The many faces of Schnorr: a toolkit for the modular design of threshold Schnorr signatures

Victor Shoup
Offchain Labs

January 17, 2025

Abstract

Recently, a number of highly optimized threshold signing protocols for Schnorr signatures have been proposed. While these proposals contain important new techniques, some of them present and analyze these techniques in very specific contexts, making it less than obvious how these techniques can be adapted to other contexts, or combined with one another. The main goal of this paper is to abstract out and extend in various ways some of these techniques, building a toolbox of techniques that can be easily combined in different ways and in different contexts. To this end, we present security results for various "enhanced" modes of attack on the Schnorr signature scheme in the non-distributed setting, and we demonstrate how to reduce the security in the distributed threshold setting to these enhanced modes of attack in the non-distributed setting. This results in a very modular approach to protocol design and analysis, which can be used to easily design new threshold Schnorr protocols that enjoy better security and/or performance properties than existing ones.

1 Introduction

Recently, a number of highly optimized threshold signing protocols for Schnorr signatures have been proposed [KG20, CKM21, BHK+24, GS24]. While these proposals contain important new techniques, some of them (notably, [KG20, CKM21, BHK+24]) present and analyze these techniques in very specific contexts, making it less than obvious how these techniques can be adapted to other contexts or combined with one another.

The main goal of this paper is to abstract out and extend in various ways some of these techniques, building a toolbox of techniques that can be easily combined in different ways and in different contexts. To this end, we present security results for various "enhanced" modes of attack on the Schnorr signature scheme in the non-distributed setting. Based on our experience, a good design methodology is to reduce the security of a distributed signing protocol to such an enhanced attack mode in the non-distributed setting. This approach was taken in the papers [GS22a, GS22b] for ECDSA — the paper [GS22b] presented the analysis for various enhanced modes of attack on ECDSA in the non-distributed setting, while [GS22a] gave distributed protocols for ECDSA along with security reductions to these

attack modes. Moreover, the paper [GS24] does exactly the same for distributed Schnorr — it gives new distributed protocols for Schnorr along with security reductions to the enhanced attack modes in the non-distributed setting that we analyze here in this paper. For completeness, we review those reductions here, filling in a number of details along the way.

We believe this modular approach can and should be adapted to other settings and can lead to better protocol designs. Typically, any particular distributed protocol will represent a compromise between many different goals (attack models, security assumptions, synchrony assumptions, communication complexity, robustness requirements, latency, computational complexity, etc). Focusing on one such design choice and then presenting a monolithic security proof can obscure the underlying security techniques and impede their adoption in alternative settings.

1.1 Background

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-distributed 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 (together with the sharing of the ephemeral secret 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. While computing presignatures in this way can improve *latency*, it does not by itself improve *throughput*. However, since such presignatures

can be generated in large batches, one can potentially also increase throughput as well by utilizing "batching" techniques. Indeed, various types of batching techniques are used in precisely this way in [BHK+24] and [GS24].

Unfortunately, using presignatures naively in this way breaks the security of Schnorr signatures. Indeed, the usual proof of security of ordinary, non-distributed 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 (and even polynomial) attacks on signing protocols that use presignatures in this way [DEF+19, BLL+22] (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 [CGG+20]. 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 many parties distributed around the globe, such 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+24] 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, due to the particular technique it uses to mitigate against presignature attacks. 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+24] 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 if many such batches of presignatures are generated in advance, the same subexponential attacks mentioned above can be used on SPRINT.

We mention here another technique that can be used to mitigate against presignature attacks, which is to simply add a group element $\delta\mathcal{G}$ to the presignature, so that the "effective presignature" is $\mathcal{R} + \delta\mathcal{G}$. Here, the "shift amount" δ is obtained from a "random beacon"

after the message to be signed has already been specified — such a random beacon should remain unpredictable until at least one honest party chooses to reveal it. A random beacon can be easily implemented in a number of ways (by generating a BLS threshold signature or by opening a previously secret-shared random value). However, this may add some latency to the online signing phase. We call such a shifted presignature a **re-randomized presignature**.

NOTES:

1. As we will discuss, in some settings, it may be possible to implement the random beacon without introducing any extra latency.
2. SPRINT can be viewed as using a variation of re-randomized presignatures, where the shift amount δ is obtained from a hash function (which may be modeled as a random oracle) rather than from a random beacon. While this distinction may seem small, it in fact has significant security implications (as we shall discuss).
3. FROST's mitigation technique can also be viewed as a variation of re-randomized presignatures, where the shift amount δ is obtained from a hash function and effective presignature is $\mathcal{R} + \delta\mathcal{S}$, where $\mathcal{S} \in E$ is a second ephemeral public key. Compared to the basic re-randomized presignatures technique, the FROST mitigation technique avoids the extra latency in the online phase incurred by the random beacon. However, this comes at the cost of essentially doubling the amount of work that needs to be done per presignature in the offline phase. It also may increase the computation cost of the online signing phase (an issue discussed at greater length in [GS24]).
4. The re-randomized presignatures technique was used in the context of ECDSA in [GS22b]. The use of re-randomized presignatures for Schnorr signatures appears in somewhat different form in [KG20]: although the FROST protocol itself does not use a random beacon, the only rigorous analysis in [KG20] is for a variant of FROST called FROST-Interactive which derives the shift amount δ using (something similar to) a random beacon (but still uses the second ephemeral public key $\mathcal{S}$). The follow-up paper [CKM21] provides a rigorous analysis (in the random oracle model) of FROST2, which is a slight variation of FROST.

1.2 Our contributions

Our main technical contributions are as follows:

1. *An analysis of the re-randomized presignature technique.* We show how to model re-randomized presignatures (based on a random beacon) using an appropriate enhanced attack mode in the non-distributed setting, and present several analyses of this mitigation technique, including a reduction to Discrete Logarithm, as well as a direct analysis in the Generic Group Model (GGM), with hash functions modeled both as random oracles and as concrete functions satisfying specific security properties. The GGM analysis gives a much tighter security bound than that obtained by going through a reduction to Discrete Logarithm.
2. *A direct analysis of the FROST presignature-attack mitigation technique in the GGM.* We show how to model FROST's mitigation strategy using an appropriate enhanced attack mode in the non-distributed setting, and give a direct analysis of this in the GGM (modeling hash functions as random oracles). Previous work showed FROST

secure in the random oracle model under the one-more discrete log (OMDL) assumption [CKM21], and showed OMDL hard in the GGM [BFP21]. However, that chain of reductions is very loose. Our results here give much better security bounds in the GGM.

3. *An analysis of SPRINT with re-randomized presignatures in the GGM.* The analysis of SPRINT in [BHK+24] is based on a presignature-attack mitigation technique based on a random oracle, and gives a reduction to Discrete Logarithm. We mentioned above that SPRINT is only secure in very limited modes of operation, due to the particulars of this mitigation technique. We show that these restrictions can be lifted if we instead use re-randomized presignatures (based on a random beacon). We show how to model this using an appropriate enhanced attack mode in the non-distributed setting, and present several analyses, including a direct security analysis in the GGM, which gives much better security bounds than those obtained by going through a reduction to Discrete Logarithm.

4. *An analysis of SPRINT with FROST mitigation in the GGM.* As mentioned above, we can analyze the SPRINT batch randomness extraction technique in combination with re-randomized presignatures. This is based on a random beacon which may add some latency to the online signing phase (but as we will discuss, in some settings this extra latency can actually be avoided). To avoid this (possible) latency cost, we can combine SPRINT's batch randomness extraction technique with FROST's presignature-attack mitigation technique. We analyze an enhanced attack mode in the non-distributed setting that models this combination. This analysis is carried out in the GGM (modeling hash functions as random oracles).

