

Concrete security bounds. For each of these variations, the same security bounds as above: if H is modeled as a random oracle with an output space of size M, the adversary's forging advantage is $O(N^2/q + N/M)$.

4 Batch randomness extraction

In Section 2.2, we introduced biased presignatures. As discussed there, this models a common situation in the threshold setting where we utilize a simple DKG protocol in which each party securely distributes shares of an ephemeral secret key and publishes the corresponding ephemeral public key, after which a collection of these ephemeral keys is agreed upon and added together to obtain a presignature. All if this is done just to create a single presignature. However, it is possible to use the same approach to generate many presignatures for the price of one, leading to significantly more efficient protocols. Specifically, instead of just adding these ephemeral keys together, we can take several different linear combinations of them, producing several presignatures. This general technique of "batch randomness extraction" first appeared in [HN06] in a somewhat different setting. It was first proposed in the context of threshold Schnorr signatures in [BHK+23].

We do not need to go into the details of how this batching done, as it can all be abstracted away as a more general type of biased presignature, as discussed in [GS23]. This generalized biasing works as follows. Let $P \leq Q$ be fixed parameters. A batch of "initial" presignatures $\mathcal{R}_k^{(1)}, \ldots, \mathcal{R}_k^{(Q)}$ is generated at random and given to the adversary. Here, there may be many such batches and k is the index of the batch. The adversary then specifies a "bias" $(U_k, \mathbf{u}'_k)$, where $U_k \in \mathbb{Z}_q^{P \times Q}$ is a full rank matrix and $\mathbf{u}'_k \in \mathbb{Z}_q^{P \times 1}$ is an arbitrary column vector. This batch of initial presignatures is then converted to a batch of "derived" presignatures $\bar{\mathcal{R}}_k^{(1)}, \ldots, \bar{\mathcal{R}}_k^{(P)}$ as follows:

$$\begin{pmatrix} \bar{\mathcal{R}}_k^{(1)} \\ \vdots \\ \bar{\mathcal{R}}_k^{(P)} \end{pmatrix} = U_k \begin{pmatrix} \mathcal{R}_k^{(1)} \\ \vdots \\ \mathcal{R}_k^{(Q)} \end{pmatrix} + \mathbf{u}'_k \mathcal{G}. \tag{13}$$

A particular signing query specifies a pair of indices (k, i) so that a message $m_{k,i}$ is signed using the derived presignature $\bar{\mathcal{R}}_k^{(i)}$.

4.1 Re-randomized presignatures

Just as in Section 2.4, when signing a message $m_{k,i}$ we can re-randomize the derived presignature $\bar{\mathcal{R}}_k^{(i)}$, replacing it with

$$\hat{\mathcal{R}}_k^{(i)} := \bar{\mathcal{R}}_k^{(i)} + \delta_{k,i} \mathcal{G},$$

where $\delta_{k,i}$ is generated at random only after the signing request has been made (this corresponds to accessing a Random Beacon in the threshold setting).

As in Section 2.4, we give an efficient reduction from this scheme to the security of the interactive Schnorr identification scheme, modeling H as a random oracle. As usual, the key is to show how to simulate the signing queries. For the kth batch of initial presignatures, for each $j \in [Q]$, our simulator will generate $\zeta_k^{(j)}, \eta_k^{(j)} \in \mathbb{Z}_q$ at random, and then compute

$$\mathcal{R}_k^{(j)} \leftarrow \zeta_k^{(j)} \mathcal{G} - \eta_k^{(j)} \mathcal{D}.$$

The adversary then specifies the bias

$$U_k = \left(\sigma_{k,i}^{(j)}\right)_{i \in [P], j \in [Q]}, \quad \mathbf{u}'_k = (\mu_{k,i})_{i \in [P]},$$

and we have

$$\bar{\mathcal{R}}_k^{(i)} = \sum_{j \in [Q]} \sigma_{k,i}^{(j)} \mathcal{R}_k^{(j)} + \mu_{k,i} \mathcal{G} \tag{14}$$

for $i \in [P]$. Therefore, we have

$$\begin{aligned}\bar{\mathcal{R}}_k^{(i)} &= \sum_{j \in [Q]} \sigma_{k,i}^{(j)} \mathcal{R}_k^{(j)} + \mu_{k,i} \mathcal{G} \\ &= \{\mu_{k,i} + \sum_{j \in [Q]} \sigma_{k,i}^{(j)} \zeta_k^{(j)}\} \mathcal{G} - \{\sum_{j \in [Q]} \sigma_{k,i}^{(j)} \eta_k^{(j)}\} \mathcal{D}.\end{aligned}$$

This implies that the re-randomized presignature is

$$\hat{\mathcal{R}}_k^{(i)} = \bar{\mathcal{R}}_k^{(i)} + \delta_{k,i} \mathcal{G} = \{\delta_{k,i} + \mu_{k,i} + \sum_{j \in [Q]} \sigma_{k,i}^{(j)} \zeta_k^{(j)}\} \mathcal{G} - \{\sum_{j \in [Q]} \sigma_{k,i}^{(j)} \eta_k^{(j)}\} \mathcal{D}$$

So to sign a message $m_{k,i}$, the simulator will choose $\delta_{k,i}$ at random, and output the signature $(\hat{\mathcal{R}}_k^{(i)}, z_{k,i})$ along with the value $\delta_{k,i}$, where

$$z_{k,i} \coloneqq \delta_{k,i} + \mu_{k,i} + \sum_{j \in [Q]} \sigma_{k,i}^{(j)} \zeta_k^{(j)},$$

and, additionally, program the random oracle so that

$$H(\langle \mathcal{D} \rangle \parallel \langle \hat{\mathcal{R}}_k^{(i)} \rangle \parallel m_{k,i}) \coloneqq h_{k,i},$$

where

$$h_{k,i} \coloneqq \sum_{j \in [Q]} \sigma_{k,i}^{(j)} \eta_k^{(j)}.$$

The fact that the matrix U_k has full rank and that the $\eta_k^{(j)}$'s are random and independent (of each other as well as everything in the adversary's view, including the values $\sigma_{k,i}^{(j)}$) means that the $h_{k,i}$'s are also random and independent, so the output of the random oracle has the right distribution.