Our results allow one to easily combine SPRINT's batch randomness extraction technique with either the re-randomized presignatures technique or the FROST technique for mitigating against presignature attacks. In particular, we can obtain a threshold Schnorr signing protocol that combines the best properties of both FROST and SPRINT:

- it is secure and robust without synchrony assumptions (like SPRINT),
- it achieves optimal resilience (i.e., tolerates up to $t < n/3$ corrupt parties, like SPRINT),
- it achieves high throughput (like SPRINT), and
- it 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).

Moreover, our results can be used to analyze the security of new protocols that are very different in design from either FROST or SPRINT. For example, [GS24] introduces new protocol techniques that give a protocol that enjoys all of the above properties, but with significantly greater throughput than SPRINT (or any other threshold Schnorr protocol, for that matter). Thanks to the modular design approach we are advocating here, the security analyses of the new protocol techniques in [GS24] are actually all quite simple, as they all go

through the same reduction to the enhanced modes of attack in the non-distributed setting that we analyze here in this paper.

As mentioned above, we carry out a security analysis of various enhanced modes of attack on (non-distributed) Schnorr in the 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 doing so is that we want to establish a general framework for proving results on various Schnorr attack modes. This framework builds on that introduced in [GS22b], in which an 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 (which could be significantly smaller than q in some settings, allowing for a more compact representation of signatures). Note that this tight security bound is not new: it was proved already in [BL19]. However, as stated above, our purpose in reproving this is to establish a general framework that will allow us to easily prove similar results for various enhanced attack modes. Moreover, our proof of this is substantially different than that in [BL19], and therefore may be of independent interest. Indeed, the proof in [BL19] is monolithic and carried out directly in the GGM+ROM. In contrast, our proof is very modular, in that we first give a reduction to a "symbolic" attack, and then give a reduction to a specific preimage property of the underlying hash function, and separately analyze this preimage property in the ROM.

We give security proofs in the GGM+ROM for all of the enhanced attack modes considered here, and 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.

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, 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.

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 $\mathcal{D}' \coloneqq \mathcal{D} + e\mathcal{G}$. 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. Additive key derivation was previously presented and analyzed in the context of threshold ECDSA signatures [GS22a], and is currently used in the threshold ECDSA signing protocol on the Internet Computer [DFI22].

1.3 The rest of the paper

Section 2 covers a number of preliminary topics. It reviews the Schnorr signature scheme, and introduces the enhanced attack modes we shall consider, along with the mitigations and variations discussed above.

In Section 3, we provide a detailed security analysis in the GGM and GGM+ROM of the Schnorr signature scheme with respect to several attack modes. To set the stage, we start with the basic attack mode, and then move on to the enhanced attack modes of re-randomized presignatures and re-randomization via hashing.

In Section 4, we provide a security analysis of enhanced modes of operation that model protocols that use batch randomness extraction.

In Appendices A and B, we give proofs of some technical lemmas.

In Appendix C, we review in some detail a general approach to construct threshold Schnorr signature schemes, and how to reduce the security of such schemes to the enhanced modes of attack defined and analyzed in this paper. This approach is very modular, allowing a system designer to "plug in" implementations of various subprotocols that satisfy appropriate security properties (such as DKG and verifiable secret sharing). These ideas are not really new, but are spread across several papers, including [Gro21, GS22a, BHK+24, GS24], sometimes in rather implicit form, and so it is hopefully useful to bring these ideas together explicitly in one place.

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 Schnorr signatures 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 Fig. 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:	Verify signature $(\mathcal{R}, z) \in E \times \mathbb{Z}_q$ on m:
$r \xleftarrow{\$} \mathbb{Z}_q,\ \mathcal{R} \leftarrow r\mathcal{G} \in E$ $h \leftarrow H(\langle \mathcal{D} \rangle \parallel \langle \mathcal{R} \rangle \parallel m) \in \mathbb{Z}_q$ $z \leftarrow r + hd$ return the signature $(\mathcal{R}, z)$	$h \leftarrow H(\langle \mathcal{D} \rangle \parallel \langle \mathcal{R} \rangle \parallel m) \in \mathbb{Z}_q$ check that $z\mathcal{G} = \mathcal{R} + h\mathcal{D}$

Figure 1: Schnorr signing and verification algorithms

NOTE: The scheme presented in Fig. 1 does not quite fully capture either BIP340 (the bitcoin version of Schnorr) or EdDSA — each have their 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. *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, [GS22a] provides fairly efficient DKG protocols that are secure and robust, with optimal resilience, in the asynchronous setting. Since these presignatures are computed in the offline phase, they may be computed in large batches to achieve better performance. The paper [GS22a] also considers such optimizations.

Biased presignatures. *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 $\bar{\mathcal{R}}_k \coloneqq u_k\mathcal{R}_k + u'_k\mathcal{G}$.*

This models a common situation in the threshold setting where we utilize a simplified 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 [GS22a] in the asynchronous communication setting. In [GS22a] 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 [GS24] for more context. We give a self-contained

discussion of all this in Appendix C.5. Note that [GS24] also gives highly optimized batched implementations of this simplified DKG.

Additive key derivation. *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 to a signing query.*

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.[1] 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.

The only nontrivial part of the proof is to simulate signaing queries. To do this, 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+19], there are attacks. Assume that the output space of H is all of $\mathbb{Z}_q$ (this is not strictly necessary, but simplifies things). Suppose the adversary is given presignatures $\mathcal{R}_1, \ldots, \mathcal{R}_K$. The adversary computes

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

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

[1] One can then reduce the security of the interactive identification scheme to the hardness of the discrete logarithm using the "forking lemma". See Chapter 19 of [BS23] for proofs of this.

the discrete logarithm problem in E. Indeed, Wagner presents an algorithm that for any constant K runs in time $O(q^{1/c})$, where $c \coloneqq 1 + \lfloor \log_2(K+1) \rfloor$. 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^*.

In fact, if the adversary can obtain $K > \log_2 q$ unused presignatures, then he can forge a signature in *polynomial time* using the "ROS attack" from [BLL+22]. While that paper does not explicitly consider the problem of attacking Schnorr with presignatures, it considers the related problem of attacking Schnorr blind signatures, and that attack is easily adapted to Schnorr with presignatures For completeness, we give that attack here.

1. The adversary first chooses three distinct messages m^*, m_0, and m_1, and computes

$$h_{k,b} \leftarrow H(\langle \mathcal{D} \rangle \parallel \langle \mathcal{R} \rangle_k \parallel m_b) \in \mathbb{Z}_q \qquad (k \in [K], b \in \{0,1\}).$$

   We may assume that $h_{k,0} \neq h_{k,1}$ for all $k \in [K]$.

2. The adversary computes the coefficients $\rho_0, \rho_1, \ldots, \rho_K \in \mathbb{Z}_q$ of the polynomial

$$\rho_0 + \sum_{k\in[K]} \rho_k \mathtt{X}_k \coloneqq \sum_{k\in[K]} 2^{k-1} \frac{\mathtt{X}_k - h_{k,0}}{h_{k,1} - h_{k,0}} \in \mathbb{Z}_q[\mathtt{X}_1, \ldots, \mathtt{X}_k].$$

   The key fact here is that the kth term of the sum on the right-hand side is 0 if we set $\mathtt{X}_k \coloneqq h_{k,0}$, and is 2^{k-1} if we set $\mathtt{X}_k \coloneqq h_{k,1}$. In particular, for every $b_1, \ldots, b_K \in \{0,1\}$, we have

$$\rho_0 + \sum_{k\in[K]} \rho_k h_{k,b_k} = \sum_{k\in[K]} 2^{k-1} b_k.$$

3. The adversary computes

$$\mathcal{R}^* \leftarrow \sum_{k\in[K]} \rho_k \mathcal{R}_k \quad \text{and} \quad h^* \leftarrow H(\langle \mathcal{D} \rangle \parallel \langle \mathcal{R}^* \rangle \parallel m^*) \in \mathbb{Z}_q,$$

   and writes $\rho_0 + h^*$ in binary as $(b_K, \ldots, b_1)_2$, so that

$$\rho_0 + h^* = \sum_{k\in[K]} 2^{k-1} b_k = \rho_0 + \sum_{k\in[K]} \rho_k h_{k,b_k},$$

   and hence

$$h^* = \sum_{k\in[K]} \rho_k h_{k,b_k}.$$

   This is where we use the assumption that $K > \log_2 q$.

4. For each $k \in [K]$, the adversary now obtains a signature $(\mathcal{R}_k, z_k)$ on m_{b_k}.

5. The adversary computes $z^* \leftarrow \sum_{k\in[K]} \rho_k z_k$ so that $(\mathcal{R}^*, z^*)$ is a valid signature on m^*.

2.4 Re-randomized presignatures

We just saw that the use of presignatures can severely (or completely) reduce the security of Schnorr signatures. One mitigation to this security weakness is to use **re-randomized presignatures**, just as in [GS22b]. The idea is that a random shift amount $\delta_k \in \mathbb{Z}_k$ 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 $\mathcal{R}'_k \coloneqq \mathcal{R}_k + \delta_k \mathcal{G}$. 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 $\mathcal{R}'_k \coloneqq u_k \mathcal{R}_k + (u'_k + \delta_k)\mathcal{G}$.

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 shift amount 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 an atomic broadcast protocol, 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. We give a self-contained discussion of all of this in Appendix C.3.

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. Just as in Section 2.3, the only nontrivial part of the proof is to simulate signing queries, which we do 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 a presignature tweak $(u_k, u'_k) \in \mathbb{Z}_q^* \times \mathbb{Z}_q$. So the effective public key is $\mathcal{D}'_k \coloneqq \mathcal{D} + e_k \mathcal{G}$, the effective presignature (used in the actual signature) is $\mathcal{R}'_k \coloneqq u_k \mathcal{R} + (u'_k + \delta_k)\mathcal{G}$ 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 3.3.

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 shift amount $\delta \in \mathbb{Z}_q$ 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. The practical impact of this is discussed in more detail in [GS24].

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 [CKM21] protocols use a hash function to derive the shift amount and use a second random group element as a part of the presignature. We abstract away the details of that protocol, and instead consider an enhanced attack mode in the non-distributed setting. Here, when the adversary makes the kth a presignature query, the challenger generates a pair of random group elements $(\mathcal{R}_k, \mathcal{S}_k)$. To sign a message m_k using this presignature pair, the effective presignature (used in the actual signature) is

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

where

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

Here, Δ is a hash function whose output space is $\mathbb{Z}_q$.

We give a full discussion of how to implement a threshold signature scheme whose security is based on this enhanced attack mode in Appendix C.4. 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 disadvantages are that (a) we have to do twice as much work to generate these presignature pairs, and (b) we cannot reap the benefits of batch re-randomization (see Section 2.4.1).

The FROST2 protocol was analyzed in [CKM21] in the random oracle model, giving a reduction to one-more discrete log (OMDL). Below in Section 3.4, we gave an analysis of the above enhanced attack mode 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 $\mathcal{R}'_k := \mathcal{R}_k + \delta_k \mathcal{G}$, where

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

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

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

and

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

The adversary sets

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

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 in subexponential time. 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^*. We also note that if $K > \log_2 q$, then we can also easily adapt the ROS attack from [BLL+22] to forge a signature in polynomial time.

3 Generic Group Model analysis

The proofs sketched in Sections 2.3 and 2.4 give reductions to breaking the interactive Schnorr identification scheme, which itself can be reduced to the DL problem via a "forking lemma" argument. This results in very "loose" reductions. An alternative approach is to carry out an analysis in the Generic Group Model, which can result is much "tighter" reductions.

In this section, we give a security analysis of Schnorr signatures in the Generic Group Model (GGM). This analysis includes both basic and enhanced attack modes. While such an analysis for the basic attack mode has already been done in [NSW09], we start with that here, so that we can develop a common framework that will be used in all of our proofs.

3.1 The EC-GGM

We shall adapt and extend many of the techniques developed in [GS22b], and in particular, we will adopt the EC-GGM (Elliptic Curve Generic Group Model) presented in that paper, because it is more faithful to Elliptic Curves by capturing point inversion, which other GGM models do not (that said, we have little doubt that all of our results hold equally well in other GGM models).

Let us review the EC-GGM. 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 nonzero 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. In the EC-GGM model, the adversary is free to "cook up" encodings of group elements that were not previously output by $\mathcal{O}_{\text{grp}}$, and supply these as inputs to `add` queries.
6. We have enhanced slightly the EC-GGM model from [GS22b]: 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]).

3.2 Analysis of basic attack

We start with an analysis in the EC-GGM of the basic security of Schnorr signatures. Let us first recall precisely the basic **chosen message attack** game, specialized to Schnorr signatures.

Attack Game 1 (CMA attack game). *In this game, an adversary $\mathfrak{A}$ interacts with a challenger as follows.*

- *First, the challenger generates a secret key d and a public key $\mathcal{D}$, and gives $\mathcal{D}$ to $\mathfrak{A}$.*
- *Next, $\mathfrak{A}$ makes a sequence of **signing queries**. In each such query, $\mathfrak{A}$ gives a message m to the challenger, who responds to $\mathfrak{A}$ with a signature $(\mathcal{R}, z)$ on m.*

*At the end of the game, $\mathfrak{A}$ outputs a message m^* and a signature $(\mathcal{R}^*, z^*)$. We say $\mathfrak{A}$ **wins the game** if $(\mathcal{R}^*, z^*)$ is a valid signature on m^*, but m^* was not submitted as a signing query. We denote by* $\text{CMA}\mathsf{adv}[\mathfrak{A}]$ *the probability that $\mathfrak{A}$ wins the game.*

Definition 1 (CMA security). *We say that the Schnorr signature scheme is **secure against chosen message attack** (or **CMA secure**) if* $\text{CMA}\mathsf{adv}[\mathfrak{A}]$ *is negligible for every efficient adversary $\mathfrak{A}$.*

3.2.1 Modeling the CMA attack game for Schnorr in the EC-GGM

We describe here how the CMA attack game (Attack Game 1) is to be interpreted in the EC-GGM. The generator $\mathcal{G}$ is encoded as $\pi(1)$ and the public key $\mathcal{D}$ is encoded as $\pi(d)$ for randomly chosen $d \in \mathbb{Z}_q$. The challenger gives these encodings of $\mathcal{G}$ and $\mathcal{D}$ to the adversary at the start of the CMA attack game.

The adversary then makes a sequence of queries to both the group and signing oracles, which are processed by the challenger. A signing oracle query on a message m is processed by the challenger as usual, generating $r \in \mathbb{Z}_q$ at random, except that it uses the group oracle to compute the encoding of $\mathcal{R} = r\mathcal{G}$ (that is, the challenger computes $\mathcal{R}$ by invoking $(\mathtt{map}, r)$). After that, the challenger computes $h \leftarrow H(\langle\mathcal{D}\rangle \parallel \langle\mathcal{R}\rangle \parallel m) \in \mathbb{Z}_q$ and $z \leftarrow r + hd$ as usual, and then gives the resulting signature $(\mathcal{R}, z)$ to the adversary.

At the end of the CMA 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 the corresponding calls to the group oracle — specifically, a call to $(\mathtt{map}, z^*)$ and call to $(\mathtt{add}, \mathcal{R}^*, \mathcal{D}, 1, h^*)$. The adversary wins the signing attack game if $(\mathcal{R}^*, z^*)$ is a valid signature on m^*, but m^* was not submitted as an input to the signing oracle.

3.2.2 Preimage attack games

In our analysis in the EC-GGM, we give a reduction to the intractability of various preimage attacks on the hash function H. These are stated a bit more generally than our immediate needs required.

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 $H(x) = 0$.

Random oracle analysis. Besides giving reductions to the intractability of these preimage attacks, we will also want to give explicit security bounds in the setting where H is modeled as a random oracle. Assume that the output space of H is a subset of $\mathbb{Z}_q$ of size M. 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 $O(N^2_{\mathrm{ch}}/q + N_{\mathrm{h}}/M)$. The term $O(N^2_{\mathrm{ch}}/q)$ 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 $O(N_{\mathrm{h}}/M)$.