The above analysis carries over in a straightforward way to handle additive key derivation. If the effective key for a given signing request is $\mathcal{D}'_{k,i} = \mathcal{D} + e_{k,i}\mathcal{G}$, then in the above simulation, we have to subtract $e_{k,i}h_{k,i}$ from the value $z_{k,i}$ we originally computed, and program the random oracle at the point corresponding to $\mathcal{D}'_{k,i}$, rather than $\mathcal{D}$. The argument is also easily adapted to deal with batch re-randomization (see Section 2.4.1) — note that the batches of signing requests used in batch re-randomization do not have to align at all with the batches of presignatures.

It is a curious fact that the above proof relies crucially on the assumption that the output space of H is all of $\mathbb{Z}_q$. However, this appears to just be an artifact of the proof, as we can prove security in the GGM without this restriction (see below in Section 4.2).

Relation to SPRINT. Our analysis here highlights and presents in a more simplified and modular form ideas that are already in present in the SPRINT protocol from [BHK$^+$23]. Note that in SPRINT, rather than the $\delta_{k,i}$'s being the output of a Random Beacon, they are actually the output of a hash function modeled as a random oracle. In fact, in SPRINT, the inputs to this random oracle includes a batch of messages $\{m_{k,i}\}_i$ to be signed using the corresponding batch of derived presignatures $\{\bar{\mathcal{R}}_{k,i}\}_i$, so that all of $\delta_{k,i}$'s for this entire batch are generated at once (in fact, just a single δ-value is used for the entire batch). However, the analysis really calls for a Random Beacon, rather than a random oracle. Indeed, the security theorem proved in [BHK⁺23] actually only analyzes an attack with just a single batch of signing requests. It works by guessing which random oracle query represents the Random Beacon, and this (among other things) results in a quite inefficient security reduction. To be useful, one must model an attack in which many batches of signing requests are processed. While [BHK⁺23] is mute on this point, it would appear that their theorem could be extended to prove the security in an attack in which batches of signing requests are processed sequentially. However, this means that only a *single* batch of unused presignatures can be outstanding at a time: if there are many such batches of unused presignatures, the same attack as described in Section 2.5 can be carried out (using one presignature per batch).

This seems to somewhat defeat the purpose of presignatures — the goal is to build up a large cache of presignatures in periods of low demand, so as to be able to quickly process bursts of signing requests in periods of high demand. However, with SPRINT, after a single batch of presignatures is produced, it must be consumed by processing a corresponding batch of signing requests before the next batch of presignatures can be produced. Regardless of the size of these batches, latency and/or throughput will be adversely affected by this restriction. For example, when an individual signing request comes in, we will have to make it wait until the batch of signing requests is full, or we can process it, discarding any unused presignatures in the batch and initiating production of the next batch of presignatures. In the latter case, by discarding unused presignatures, the overall throughput of the system is reduced; moreover, the next signing request that comes in will have to wait for the production of that next batch of presignatures to complete.

4.2 Generic group model analysis

In Section 4.1, we analyzed the scheme with re-randomized presignatures in the random oracle model, giving a reduction to the security of Schnorr's identification scheme. Here, we give an analysis in the generic group model.

Analogous to (14), we have

$$\bar{\mathtt{R}}_k^{(i)} = \sum_{j\in[Q]} \sigma_{k,i}^{(j)} \mathtt{R}_k^{(j)} + \mu_{k,i}, \tag{15}$$

for $i \in [P]$, where the $\mathtt{R}_k^{(j)}$ and $\bar{\mathtt{R}}_k^{(i)}$ are variables which symbolically represent the discrete logarithms of the group elements $\mathcal{R}_k^{(j)}$ and $\bar{\mathcal{R}}_k^{(i)}$.

It is convenient to extend (15) to all $i \in [Q]$. To do this, we can simply add $Q - P$ rows to the matrix U_k and the column vector $\mathbf{u}'_k$ in an arbitrary way, subject to the constrain that U_k is now a nonsingular $Q \times Q$ matrix. This defines a bijective $\mathbb{Z}_q$-linear map between the $\mathbb{Z}_q$-vector spaces $\mathbb{Z}_q + \sum_{j\in[Q]} \mathbb{Z}_q \mathtt{R}_k^{(j)}$ and $\mathbb{Z}_q + \sum_{i\in[Q]} \mathbb{Z}_q \bar{\mathtt{R}}_k^{(i)}$ (which acts as the identity on $\mathbb{Z}_q$).

Note that for a given k, this bijective map is only defined after the adversary specifies the bias $(U_k, \mathbf{u}'_k)$. In the symbolic simulation, when this occurs, we use this bijective map to substitute, throughout $\mathit{Domain}(\pi)$, each variable $\mathtt{R}_k^{(j)}$, for $j \in [Q]$, by its corresponding value in $\mathbb{Z}_q + \sum_{i\in[Q]} \mathbb{Z}_q \bar{\mathtt{R}}_k^{(i)}$ under this map.