3.2.3 Security theorem for the basic CMA attack

We now state and prove our main security theorem for the CMA-security of Schnorr signatures in the EC-GGM. This theorem reduces the security of Schnorr in the EC-GGM to the intractability of the preimage attacks on H presented in Section 3.2.2. It also gives security bounds in the setting where we additionally model H as a random oracle whose output space is a subset of $\mathbb{Z}_q$ of size M. We assume that the adversary $\mathfrak{A}$ makes

- at most N_{sig} signing queries, and

- at most N_{grp} queries to the group oracle.

We assume here that N_{grp} includes the group oracle queries made by the challenger 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 during the attack; moreover, in the random oracle setting, N also includes the number of random oracle queries.

Theorem 1 (CMA security of Schnorr in EC-GGM). *If an adversary $\mathfrak{A}$ has an advantage $\aleph$ in the CMA attack game in the EC-GGM, then*

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

Here, $\aleph_X$ is the advantage of an adversary $\mathfrak{A}_X$ in winning Preimage Attack X, for $X \in \{\text{I}, \text{II}, \text{III}\}$. Each $\mathfrak{A}_X$ has roughly the same running time as $\mathfrak{A}$. Moreover, $\mathfrak{A}_{\text{I}}$ makes at most N_{sig} challenge queries and $\mathfrak{A}_{\text{II}}$ makes at most N_{grp} challenge queries.

In addition, if we model H as a random oracle, then we have

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

To prove the theorem, we use the same general framework developed in [GS22b], where we first replace the "real" attack game in the EC-GGM by a "lazy simulation", and then replace that by a "symbolic simulation". The move from real attack to symbolic simulation is usually straightforward and fairly mechanical, and allows us to then focus on the "meat" of the proof in a more intuitive fashion.

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 as we go along. 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. Fig. 2 gives the details of **Lazy-Sim**. This is essentially the same as the lazy simulator in Fig. 2 in [GS22b], 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 $\mathit{Domain}(\pi)$ will now consist of polynomials of the form $a + b\mathtt{D}$, where $a, b \in \mathbb{Z}_q$ and $\mathtt{D}$ is a variable (or indeterminant). Here, $\mathtt{D}$ symbolically represents the value of d. Note that π will otherwise still satisfy all of the requirements of an encoding function. Fig. 3 gives the details of **Symbolic-Sym**. This is essentially the same as the symbolic simulator in Fig. 3 in [GS22b], 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 $r \leftarrow z - h\mathtt{D}$, 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 $(\mathcal{R}, z)$.

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 $i \notin \mathit{Domain}(\pi)$:
      i. $\mathcal{P} \xleftarrow{\$} E$;
         while $\mathcal{P} \in \mathit{Range}(\pi)$ do: $\mathcal{P} \xleftarrow{\$} E$
      ii. add $(-i, -\mathcal{P})$ and $(i, \mathcal{P})$ to π
   (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. $i \xleftarrow{\$} \mathbb{Z}_q$;
         while $i \in \mathit{Domain}(\pi)$ do: $i \xleftarrow{\$} \mathbb{Z}_q$
      ii. add $(-i, -\mathcal{P}_j)$ and $(i, \mathcal{P}_j)$ to π
   (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 following lemma is fairly straightforward, and may be proved along the same lines as Lemma 1 in [GS22b].

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

Proof. See Appendix A. □

By virtue of Lemma 1, it suffices to prove Theorem 1 in the Symbolic-Sim game.

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

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 $h \coloneqq H(\langle \mathcal{D} \rangle \parallel \langle \mathcal{R} \rangle \parallel m)$. 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 Preimage Attack I. In particular, we can easily convert $\mathfrak{A}$ into an adversary $\mathfrak{A}_{\mathrm{I}}$ that attack as H as in Preimage Attack I, makes at most N_{sig} challenge queries, and wins that game with probability identical to the probability that $\mathfrak{A}$ succeeds in making a Type I forgery. (Note that in this reduction to Preimage Attack I, we have $\mathcal{D}'_k = \mathcal{D}^* = \mathcal{D}$ — the extra flexibility is needed only to deal with an analysis of a variation with additive key derivation, discussed below.)

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

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 $\texttt{abort}$
      ii. add $(-i, -\mathcal{P})$ and $(i, \mathcal{P})$ to π
   (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 $\texttt{abort}$
      ii. add $(-i, -\mathcal{P}_j)$ and $(i, \mathcal{P}_j)$ to π
   (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 $\texttt{abort}$
   (e) add $(-r, -\mathcal{R})$ and $(r, \mathcal{R})$ to π
   (f) return $(\mathcal{R}, z)$

Figure 3: Symbolic-Sim

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, or by the challenger in the initialization step). Note that the assumption $h^* \neq 0$ is used here to rule out the possibility of the encoding of $\mathcal{R}^*$ being "cooked up" directly by the adversary.

In this case, the adversary must essentially win Preimage Attack II. In particular, we can easily convert $\mathfrak{A}$ into an adversary $\mathfrak{A}_{\text{II}}$ that attack as H as in Preimage Attack II, makes at most N_{grp} challenge queries, and wins that game with probability identical to the probability that $\mathfrak{A}$ succeeds in making a Type II forgery. (Note that in this reduction to Preimage Attack II, we have $\mathcal{D}^* = \mathcal{D}$ — the extra flexibility is needed only to deal with an analysis of a variation with additive key derivation, discussed below.)

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

In this case, the adversary must find a preimage of zero under H, that is, it must essentially win Preimage Attack III. Note this case does not arise in the analysis of [NSW09], because they do not allow the adversary to directly "cook up" the encoding of $\mathcal{R}^*$, which is allowed in the EC-GGM.

The security bounds (1) and (2) are readily verified. That completes the proof of Theorem 1.

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], although using a completely different proof.

3.3 Analysis of attack with re-randomized presignatures

We next give an analysis in the EC-GGM of the security of Schnorr signatures with re-randomized presignatures. Let us first state precisely the attack game that characterizes this security property.

Attack Game 2 (CMA-RRP security). *In this game, an adversary $\mathfrak{A}$ interacts with a challenger as follows.*

- *First, the challenger generates a secret key d and a public key $\mathcal{D}$, initializes a set $\mathcal{K}$ of indices to the empty set, and gives $\mathcal{D}$ to $\mathfrak{A}$.*
- *Next, $\mathfrak{A}$ makes a sequence of presignature and signing queries:*
  - *in the kth **presignature query**, for $k = 1, 2, \ldots$, the challenger does the following: computes*
    - *computes $r_k \xleftarrow{\$} \mathbb{Z}_q$ and $\mathcal{R}_k \leftarrow r_k\mathcal{G}$,*
    - *adds the index k to $\mathcal{K}$, and*
    - *sends the presignature $\mathcal{R}_k$ to $\mathfrak{A}$;*
  - *in a **signing query**, the adversary specifies an index k and a message m_k; if $k \in \mathcal{K}$, the challenger does the following:*
    - *computes $\delta_k \xleftarrow{\$} \mathbb{Z}_q$ and $\mathcal{R}'_k \leftarrow \mathcal{R}_k + \delta_k\mathcal{G}$,*
    - *computes the signature $(\mathcal{R}'_k, z_k)$ on m_k using the effective presignature $\mathcal{R}'_k$,*
    - *removes k from $\mathcal{K}$, and*
    - *sends δ_k and $(\mathcal{R}'_k, z_k)$ to $\mathfrak{A}$.*

*At the end of the game, $\mathfrak{A}$ outputs a message m^* and a signature $(\mathcal{R}^*, z^*)$. We say $\mathfrak{A}$ **wins the game** if $(\mathcal{R}^*, z^*)$ is a valid signature on m^*, but m^* was not submitted as a signing query. We denote by* CMA-RRP$\mathsf{adv}[\mathfrak{A}]$ *the probability that $\mathfrak{A}$ wins the game.*

Definition 2 (CMA-RRP security). *We say that the Schnorr signature scheme is **CMA-RRP secure** if* CMA-RRP$\mathsf{adv}[\mathfrak{A}]$ *is negligible for every efficient adversary $\mathfrak{A}$.*

The above definition does not cover biased presignatures or additive key derivation.

- We add **biased presignatures** to the CMA-RRP attack game as follows: in each signing query, the adversary $\mathfrak{A}$ additionally supplies a bias $(u_k, u'_k) \in \mathbb{Z}_q^* \times \mathbb{Z}_q$, and then challenger computes the biased presignature $\bar{\mathcal{R}}_k \coloneqq u_k\mathcal{R}_k + u'_k\mathcal{G}$, and uses this in place of the presignature $\mathcal{R}_k$ (so that the effective presignature is $\mathcal{R}'_k = \bar{\mathcal{R}}_k + \delta_k\mathcal{G}$).
- We add **additive key derivation** to the CMA-RRP attack game as follows: in each signing query, the adversary $\mathfrak{A}$ additionally supplies a tweak $e_k \in \mathbb{Z}_q$, and the challenger computes the effective public key as $\mathcal{D}'_k \coloneqq \mathcal{D} + e_k\mathcal{G}$ and the resulting signature is relative to this effective public key. In addition, at the end of the game, the adversary must also supply a tweak $e^* \in \mathbb{Z}_q$, and the adversary wins the game if $(\mathcal{R}^*, z^*)$ is valid signature on m^* relative to the effective public key $\mathcal{D}^* \coloneqq \mathcal{D} + e^*\mathcal{G}$, and the pair (m^*, e^*) was not submitted to any signing query.

We can also add both biased presignatures and additive key derivation to the CMA-RRP attack game.

In Section 2.4.1, we mentioned the possibility of batch re-randomization. To adapt the CMA-RRP attack game and all of its variants to this setting, one would allow the adversary to submit any number of signing queries in a single batch. The difference is that the challenger generates a single random shift amount $\delta \in \mathbb{Z}_q$ that is used for each signing request in the batch. The response to the adversary includes δ and the signatures for each signing request in the batch.

3.3.1 Another preimage attack

We need to add another attack to our list of preimage attack games in Section 3.2.2.

Preimage Attack II′ on H.

- The challenger gives a collection $\{\mathcal{R}_i^*\}_{i=1}^{N_{\mathrm{ch}}}$ 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 $\mathcal{I}_k \subseteq \{1, \ldots, N_{\mathrm{ch}}\}$ 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, $(m^*, \mathcal{D}^*)$, and $\epsilon \in \{\pm 1\}$ such that

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

Random oracle analysis. Assume that the output space of H is a subset of $\mathbb{Z}_q$ of size M. 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

$$H(\langle \mathcal{D}^* \rangle \parallel \langle \epsilon \mathcal{R}_i^* \rangle \parallel m^*), \tag{3}$$

there is a unique index i 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 (3) to lead to a win. Here, k is the index of the completion query which included i in $\mathcal{I}_k$. Moreover, assuming that $\mathcal{R}'_k \neq \pm \mathcal{R}_i^*$ 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 (3) leads to a win with probability at most $1/M$. From this, we see that the adversary wins the attack with probability at most

$$O((N_{\mathrm{ch}} + N_{\mathrm{cmp}} + N_{\mathrm{h}})^2/q + N_{\mathrm{h}}/M). \tag{4}$$

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) $k \leftarrow 0$; $\mathcal{K} \leftarrow \emptyset$
   (f) return $(\mathcal{G}, \mathcal{D})$
2. To process a group oracle query $(\texttt{map}, i)$:
   (a) if $i \notin \mathit{Domain}(\pi)$:
      i. $\mathcal{P} \xleftarrow{\$} E$; while $\mathcal{P} \in \mathit{Range}(\pi)$ do: $\mathcal{P} \xleftarrow{\$} E$
      ii. add $(-i, -\mathcal{P})$ and $(i, \mathcal{P})$ to π
   (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. $i \xleftarrow{\$} \mathbb{Z}_q$; while $i \in \mathit{Domain}(\pi)$ do: $i \xleftarrow{\$} \mathbb{Z}_q$
      ii. add $(-i, -\mathcal{P}_j)$ and $(i, \mathcal{P}_j)$ to π
   (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) $k \leftarrow k + 1$; $\mathcal{K} \leftarrow \mathcal{K} \cup \{k\}$
   (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 $k \in \mathcal{K}$:
   (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) $\mathcal{K} \leftarrow \mathcal{K} \setminus \{k\}$; return $(\mathcal{R}'_k, z_k, \delta_k)$

Figure 4: Lazy-Sim

3.3.2 Security theorem for the CMA-RRP attack game

We now state and prove our main security theorem for the CMA-RRP security of Schnorr signatures in the EC-GGM. The bounds M, N, N_{sig}, and N_{grp} are defined as in Section 3.2.3, except that N_{grp} now also includes the group oracle queries made by the challenger in processing presignature queries. In addition, we define L to be the maximum number of unused presignatures that are extant at any time (this is the maximum size that the set $\mathcal{K}$ takes during Attack Game 2).

Theorem 2 (CMA-RRP security of Schnorr in EC-GGM). *If an adversary $\mathfrak{A}$ has an advantage $\aleph$ in the CMA-RRP attack game in the EC-GGM, then*

$$\aleph = O(N^2/q + \aleph_{\text{I}} + \aleph_{\text{II}} + \aleph_{\text{II}'} + \aleph_{\text{III}}). \tag{5}$$

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

In addition, if we model H as a random oracle, then we have

$$\aleph = O(N^2/q + N/M). \tag{6}$$

As in the proof of Theorem 1, to prove this theorem, we first replace the "real" attack game in the EC-GGM by a "lazy simulation", and then replace that by a "symbolic simulation". Fig. 4 gives the detailed logic of **Lazy-Sim**, which is a lazy simulation of the forgery attack game. Fig. 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 [GS22b], in which each presignature $\mathcal{R}_k$ corresponds to

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) $k \leftarrow 0$; $\mathcal{K} \leftarrow \emptyset$
   (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 π
   (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 π
   (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) $k \leftarrow k + 1$; $\mathcal{K} \leftarrow \mathcal{K} \cup \{k\}$
   (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 $k \in \mathcal{K}$:
   (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 π
   (e) $h_k \leftarrow H(\langle \mathcal{D} \rangle \parallel \langle \mathcal{R}'_k \rangle \parallel m_k) \in \mathbb{Z}_q$
   (f) $z_k \xleftarrow{\$} \mathbb{Z}_q$;
   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) $\mathcal{K} \leftarrow \mathcal{K} \setminus \{k\}$; return $(\mathcal{R}'_k, z_k, \delta_k)$

Figure 5: Symbolic-Sim

a variable $\texttt{R}_k$, meaning that $\pi(\texttt{R}_k) = \mathcal{R}_k$. When such a presignature is 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 $\pi(\texttt{R}_k + \delta_k) = \mathcal{R}'_k$, and then substitute

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

throughout $\mathit{Domain}(\pi)$. This same "substitution strategy" for dealing with presignatures in the GGM was used extensively in [GS22b], and works equally well here. The following lemma is fairly straightforward, and may be proved along the same lines as Lemma 2 in [GS22b].

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

Proof. See Appendix B. □

By virtue of this lemma, it suffices to analyze the forgery attack in the symbolic simulation. We break the analysis into the same three cases as in the proof of Theorem 1, depending on the type of the forged signature $(\mathcal{R}^*, z^*)$ on a message m^*.

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 3.2.3.

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

Since $h^* \neq 0$, the group encoding $\mathcal{R}^*$ was generated at random as the result of a group oracle query (made by the adversary, or by the challenger in the initialization step or

in processing a presignature query). Suppose that initially

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

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 3.2.3.