Now consider what happens at a later time in the symbolic simulation (after we have already substituted the variables $\mathtt{R}_k^{(j)}$ with the variables $\bar{\mathtt{R}}_k^{(i)}$) when we sign a message $m_{k,i}$ using the derived presignature $\bar{\mathcal{R}}_k^{(i)}$, which is re-randomized as $\hat{\mathcal{R}}_k^{(i)} \coloneqq \bar{\mathcal{R}}_k^{(i)} + \delta_{k,i}\mathcal{G}$. Here, $\delta_{k,i}$ is generated at random by only after the signing request has been made (this corresponds to accessing a Random Beacon in the threshold setting). Here, the symbolic simulation chooses $z_{k,i}, \delta_{k,i} \in \mathbb{Z}_q$ and $\hat{\mathcal{R}}_k^{(i)} \in E$ at random, programs π so that $\pi(\bar{\mathtt{R}}_k^{(i)} + \delta_{k,i}) = \hat{\mathcal{R}}_k^{(i)}$, and makes the substitution

$$\bar{\mathtt{R}}_k^{(i)} \mapsto z_{k,i} - \delta_{k,i} - h_{k,i}\mathtt{D}$$

throughout $\mathit{Domain}(\pi)$.

After defining the symbolic simulation in this way, the rest of the argument is essentially the same as in Section 3.2. The argument is also easily adapted to handle additive key derivation. In addition, the argument is easily adapted to deal with batch re-randomization (see Section 2.4.1) — note that the batches of signing requests used in batch re-randomization do not have to align at all with the batches of presignatures. For all of these variations, we get essentially the same reduction to the various preimage problems as in Section 3.2, with the same concrete security bounds as in Section 3.2.2, except the running times of the various adversaries in the reductions may be somewhat higher.[4] If we are only interested in security bounds in the GGM+ROM, then the same bounds hold.

4.3 Re-randomizing presignatures via hashing

In this section, we present a protocol that combines the technique of re-randomization via hashing (as in Section 3.3) with batch randomness extraction. We analyze the protocol in the GGM plus ROM.

We assume that we create batches of presignatures in pairs. So we first create a batch $\mathcal{R}_k^{(1)}, \ldots, \mathcal{R}_k^{(Q)}$ and a batch $\mathcal{S}_k^{(1)}, \ldots, \mathcal{S}_k^{(Q)}$ of initial presignatures, after which the adversary specifies biases $(U_k, \mathbf{u}'_k)$ and $(V_k, \mathbf{v}'_k)$, from which we then obtain a batch $\bar{\mathcal{R}}_k^{(1)}, \ldots, \bar{\mathcal{R}}_k^{(P)}$ of derived presignatures (using the first bias) a batch $\bar{\mathcal{S}}_k^{(1)}, \ldots, \bar{\mathcal{S}}_k^{(P)}$ of derived presignatures (using the second bias). We also assume that after these biases are given, a random value ρ_k is chosen at random from some set of size at least q and published (in the threshold setting, this corresponds to accessing a Random Beacon, but it is only needed in the offline preprocessing phase).

To sign a message $m_{k,i}$, for any k and $i \in \mathcal{I}$, the derived presignature is

$$\hat{\mathcal{R}}_k^{(i)} \coloneqq \bar{\mathcal{R}}_k^{(i)} + \delta_{k,i}\bar{\mathcal{S}}_k^{(i)},$$

where

$$\delta_{k,i} \coloneqq \Delta(\langle \mathcal{D} \rangle \parallel \langle k \rangle \parallel \langle i \rangle \parallel \langle \rho_k \rangle \parallel m_{k,i}).$$

As in Section 3.3, both Δ and H are modeled as a random oracles.

[4] This is because they need to track variables in *Domain* that may never disappear, so that now the value of L in the additive term $O(LN)$ has to be replaced by a bound on the total number presignatures generated.

NOTES:

1. If we use additive key derivation, then the signing request includes $e_{k,i} \in \mathbb{Z}_q$, and we replace the public key $\mathcal{D}$ by $\mathcal{D}'_{k,i} \coloneqq \mathcal{D} + e_{k,i}\mathcal{G}$ in the calculation of $\delta_{k,i}$, and we replace the secret key d by $d + e_{i,k}$ in the signing algorithm. While we will not analyze this variation in detail, the analysis we present applies equally well to this variation.
2. We can easily apply an additional layer of batching, so that we generate many batches at once. This means we can use "batched asynchronous VSS" protocols to more efficiently generate many dealings at once (for example, as in [GS23]), leading to even more efficient protocols.

Analogous to (15), we have

$$\bar{\mathtt{S}}_k^{(i)} = \sum_{j \in [Q]} \tau_{k,i}^{(j)} \mathtt{R}_k^{(j)} + \nu_{k,i}, \tag{16}$$

for $i \in [P]$, where $\mathtt{S}_k^{(j)}$ and $\bar{\mathtt{S}}_k^{(i)}$ are variables which symbolically represent the discrete logarithms of the group elements $\mathcal{S}_k^{(j)}$ and $\bar{\mathcal{S}}_k^{(i)}$. Just as we did in relation to (15), we can extend this to all $i \in [Q]$. This defines a bijective $\mathbb{Z}_q$-linear map between the $\mathbb{Z}_q$-vector spaces $\mathbb{Z}_q + \sum_{j \in [Q]} \mathbb{Z}_q \mathtt{S}_k^{(j)}$ and $\mathbb{Z}_q + \sum_{i \in [Q]} \mathbb{Z}_q \bar{\mathtt{S}}_k^{(i)}$ (which acts as the identity on $\mathbb{Z}_q$).

Analogous to what we did in Section 4.2, in the symbolic simulation, when the corresponding bias has been specified, and this bijective map has been determined, we use this bijective map to substitute, throughout $\mathit{Domain}(\pi)$, each variable $\mathtt{R}_k^{(j)}$, for $j \in [Q]$, by its corresponding value in $\mathbb{Z}_q + \sum_{i \in [Q]} \mathbb{Z}_q \bar{\mathtt{R}}_k^{(i)}$ under this map, and each variable $\mathtt{S}_k^{(j)}$, for $j \in [Q]$, by its corresponding value in $\mathbb{Z}_q + \sum_{i \in [Q]} \mathbb{Z}_q \bar{\mathtt{S}}_k^{(i)}$.

Analogous to what we did in Section 3.3, when a signing query on a message $m_{k,i}$ is made that uses the derived presignature $(\bar{\mathcal{R}}_{k,i}, \bar{\mathcal{S}}_{k,i})$, a preliminary computation

$$\begin{aligned}
\delta_{k,i} &\leftarrow \Delta(\langle \mathcal{D} \rangle \parallel \langle k \rangle \parallel \langle i \rangle \parallel \langle \rho_k \rangle \parallel m_{k,i}), \\
\hat{\mathcal{R}}_{k,i} &\leftarrow \bar{\mathcal{R}}_{k,i} + \delta_{k,i} \bar{\mathcal{S}}_{k,i}, \\
h_{k,i} &\leftarrow H(\langle \mathcal{D} \rangle \parallel \langle \hat{\mathcal{R}}_{k,i} \rangle \parallel m_{k,i})
\end{aligned}$$

is made. WLOG, we can assume that the adversary has already computed these values himself. Next, the signing oracle generates $z_{k,i}$ at random. Before returning the signature $(\hat{\mathcal{R}}_{k,i}, z_k)$, the signing oracle also substitutes

$$\bar{\mathtt{R}}_k^{(i)} \mapsto z_{k,i} - \delta_{k,i} \bar{\mathtt{S}}_k^{(i)} - h_{k,i} \mathtt{D} \tag{17}$$

throughout $\mathit{Domain}(\pi)$.

Analogous to what we did in Section 3.3, we make some additional assumptions on the adversary. Namely, after the bias has been specified for the kth pair of batches, and the corresponding value ρ_k has been generated, we ensure that whenever the adversary makes a random oracle query

$$\delta_{k,i} \leftarrow \Delta(\langle \mathcal{D} \rangle \parallel \langle k \rangle \parallel \langle i \rangle \parallel \langle \rho_k \rangle \parallel m_{k,i}),$$

it immediately makes a "special `add` query" to the group oracle

$$(\mathtt{add}, \bar{\mathcal{R}}_{k,i}, \bar{\mathcal{S}}_{k,i}, 1, \delta_{k,i})$$

to obtain the encoding of the group element $\hat{\mathcal{R}}_{k,i} \leftarrow \bar{\mathcal{R}}_{k,i} + \delta_{k,i}\bar{\mathcal{S}}_{k,i}$. We may also assume that it then immediately makes the random oracle query

$$h_{k,i} \leftarrow H(\langle \mathcal{D} \rangle \parallel \langle \hat{\mathcal{R}}_{k,i} \rangle \parallel m_{k,i}).$$

Analogous to what we did in Section 3.3, suppose the forgery is a signature $(\mathcal{R}^*, z^*)$ on a message m^*. We may assume that $\mathcal{R}^*$ was randomly generated by the a group oracle query — otherwise, the adversary must essentially win Preimage Attack III on H (as in Section 3.1.4, but where H is modeled as a random oracle).

Suppose that initially

$$\begin{aligned}\pi^{-1}(\mathcal{R}^*) = a + b\mathtt{D} +& \\ &\sum_{\ell,j}(c_{\ell,j}\mathtt{R}_\ell^{(j)} + d_{\ell,j}\mathtt{S}_\ell^{(j)}) + \\ &\sum_{k,i}(\bar{c}_{k,i}\bar{\mathtt{R}}_k^{(i)} + \bar{d}_{k,i}\bar{\mathtt{S}}_k^{(i)}).\end{aligned}$$

Here,

- the sum on ℓ, j corresponds to presignatures from pairs of batches whose bias has not yet been specified (and whose corresponding random values ρ_ℓ have not yet been generated), while
- the sum on k, i corresponds to presignatures from pairs of batches whose bias been specified (and whose corresponding random values ρ_ℓ have been generated).

Note that all of the constants a, b, $c_{\ell,j}, d_{\ell,j}$, $\bar{c}_{k,i}, \bar{d}_{k,i}$ are fixed before $\mathcal{R}^*$ is generated at random. In order for the forgery to be valid, all of the variables except $\mathtt{D}$ must be eliminated by substitution so as to end up with

$$\pi^{-1}(\mathcal{R}^*) = z^* - h^*\mathtt{D}.$$

We argue below that the sum on ℓ, j must be empty, as otherwise the forgery will be valid with only negligible probability. Assuming this for now, we focus on the sum on k, i. After substitution, we have

$$\pi^{-1}(\mathcal{R}^*) = \underbrace{\{a + \sum_{k,i}\bar{c}_{k,i}z_{k,i}\}}_{=z^*} + \underbrace{\{b - \sum_{k,i}\bar{c}_{k,i}h_{k,i}\}}_{=-h^*}\mathtt{D} + \sum_{k,i}\underbrace{\{\bar{d}_{k,i} - \bar{c}_{k,i}\delta_{k,i}\}}_{=0}\bar{\mathtt{S}}_k^{(i)}. \qquad (18)$$

For the forgery to be valid, we must have $\bar{d}_{k,i} - \bar{c}_{k,i}\delta_{k,i} = 0$ for each k, i.

The rest of the argument is analogous to what we did in the Generic Group Model analysis in Section 3.3. Namely, if the forgery is to be valid, then with overwhelming probability, the adversary must have already made queries to Δ that produce the outputs $\bar{d}_{k,i}/\bar{c}_{k,i}$. Thus, at the time we generate $\mathcal{R}^*$, the inputs to H that determine the $h_{k,i}$'s have already been determined. Therefore, in order for the adversary to find m^*, he must essentially win Preimage Attack I or II on H (as in Section 3.1.4, but where H is modeled as a random oracle).