Otherwise, in order for the forgery to be valid, the $\mathtt{R}_k$ variables need to be eliminated by substitution, so as to end up with

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

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

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

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

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

Observe that at this point in time, h_ℓ is the output of a hash, one of whose inputs is the group element $\mathcal{R}'_\ell$, which was generated at random after b' and c_ℓ were fixed, and so, to succeed, the adversary must solve a certain type of preimage problem — essentially, the adversary must win Preimage Attack II′ (see Section 3.3.1). In this preimage attack, the random challenge values $\mathcal{R}^*_i$ correspond to outputs from the group oracle in the symbolic simulation of the signing attack, while the random values $\mathcal{R}'_k$ produced by the completion queries correspond to the outputs of the signing oracle. The kth completion query in the preimage 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. (Note that in this reduction to Preimage Attack II′, we have $\mathcal{D}'_k = \mathcal{D}^* = \mathcal{D}$ — the extra flexibility is needed only to deal with an analysis of a variation with additive key derivation, discussed 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 3.2.3.

The security bounds (5) and (6) are readily verified. For the latter, the bound (4) is helpful. That completes the proof of Theorem 2.

3.3.3 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 $\mathcal{D}'_k := \mathcal{D} + e_k\mathcal{G}$, 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 Theorem 2 also hold here.

3.4 Re-randomizing presignatures via hashing

We next give an analysis in the EC-GGM of the security of Schnorr signatures with presignatures that are re-randomized via a hash function Δ whose output space is $\mathbb{Z}_q$. Let us first state precisely the attack game that characterizes this security property.

Attack Game 3 (CMA-HRRP security). *In this game, an adversary $\mathfrak{A}$ interacts with a challenger as follows.*

- *First, the challenger generates a secret key d and a public key $\mathcal{D}$, initializes a set $\mathcal{K}$ of indices to the empty set, and gives $\mathcal{D}$ to $\mathfrak{A}$.*
- *Next, $\mathfrak{A}$ makes a sequence of presignature and signing queries:*
  - *in the kth **presignature query**, for $k = 1, 2, \ldots$, the challenger does the following:*
    - *computes $r_k, s_k \xleftarrow{\$} \mathbb{Z}_q$, $\mathcal{R}_k \leftarrow r_k\mathcal{G}$, and $\mathcal{S}_k \leftarrow s_k\mathcal{G}$*
    - *adds the index k to $\mathcal{K}$, and*
    - *sends the presignature pair $(\mathcal{R}_k, \mathcal{S}_k)$ to $\mathfrak{A}$;*
  - *in a **signing query**, the adversary specifies an index k and a message m_k; if $k \in \mathcal{K}$, the challenger does the following:*
    - *computes*
      $$\delta_k \leftarrow \Delta(\langle\mathcal{D}\rangle \parallel \langle\mathcal{R}_k\rangle \parallel \langle\mathcal{S}_k\rangle \parallel m_k)$$
      and $\mathcal{R}'_k \leftarrow \mathcal{R}_k + \delta_k\mathcal{S}_k$,
    - *computes the signature $(\mathcal{R}'_k, z_k)$ on m_k using the effective presignature $\mathcal{R}'_k$,*
    - *removes k from $\mathcal{K}$, and*
    - *sends $(\mathcal{R}'_k, z_k)$ to $\mathfrak{A}$.*

*At the end of the game, $\mathfrak{A}$ outputs a message m^* and a signature $(\mathcal{R}^*, z^*)$. We say $\mathfrak{A}$ **wins the game** if $(\mathcal{R}^*, z^*)$ is a valid signature on m^*, but m^* was not submitted as a signing query. We denote by* CMA-HRRP$\mathsf{adv}[\mathfrak{A}]$ *the probability that $\mathfrak{A}$ wins the game.*

Definition 3 (CMA-HRRP security). *We say that the Schnorr signature scheme is **CMA-HRRP secure** if* CMA-HRRP$\mathsf{adv}[\mathfrak{A}]$ *is negligible for every efficient adversary $\mathfrak{A}$.*

We can add **additive key derivation** to the CMA-HRRP attack game in precisely the same as in the CMA-RRP attack game, except that in processing a signing query, the derived public key $\mathcal{D}'_k$ should also be input to the hash Δ, in place of $\mathcal{D}$.

We shall not immediately consider biased presignatures in this setting. Rather, we will deal with biased presignatures in this setting as a special case of batch randomness extraction in Section 4.4, as we will have to slightly modify the enhanced attack mode to make the security proof go through.[2]

We will not consider batch re-randomization at all in this setting.

3.4.1 Security theorem for the CMA-HRRP attack game

We now state and prove our main security theorem for the CMA-HRRP security of Schnorr signatures in the EC-GGM. Here, we will model Δ as a random oracle with output space $\mathbb{Z}_q$. Since we are already working in the random oracle model, we will also simply model as H as random oracle. We assume that the output space of H is a subset of $\mathbb{Z}_q$ of size M, and that the total number of queries made by $\mathfrak{A}$, including signing, group oracle, and random oracle queries, is at most N.

Theorem 3 (CMA-HRRP security of Schnorr in EC-GGM). *If an adversary $\mathfrak{A}$ has an advantage $\aleph$ in the CMA-HRRP attack game in the EC-GGM, then we have*

$$\aleph = O(N^2/q + N/M). \tag{7}$$

To prove this theorem, we first observe that when a signing query on a message m_k is made that uses the presignature pair $(\mathcal{R}_k, \mathcal{S}_k)$, a preliminary computation

$$\begin{aligned} \delta_k &\leftarrow \Delta(\langle \mathcal{D} \rangle \parallel \langle \mathcal{R}_k \rangle \parallel \langle \mathcal{S}_k \rangle \parallel m_k), \\ \mathcal{R}'_k &\leftarrow \mathcal{R}_k + \delta_k \mathcal{S}_k, \\ h_k &\leftarrow H(\langle \mathcal{D} \rangle \parallel \langle \mathcal{R}'_k \rangle \parallel m_k) \end{aligned} \tag{8}$$

is made. WLOG, we can assume that the adversary has already computed these values himself before making the signing query. This means that the signing oracle does not make any group or random oracle queries. To simplify the analysis, we make one more assumption about the adversary. Namely, we ensure that whenever the adversary makes a random oracle query of the form

$$\delta \leftarrow \Delta(\langle \mathcal{D}' \rangle \parallel \langle \mathcal{R} \rangle \parallel \langle \mathcal{S} \rangle \parallel m), \tag{9}$$

for some $\mathcal{D}'$, $\mathcal{R}$, $\mathcal{S}$, and m, it immediately makes a "special `add` query"

$$(\texttt{add}, \mathcal{R}, \mathcal{S}, 1, \delta) \tag{10}$$

to the group oracle to obtain the encoding of the group element $\mathcal{R}' \coloneqq \mathcal{R} + \delta\mathcal{S}$, and then it immediately makes the random oracle query

$$h \leftarrow H(\langle \mathcal{D}' \rangle \parallel \langle \mathcal{R}' \rangle \parallel m). \tag{11}$$

With this simplifying assumption, we have the following **invariant**:

[2] In versions of this paper on `https://eprint.iacr.org/` dated 2024-04-08 and earlier, a brief proof sketch was given for biased presignatures in this setting without modification. However, that proof sketch was rather messy and incomplete, and it is not clear that it can be made to work.

At any point in during the attack, for every query of the random oracle Δ of the form (9), the corresponding group element $\mathcal{R}'$ and hash value h have already been computed, with at most one exception, and this exception will be immediately eliminated by the steps (10) and (11).