We now return to the claim that the sum on ℓ, j must be empty. Suppose it is not. Then for some ℓ the corresponding the bias for the corresponding pair of batches has not yet been specified at the time $\mathcal{R}^*$ is generated, which means that the corresponding random value ρ_ℓ has not yet been generated either. For the forgery to be valid, the corresponding bias must be specified, which only then determines values $\bar{c}_{\ell,i}$ and $\bar{d}_{\ell,i}$ before ρ_ℓ is generated, and then we must also make the coefficient $\bar{d}_{\ell,i} - \bar{c}_{\ell,i}\delta_{\ell,i}$ on $\bar{\mathtt{S}}_{\ell,i}$ vanish for each i via substitution. Since ρ_ℓ is input to the hash Δ to determine each $\delta_{\ell,i}$, the probability that the adversary can find other inputs to Δ so that $\bar{d}_{\ell,i} - \bar{c}_{\ell,i}\delta_{\ell,i} = 0$ for each i will be negligible.

Concrete security bounds. One can show that if H is modeled as a random oracle with an output space of size M, the adversary's forging advantage is $O(N^2/q + N/M)$.

References

[BHK+23] F. Benhamouda, S. Halevi, H. Krawczyk, Y. Ma, and T. Rabin. SPRINT: High-throughput robust distributed schnorr signatures. Cryptology ePrint Archive, Paper 2023/427, 2023. URL https://eprint.iacr.org/2023/427. https://eprint.iacr.org/2023/427.

[BL19] J. Blocki and S. Lee. On the multi-user security of short schnorr signatures with preprocessing. Cryptology ePrint Archive, Paper 2019/1105, 2019. https://eprint.iacr.org/2019/1105.

[BLS01] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the weil pairing. In C. Boyd, editor, *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–532. Springer, 2001.

[Bol03] A. Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In Y. Desmedt, editor, *Public Key Cryptography - PKC 2003, 6th International Workshop on Theory and Practice in Public Key Cryptography, Miami, FL, USA, January 6-8, 2003, Proceedings*, volume 2567 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2003.

[Cer10] Certicom Research. Sec 2: Recommended elliptic curve domain parameters, 2010. Version 2.0, http://www.secg.org/sec2-v2.pdf.

[CKM21] E. Crites, C. Komlo, and M. Maller. How to prove schnorr assuming schnorr: Security of multi- and threshold signatures. Cryptology ePrint Archive, Paper 2021/1375, 2021. https://eprint.iacr.org/2021/1375.

[DEF+18] M. Drijvers, K. Edalatnejad, B. Ford, E. Kiltz, J. Loss, G. Neven, and I. Stepanovs. On the security of two-round multi-signatures. Cryptology ePrint Archive, Paper 2018/417, 2018. https://eprint.iacr.org/2018/417.

[DFI22] The DFINITY Team. The internet computer for geeks. Cryptology ePrint Archive, Report 2022/087, 2022. https://ia.cr/2022/087.

[DN07] I. Damgård and J. B. Nielsen. Scalable and unconditionally secure multiparty computation. In A. Menezes, editor, *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings*, volume 4622 of *Lecture Notes in Computer Science*, pages 572–590. Springer, 2007.

[FY92] M. K. Franklin and M. Yung. Communication complexity of secure computation (extended abstract). In S. R. Kosaraju, M. Fellows, A. Wigderson, and J. A. Ellis, editors, *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada*, pages 699–710. ACM, 1992.

[GG20] R. Gennaro and S. Goldfeder. One round threshold ECDSA with identifiable abort. Cryptology ePrint Archive, Report 2020/540, 2020. https://ia.cr/2020/540.

[GJKR07] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure distributed key generation for discrete-log based cryptosystems. *J. Cryptol.*, 20(1):51–83, 2007.

[GS21] J. Groth and V. Shoup. On the security of ECDSA with additive key derivation and presignatures. Cryptology ePrint Archive, Report 2021/1330, 2021. https://ia.cr/2021/1330.

[GS22] J. Groth and V. Shoup. Design and analysis of a distributed ECDSA signing service. Cryptology ePrint Archive, Report 2022/506, 2022. https://ia.cr/2022/506.

[GS23] J. Groth and V. Shoup. Fast batched asynchronous distributed key generation. Cryptology ePrint Archive, Paper 2023/1175, 2023. https://eprint.iacr.org/2023/1175.

[HN06] M. Hirt and J. B. Nielsen. Robust multiparty computation with linear communication complexity. In C. Dwork, editor, *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*, volume 4117 of *Lecture Notes in Computer Science*, pages 463–482. Springer, 2006.

[KG20] C. Komlo and I. Goldberg. Frost: Flexible round-optimized schnorr threshold signatures. Cryptology ePrint Archive, Paper 2020/852, 2020. https://eprint.iacr.org/2020/852.

[NSW09] G. Neven, N. P. Smart, and B. Warinschi. Hash function requirements for schnorr signatures. *J. Math. Cryptol.*, 3(1):69–87, 2009.

[PS96] D. Pointcheval and J. Stern. Provably secure blind signature schemes. In K. Kim and T. Matsumoto, editors, *Advances in Cryptology - ASIACRYPT '96, International Conference on the Theory and Applications of Cryptology and Information Security, Kyongju, Korea, November 3-7, 1996, Proceedings*, volume 1163 of *Lecture Notes in Computer Science*, pages 252–265. Springer, 1996.

[RRJ+22] T. Ruffing, V. Ronge, E. Jin, J. Schneider-Bensch, and D. Schröder. ROAST: Robust asynchronous schnorr threshold signatures. Cryptology ePrint Archive, Paper 2022/550, 2022. URL https://eprint.iacr.org/2022/550. https://eprint.iacr.org/2022/550.

[Wag02] D. A. Wagner. A generalized birthday problem. In M. Yung, editor, *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, volume 2442 of *Lecture Notes in Computer Science*, pages 288–303. Springer, 2002.

[Wui20] P. Wuille. Bip32: Hierarchical deterministic wallets, 2020. https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki.

[Zha22] M. Zhandry. To label, or not to label (in generic groups). Cryptology ePrint Archive, Paper 2022/226, 2022. https://eprint.iacr.org/2022/226.