Again, to prove this theorem, we first replace the "real" attack game in the EC-GGM by a "lazy simulation", and then replace that by a "symbolic simulation". The move to the lazy simulation should by now be routine. As for the symbolic simulation, the initialization step and group oracle queries are exactly as in Fig. 5. Presignature queries are handled the same as in Fig. 5, except that in addition to invoking $(\texttt{map}, \mathtt{R}_k)$ to get $\mathcal{R}_k$, the challenger also invokes $(\texttt{map}, \mathtt{S}_k)$ to get $\mathcal{S}_k$, where $\mathtt{S}_k$ is a new variable. Thus, we have $\pi(\mathtt{R}_k) = \mathcal{R}_k$ and $\pi(\mathtt{S}_k) = \mathcal{S}_k$. Finally, when a request is made to sign m_k using the presignature pair $(\mathcal{R}_k, \mathcal{S}_k)$, the challenger computes the values (8) (which, by assumption, have already been computed by the adversary), and generates $z_k \in \mathbb{Z}_q$ at random, so that the resulting signature is $(\mathcal{R}'_k, z_k)$. Before returning this signature to the adversary, the challenger substitutes

$$\mathtt{R}_k \mapsto z_k - \delta_k \mathtt{S}_k - h_k \mathtt{D} \tag{12}$$

throughout $\mathit{Domain}(\pi)$. The symbolic simulation will "fail" if any of these substitutions cause $\mathit{Domain}(\pi)$ to "collapse" (i.e., if two distinct elements of $\mathit{Domain}(\pi)$ 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^*. Since the signing oracle does not generate any new group elements, 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 3.2.2, but where H is modeled as a random oracle).

Suppose that initially

$$\pi^{-1}(\mathcal{R}^*) = a + b\mathtt{D} + \sum_k (c_k \mathtt{R}_k + d_k \mathtt{S}_k), \tag{13}$$

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 $\mathtt{R}_k, \mathtt{S}_k$ 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

$$\pi^{-1}(\mathcal{R}^*) = \underbrace{\{a + \sum_k c_k z_k\}}_{=z^*} + \underbrace{\{b - \sum_k c_k h_k\}}_{=-h^*} \mathtt{D} + \sum_k \underbrace{\{d_k - c_k \delta_k\}}_{=0} \mathtt{S}_k. \tag{14}$$

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 the adversary generates $\mathcal{R}^*$, we can inspect the queries to the random oracle Δ to find for each index k an input

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

to Δ that yields the output d_k/c_k. This inspection process could fail, in the sense that for some k it does not correctly identify the unique signing request that will eliminate $(\mathtt{R}_k, \mathtt{S}_k)$, but this happens with negligible probability (see below). Thus, at the time we generate $\mathcal{R}^*$, the signing queries that define the h_k's must already be preordained — *this is the crux of the proof (and is the reason that a generalized birthday or ROS attack as in Section 2.3 cannot be deployed here).*

More precisely, either

(a) all of the h_k's have already been computed, or

(b) some h_k has not yet been computed.

In case (a), to succeed, the adversary must essentially win Preimage Attack II on H (as in Section 3.2.2, but where H is modeled as a random oracle). Now consider case (b). By the invariant discussed above (right after (11)), $\mathcal{R}^*$ must have been generated as a result of a "special `add` query". This means that $\mathcal{R}^*$ must be equal to $\pm\mathcal{R}'_k$, where $\mathcal{R}'_k = \mathcal{R}_k + \delta_k \mathcal{S}_k$ was created by this special `add` query, and must eventually be used as the effective presignature in the signing query identified by the inspection process that eliminates $(\mathtt{R}_k, \mathtt{S}_k)$. Therefore, to succeed, the adversary must essentially win Preimage Attack I on H (again, as in Section 3.2.2, but where H is modeled as a random oracle).

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 (13) 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 $O(N_{\mathrm{grp}} N_{\mathrm{h}}/q)$.

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 $O(N^2/q + N/M)$. That completes the proof of Theorem 3.

3.4.2 Variations

If we use additive key derivation, deriving $\mathcal{D}'_k := \mathcal{D} + e_k \mathcal{G}$, then we also need to include $\mathcal{D}'_k$ as input to Δ, in place of $\mathcal{D}$. The substitution (12) then becomes

$$\mathtt{R}_k \mapsto z_k - h_k e_k - \delta_k \mathtt{S}_k - h_k \mathtt{D} \tag{15}$$

and the constant term in (14) is adjusted accordingly. The main argument does not change too much, and we leave the details to the reader to verify that the same security bounds as in Theorem 3 hold.

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. See Appendix C.5 for more details. 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+24], and further explored in [GS24].

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 [GS24] — but see Appendix C.6 here for a more complete and self-contained discussion. This generalized biasing can be modeled as an enhanced attack mode, which we define next for the specific setting of re-randomized presignatures.

4.1 Re-randomized presignatures

We begin by defining an enhanced attack mode that models batch randomness extraction with re-randomized presignatures. The attack game generalizes Attack Game 2.

Let $P \leq Q$ be fixed parameters. In this attack game, the adversary issues *presignature queries* to the challenger as usual. However, to process the kth such query, the challenger generates a batch of "initial" presignatures $\mathcal{R}_{k,1}, \ldots, \mathcal{R}_{k,Q}$ at random and these are immediately given to the adversary. The adversary may also issue *biasing queries*. Each such query specifies the index k of a batch of initial presignatures, along with 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 "biased" presignatures $\bar{\mathcal{R}}_{k,1}, \ldots, \bar{\mathcal{R}}_{k,P}$ as follows:

$$\begin{pmatrix} \bar{\mathcal{R}}_{k,1} \\ \vdots \\ \bar{\mathcal{R}}_{k,P} \end{pmatrix} = U_k \begin{pmatrix} \mathcal{R}_{k,1} \\ \vdots \\ \mathcal{R}_{k,Q} \end{pmatrix} + \mathbf{u}'_k \mathcal{G}. \tag{16}$$

Note that a biasing query may be applied to the kth batch of initial presignatures only after that batch has already been generated by a presignature query, and only once.

A *signing query* now specifies a pair of indices (k, i) and a message $m_{k,i}$. Note that k must be the index of a batch of biased presignatures, i must be in the range $1, \ldots, P$, and (k, i) must not have already been used in a previous signing query. The challenger processes this request using the biased presignature $\bar{\mathcal{R}}_{k,i}$ by first generating a shift amount $\delta_{k,i} \in \mathbb{Z}_q$ at random, and then signing the message using the effective presignature

$$\mathcal{R}'_{k,i} \coloneqq \bar{\mathcal{R}}_{k,i} + \delta_{k,i} \mathcal{G}.$$

The challenger gives the shift amount $\delta_{k,i}$ to the adversary along with the resulting signature $(\mathcal{R}'_{k,i}, z_{k,i})$. This enhanced mode of attack corresponds to a threshold implementation in which δ_k is generated by a random beacon — see Appendix C.3 for details.

Note that in the attack mode for biased presignatures Section 2.2, we simply modified the signing query to include a bias. Here, we need to introduce a separate "biasing query", since biasing is applied to initial presignatures in batches, while signing queries are applied to individual biased presignatures.

4.2 Random oracle analysis

Generalizing the analysis in Section 2.4, we give here an efficient reduction from the security of this enhanced attack mode 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

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

and we have

$$\bar{\mathcal{R}}_{k,i} = \sum_{j\in[Q]} \sigma^{(j)}_{k,i}\mathcal{R}_{k,j} + \mu_{k,i}\mathcal{G} \tag{17}$$

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

$$\begin{aligned}\bar{\mathcal{R}}_{k,i} &= \sum_{j\in[Q]} \sigma^{(j)}_{k,i}\mathcal{R}_{k,j} + \mu_{k,i}\mathcal{G} \\ &= \{\mu_{k,i} + \sum_{j\in[Q]} \sigma^{(j)}_{k,i}\zeta_{k,j}\}\mathcal{G} - \{\sum_{j\in[Q]} \sigma^{(j)}_{k,i}\eta_{k,j}\}\mathcal{D}.\end{aligned}$$

This implies that the re-randomized presignature is

$$\mathcal{R}'_{k,i} = \bar{\mathcal{R}}_{k,i} + \delta_{k,i}\mathcal{G} = \{\delta_{k,i} + \mu_{k,i} + \sum_{j\in[Q]} \sigma^{(j)}_{k,i}\zeta_{k,j}\}\mathcal{G} - \{\sum_{j\in[Q]} \sigma^{(j)}_{k,i}\eta_{k,j}\}\mathcal{D}$$

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

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

and, additionally, program the random oracle so that

$$H(\langle\mathcal{D}\rangle \parallel \langle\mathcal{R}'_{k,i}\rangle \parallel m_{k,i}) \coloneqq h_{k,i},$$

where

$$h_{k,i} := \sum_{j \in [Q]} \sigma_{k,i}^{(j)} \eta_{k,j}.$$

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 4.3).