A Proof of Lemma 1

In order to make certain arguments simpler, we shall replace our lazy simulator in Figure 2 by a slightly more lazy simulator. This simulator may `abort` under certain conditions, which means the entire experiment halts and no forgery is produced.

Lazy-Sim1:

```
1. Initialization:
   (a) $\pi \leftarrow \{(0, \mathcal{O})\}$.
   (b) $d \xleftarrow{\$} \mathbb{Z}_q$
   (c) invoke $(\texttt{map}, 1)$ to obtain $\mathcal{G}$
   (d) invoke $(\texttt{map}, d)$ to obtain $\mathcal{D}$
   (e) return $(\mathcal{G}, \mathcal{D})$
2. To process a group oracle query $(\texttt{map}, i)$:
   (a) if $i \notin Domain(\pi)$:
       i. $\mathcal{P} \xleftarrow{\$} E$; if $\mathcal{P} \in Range(\pi)$ then abort
       ii. add $(-i, -\mathcal{P})$ and $(i, \mathcal{P})$ to $\pi$
   (b) return $\pi(i)$
3. To process a group oracle query $(\texttt{add}, \mathcal{P}_1, \mathcal{P}_2, c_1, c_2)$:
   (a) for $j = 1, 2$: if $\mathcal{P}_j \notin Range(\pi)$:
       i. $i \xleftarrow{\$} \mathbb{Z}_q$; if $i \in Domain(\pi)$ then abort
```

```
        ii. add $(-i, -\mathcal{P}_j)$ and $(i, \mathcal{P}_j)$ to $\pi$
    (b) invoke $(\texttt{map}, c_1\pi^{-1}(\mathcal{P}_1) + c_2\pi^{-1}(\mathcal{P}_2))$ and return the result
4. To process a request to sign $m$:
    (a) $r \xleftarrow{\$} \mathbb{Z}_q$;  if $r \in \mathit{Domain}(\pi)$ then abort
    (b) invoke $(\texttt{map}, r)$ to get $\mathcal{R}$
    (c) $h \leftarrow H(\langle\mathcal{D}\rangle \parallel \langle\mathcal{R}\rangle \parallel m) \in \mathbb{Z}_q$
    (d) $z \leftarrow r + hd$
    (e) return $(\mathcal{R}, z)$
```

The changes are highlighted. It is trivial to verify that the adversary's forging advantage in this game differs from that in the original attack game by $O(N^2/q)$. Indeed, we can view both games as operating on the same sample space, and both games proceed identically unless a specific failure event occurs in the Lazy-Sim1 game. One sees that this failure event occurs with probability $O(N^2/q)$.

Now we modify the logic for processing signing requests, as follows:

```
4. To process a request to sign $m$:
    (a) $\mathcal{R} \xleftarrow{\$} E$, $z \xleftarrow{\$} \mathbb{Z}_q$
    (b) $h \leftarrow H(\langle\mathcal{D}\rangle \parallel \langle\mathcal{R}\rangle \parallel m) \in \mathbb{Z}_q$
    (c) $r \leftarrow z - hd$
    (d) if $r \in \mathit{Domain}(\pi)$ or $\mathcal{R} \in \mathit{Range}(\pi)$ then abort
    (e) add $(-r, -\mathcal{R})$ and $(r, \mathcal{R})$ to $\pi$
    (f) return $(\mathcal{R}, z)$
```

Let us call this **Lazy-Sim2**. It is easy to verify that this *perfectly* simulates the behavior of Lazy-Sim1, and so the adversary's forgery advantage does not change at all. Indeed, both simulators generate r and $\mathcal{R}$ at random and abort if $r \in \mathit{Domain}(\pi)$ or $\mathcal{R} \in \mathit{Range}(\pi)$.

We now define a **symbolic** simulation of the attack game. The essential difference in this game is that $\mathit{Domain}(\pi)$ will now consist of polynomials of the form $a + b\texttt{D}$, where $a, b \in \mathbb{Z}_q$ and $\texttt{D}$ is an indeterminant. Note that π will otherwise still satisfy all of the requirements of an encoding function. The simulator is identical to Lazy-Sim2, except as highlighted:

Symbolic-Sim0:

```
1. Initialization:
    (a) $\pi \leftarrow \{(0, \mathcal{O})\}$.
    (b) $d \xleftarrow{\$} \mathbb{Z}_q$
    (c) invoke $(\texttt{map}, 1)$ to obtain $\mathcal{G}$
    (d) invoke $(\texttt{map}, \texttt{D})$ to obtain $\mathcal{D}$
    (e) return $(\mathcal{G}, \mathcal{D})$
2. To process a group oracle query $(\texttt{map}, i)$:
    (a) if $i \notin \mathit{Domain}(\pi)$:
        i. $\mathcal{P} \xleftarrow{\$} E$;  if $\mathcal{P} \in \mathit{Range}(\pi)$ then abort
        ii. add $(-i, -\mathcal{P})$ and $(i, \mathcal{P})$ to $\pi$
    (b) return $\pi(i)$

3. To process a group oracle query $(\texttt{add}, \mathcal{P}_1, \mathcal{P}_2, c_1, c_2)$:
    (a) for $j = 1, 2$: if $\mathcal{P}_j \notin \mathit{Range}(\pi)$:
        i. $i \xleftarrow{\$} \mathbb{Z}_q$; if $i \in \mathit{Domain}(\pi)$ then abort
        ii. add $(-i, -\mathcal{P}_j)$ and $(i, \mathcal{P}_j)$ to $\pi$
    (b) invoke $(\texttt{map}, c_1\pi^{-1}(\mathcal{P}_1) + c_2\pi^{-1}(\mathcal{P}_2))$ and return the result

4. To process a request to sign $m$:
    (a) $\mathcal{R} \xleftarrow{\$} E$, $z \xleftarrow{\$} \mathbb{Z}_q$
    (b) $h \leftarrow H(\langle\mathcal{D}\rangle \parallel \langle\mathcal{R}\rangle \parallel m) \in \mathbb{Z}_q$
    (c) $r \leftarrow z - h\mathtt{D}$
    (d) if $r \in \mathit{Domain}(\pi)$ or $\mathcal{R} \in \mathit{Range}(\pi)$ then abort
    (e) add $(-r, -\mathcal{R})$ and $(r, \mathcal{R})$ to $\pi$
    (f) return $(\mathcal{R}, z)$
```

Note that while the simulator may invoke `map` with non-constant polynomials, the adversay does not.

We can model the attack game with respect to both simulators Lazy-Sim2 and Symbolic-Sim0, with each operating on the same underlying sample space. This sample space comprises the random tape of the adversary an the random choices made by the simulators (including d). That is, the outcomes of both games are determined by these values in the sample space, although the computations performed in each game are different, and so the outcomes of the games may differ.

Let us define the following Event Z, which we define in terms of the Symbolic-Sim0 attack game. For a polynomial P in $\mathbb{Z}_q[\mathtt{D}]$, we define $[P] \in \mathbb{Z}_q$ to be the value of P with $\mathtt{D}$ replaced by d. For a set S of such polynomials, we define $[S] := \{[P] : P \in S\}$. Event Z is the event that at one of the highlighted tests of the form "$P \in \mathit{Domain}(\pi)$", we have $P \notin \mathit{Domain}(\pi)$ but $[P] \in [\mathit{Domain}(\pi)]$.

We claim that these two games proceed identically unless Z occurs. This should be clear. It follows that the forging probability in these two games differs by at most $\Pr[Z]$.

It should also be clear from the Schwartz-Zippel Lemma that $\Pr[Z] = O(N^2/q)$. Here, we use the fact that in the Symbolic-Sim0 attack game, the value of d is independent of the coefficients of the polynomials that determine Event Z.

Note that Symbolic-Sim0 as defined here is identical to Symbolic-Sim defined in Figure 3, except that in the latter, we have deleted the initialization of d in Step 1(b) of the former, as d is not actually needed. That proves the lemma.

B Proof of Lemma 2

In order to make certain arguments simpler, we replace our lazy simulator in Figure 4 by a slightly more lazy simulator, which may `abort` under certain conditions which means the entire experiment halts and no forgery is produced.

Lazy-Sim1:

```
1. Initialization:
    (a) $\pi \leftarrow \{(0, \mathcal{O})\}$.
    (b) $d \xleftarrow{\$} \mathbb{Z}_q$
    (c) invoke $(\texttt{map}, 1)$ to obtain $\mathcal{G}$
    (d) invoke $(\texttt{map}, d)$ to obtain $\mathcal{D}$
    (e) $k \leftarrow 0$; $K \leftarrow \emptyset$
    (f) return $(\mathcal{G}, \mathcal{D})$
2. To process a group oracle query $(\texttt{map}, i)$:
    (a) if $i \notin Domain(\pi)$:
        i. $\mathcal{P} \xleftarrow{\$} E$; if $\mathcal{P} \in Range(\pi)$ then $\texttt{abort}$
        ii. add $(-i, -\mathcal{P})$ and $(i, \mathcal{P})$ to $\pi$
    (b) return $\pi(i)$
3. To process a group oracle query $(\texttt{add}, \mathcal{P}_1, \mathcal{P}_2, c_1, c_2)$:
    (a) for $j = 1, 2$: if $\mathcal{P}_j \notin Range(\pi)$:
        i. $i \xleftarrow{\$} \mathbb{Z}_q$; if $i \in Domain(\pi)$ then $\texttt{abort}$
        ii. add $(-i, -\mathcal{P}_j)$ and $(i, \mathcal{P}_j)$ to $\pi$
    (b) invoke $(\texttt{map}, c_1\pi^{-1}(\mathcal{P}_1) + c_2\pi^{-1}(\mathcal{P}_2))$ and return the result
4. To process a presignature request:
    (a) $k \leftarrow k + 1$; $K \leftarrow K \cup \{k\}$
    (b) $r_k \xleftarrow{\$} \mathbb{Z}_q$
    (c) invoke $(\texttt{map}, r_k)$ to get $\mathcal{R}_k$
    (d) return $\mathcal{R}_k$
5. To process a request to sign $m_k$ using presignature number $k \in K$:
    (a) $\delta_k \xleftarrow{\$} \mathbb{Z}_q$; $r'_k \leftarrow r_k + \delta_k$; if $r'_k \in Domain(\pi)$ then $\texttt{abort}$
    (b) invoke $(\texttt{map}, r'_k)$ to get $\mathcal{R}'_k$
    (c) $h_k \leftarrow H(\langle \mathcal{D} \rangle \parallel \langle \mathcal{R}'_k \rangle \parallel m_k) \in \mathbb{Z}_q$
    (d) $z_k \leftarrow r_k + \delta_k + h_k d$
    (e) $K \leftarrow K \setminus \{k\}$; return $(\mathcal{R}'_k, z_k, \delta_k)$
```

The changes are highlighted. It is trivial to verify that the adversary's forging advantage in this game differs from that in the original attack game by $O(N^2/q)$.

We now modify the way signing requests are modified as follows:

```
5. To process a request to sign $m_k$ using presignature number $k \in K$:
    (a) $\delta_k \xleftarrow{\$} \mathbb{Z}_q$; $r'_k \leftarrow r_k + \delta_k$
    (b) $\mathcal{R}'_k \xleftarrow{\$} E$
    (c) if $r'_k \in Domain(\pi)$ or $\mathcal{R}'_k \in Range(\pi)$ then $\texttt{abort}$
```

```
   (d) add $(r'_k, \mathcal{R}'_k)$ and $(-r'_k, -\mathcal{R}'_k)$ to $\pi$
   (e) $h_k \leftarrow H(\langle \mathcal{D} \rangle \parallel \langle \mathcal{R}'_k \rangle \parallel m_k) \in \mathbb{Z}_q$
   (f) $z_k \leftarrow r_k + \delta_k + h_k d$
   (g) $K \leftarrow K \setminus \{k\}$; return $(\mathcal{R}'_k, z_k, \delta_k)$
```

Let us call this **Lazy-Sim2**. It is easy to verify that this *perfectly* simulates the behavior of Lazy-Sim1, and so the adversary's forgery advantage does not change at all.

We now define a **symbolic** simulation of the attack game. The essential difference in this game is that $\mathit{Domain}(\pi)$ will now consist of linear polynomials over $\mathbb{Z}_q$ in the indeterminants $\mathtt{D}, \mathtt{R}_1, \mathtt{R}_2, \ldots$. The simulator is identical to Lazy-Sim2, except as highlighted:

Symbolic-Sim0:

```
1. Initialization:
   (a) $\pi \leftarrow \{(0, \mathcal{O})\}$.
   (b) $d \xleftarrow{\$} \mathbb{Z}_q$
   (c) invoke $(\mathtt{map}, 1)$ to obtain $\mathcal{G}$
   (d) invoke $(\mathtt{map}, \mathtt{D})$ to obtain $\mathcal{D}$
   (e) $k \leftarrow 0$; $K \leftarrow \emptyset$
   (f) return $(\mathcal{G}, \mathcal{D})$

2. To process a group oracle query $(\mathtt{map}, i)$:
   (a) if $i \notin \mathit{Domain}(\pi)$:
       i. $\mathcal{P} \xleftarrow{\$} E$; if $\mathcal{P} \in \mathit{Range}(\pi)$ then abort
       ii. add $(-i, -\mathcal{P})$ and $(i, \mathcal{P})$ to $\pi$
   (b) return $\pi(i)$

3. To process a group oracle query $(\mathtt{add}, \mathcal{P}_1, \mathcal{P}_2, c_1, c_2)$:
   (a) for $j = 1, 2$: if $\mathcal{P}_j \notin \mathit{Range}(\pi)$:
       i. $i \xleftarrow{\$} \mathbb{Z}_q$; if $i \in \mathit{Domain}(\pi)$ then abort
       ii. add $(-i, -\mathcal{P}_j)$ and $(i, \mathcal{P}_j)$ to $\pi$
   (b) invoke $(\mathtt{map}, c_1\pi^{-1}(\mathcal{P}_1) + c_2\pi^{-1}(\mathcal{P}_2))$ and return the result

4. To process a presignature request:
   (a) $k \leftarrow k + 1$; $K \leftarrow K \cup \{k\}$
   (b) $r_k \xleftarrow{\$} \mathbb{Z}_q$
   (c) invoke $(\mathtt{map}, \mathtt{R}_k)$ to get $\mathcal{R}_k$
   (d) return $\mathcal{R}_k$

5. To process a request to sign $m_k$ using presignature number $k \in K$:
   (a) $\delta_k \xleftarrow{\$} \mathbb{Z}_q$; $r'_k \leftarrow \mathtt{R}_k + \delta_k$
   (b) $\mathcal{R}'_k \xleftarrow{\$} E$
   (c) if $r'_k \in \mathit{Domain}(\pi)$ or $\mathcal{R}'_k \in \mathit{Range}(\pi)$ then abort
```

(d) add $(r'_k, \mathcal{R}'_k)$ and $(-r'_k, -\mathcal{R}'_k)$ to π

(e) $h_k \leftarrow H(\langle \mathcal{D} \rangle \parallel \langle \mathcal{R}'_k \rangle \parallel m_k) \in \mathbb{Z}_q$

(f) $z_k \leftarrow r_k + \delta_k + h_k d$;
substitute $\mathtt{R}_k \mapsto z_k - \delta_k - h_k \mathtt{D}$ throughout $\mathit{Domain}(\pi)$ and abort if $\mathit{Domain}(\pi)$ "collapses" (i.e., two distinct elements of $\mathit{Domain}(\pi)$ become equal after the substitution)

(g) $K \leftarrow K \setminus \{k\}$; return $(\mathcal{R}'_k, z_k, \delta_k)$

Let us define the following Event Z, which we define in terms of the Symbolic-Sim0 attack game. For a polynomial P in $\mathbb{Z}_q[\mathtt{D}, \mathtt{R}_1, \mathtt{R}_2, \ldots]$, we define $[P] \in \mathbb{Z}_q$ to be the value of P with $\mathtt{D}$ replaced by d, $\mathtt{R}_1$ replaced by r_1, $\mathtt{R}_2$ replaced by r_2, and so on. For a set S of such polynomials, we define $[S] := \{[P] : P \in S\}$. Event Z is the event that at one of the highlighted tests of the form "$P \in \mathit{Domain}(\pi)$", we have $P \notin \mathit{Domain}(\pi)$ but $[P] \in [\mathit{Domain}(\pi)]$.

We claim that the Lazy-Sim2 and Symbolic-Sim0 attack games proceed identically unless Z occurs. This should be clear — in particular, so long as Z does not occur, the subsitution in Step 5(f) will not fail. It follows that the forging probability in these two games differs by at most $\Pr[Z]$.

It should also be clear that $\Pr[Z] = O(N^2/q)$ — this follows from the Schwartz-Zippel Lemma and the following observation:

> at any point in time in the Symbolic-Sim0 game, the polynomials that determine Event Z involve the variables $\mathtt{D}$ and $\{\mathtt{R}_k\}_{k \in K}$, and the coefficients of these polynomials are independent of d and $\{r_k\}_{k \in K}$.

It should also be clear that this symbolic attack game is equivalent to the one in Figure 5 — to move from the former to the latter, we simply drop the computation of d and the r_k's, and replace the z_k's by random elements of $\mathbb{Z}_q$. That proves the lemma.