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+24]. 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 biased presignatures $\{\bar{\mathcal{R}}_{k,i}\}_i$, so that all of the $\delta_{k,i}$'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+24] 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+24] 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. Indeed, one of goals of presignatures is to reduce the latency in processing signing requests by precomputing 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. Moreover, as discussed in [GS24], another goal of presignatures is to simply increase overall throughput by exploiting the fact that batching itself can significantly reduce the amortized cost of producing presignatures. 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 safely 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.

4.3 Generic group model analysis

In Section 4.2 we analyzed the enhanced attack mode modeling batch randomness extraction 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, which generalizes the analysis in Section 3.3.

The attack game is as defined in Section 4.1. We shall prove a result analogous to Theorem 2 — see Section 4.3.1 for a summary.

The analysis follows the same outline as in Section 3.3, where one first moves from the real attack game to a symbolic simulation. Analogous to (17), we have

$$\bar{\mathtt{R}}_{k,i} = \sum_{j\in[Q]} \sigma_{k,i}^{(j)} \mathtt{R}_{k,j} + \mu_{k,i}, \tag{18}$$

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 (18) 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 constraint 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 biased presignature $\bar{\mathcal{R}}_{k,i}$, which is re-randomized to obtain the effective presignature $\mathcal{R}'_{k,i} \coloneqq \bar{\mathcal{R}}_{k,i} + \delta_{k,i}\mathcal{G}$. Here, $\delta_{k,i}$ is generated at random by only after the signing request has been made. Here, the symbolic simulation chooses $z_{k,i}, \delta_{k,i} \in \mathbb{Z}_q$ and $\mathcal{R}'_{k,i} \in E$ at random, programs π so that $\pi(\bar{\mathtt{R}}_{k,i} + \delta_{k,i}) = \mathcal{R}'_{k,i}$, 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 the proof of Theorem 2.

4.3.1 Summary

The concrete security bounds in Theorem 2 also hold in this setting, except the running times of the various adversaries in the reductions may be somewhat higher.[3] However, if we are only interested in security bounds in the GGM+ROM, then these running times do not really matter. The argument is also easily adapted to handle additive key derivation, as well as 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. The same security bounds hold for all of these variations.

4.4 Re-randomizing presignatures via hashing

In this section, we present an enhanced attack mode that corresponds to a protocol that combines the technique of re-randomization via hashing (as in Section 3.4) with batch randomness extraction. This attack game generalizes Attack Game 3, but also adds one new element. We analyze this attack mode in the GGM plus ROM.

As in Section 4.1, we let $P \leq Q$ be fixed parameters. In this attack game, we assume that in response to *presignature query*, the challenger generates a pair of batches of initial presignatures: a batch $\mathcal{R}_{k,1}, \ldots, \mathcal{R}_{k,Q}$ and a batch $\mathcal{S}_{k,1}, \ldots, \mathcal{S}_{k,Q}$. After this, the adversary may issue a corresponding *biasing query*, in which the adversary specifies biases $(U_k, \mathbf{u}'_k)$ and $(V_k, \mathbf{v}'_k)$, which (analogous to (16)) defines a batch $\bar{\mathcal{R}}_{k,1}, \ldots, \bar{\mathcal{R}}_{k,P}$ of biased presignatures (using the first bias) and a batch $\bar{\mathcal{S}}_{k,1}, \ldots, \bar{\mathcal{S}}_{k,P}$ of biased presignatures (using the second bias). In addition:

> *the challenger generates a random nonce ρ_k from some set of size at least q which is given to the adversary as the result of the biasing query.*

A *signing query* specifies a pair of indices (k, i) and a message $m_{k,i}$, and the challenger signs $m_{k,i}$ using the effective presignature

$$\mathcal{R}'_{k,i} \coloneqq \bar{\mathcal{R}}_{k,i} + \delta_{k,i}\bar{\mathcal{S}}_{k,i},$$

where

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

NOTES:

1. In processing a biasing query, the challenger generates a random nonce ρ_k, which is then later included as an input to the hash Δ when generating a signature using one of the resulting pairs of biased presignatures. This nonce mechanism is essential to our proof of security. In the threshold setting, such a nonce is generated using a random beacon, but it is only needed in the offline preprocessing phase, so this is not a practical concern. See Appendix C.6 for more details.

2. In Section 3.4, our analysis did not cover biased presignatures. The presentation here fills that gap, simply by setting $P = Q = 1$. However, unlike in Section 3.4, our analysis here requires the use of these random nonces. If we do not wish to use batch randomness extraction, but

[3] 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.

still use biased presignatures that arise as in Appendix C.5, we can simply generate such biased presignatures in batches, using one nonce per batch, and model it using this enhanced attack mode with $P = Q =$ the batch size.

3. The FROST and FROST2 protocols do not rely on random nonces as we do here. Our need for these nonces is perhaps an artifact of our modular approach. For example, in FROST2, the inputs to the hash function Δ are tightly coupled to the particular mechanism used to generate presignatures. We have completely decoupled these two things, which allows for more flexibility and efficiency in the generation of presignatures, at the (very minor) cost of generating these random nonces in the offline preprocessing phase.

We now proceed to analyze this mode of attack in the GGM+ROM. As in Section 3.4, both Δ and H are modeled as a random oracles. We shall prove a result analogous to Theorem 3 — see Section 4.4.1 for a summary.

Analogous to (18), we have

$$\bar{\mathtt{S}}_{k,i} = \sum_{j \in [Q]} \tau_{k,i}^{(j)} \mathtt{S}_{k,j} + \nu_{k,i}, \tag{19}$$

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 (18), 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 4.3, 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 3.4, when a signing query on a message $m_{k,i}$ is made that uses the biased presignature pair $(\bar{\mathcal{R}}_{k,i}, \bar{\mathcal{S}}_{k,i})$, a preliminary computation

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

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 $(\mathcal{R}'_{k,i}, z_k)$, the signing oracle also substitutes

$$\bar{\mathtt{R}}_{k,i} \mapsto z_{k,i} - \delta_{k,i} \bar{\mathtt{S}}_{k,i} - h_{k,i} \mathtt{D} \tag{20}$$

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

Analogous to what we did in Section 3.4, we make some additional assumptions on the adversary. Namely, we ensure that whenever the adversary makes a random oracle query

$$\delta \leftarrow \Delta(\langle \mathcal{D}' \rangle \parallel \langle \mathcal{R} \rangle \parallel \langle \mathcal{S} \rangle \parallel \langle \rho \rangle \parallel m),$$

for some $\mathcal{D}'$, $\mathcal{R}$, $\mathcal{S}$, ρ, and m, it immediately makes a "special `add` query" to the group oracle

$$(\mathtt{add}, \bar{\mathcal{R}}, \bar{\mathcal{S}}, 1, \delta)$$

to obtain the encoding of the group element $\mathcal{R}' \leftarrow \bar{\mathcal{R}} + \delta\bar{\mathcal{S}}$, and then it immediately makes the random oracle query

$$h \leftarrow H(\langle \mathcal{D}' \rangle \parallel \langle \mathcal{R}' \rangle \parallel m).$$

Analogous to what we did in Section 3.4, 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 3.2.2, but where H is modeled as a random oracle).

Suppose that initially

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

Here,

- the sum on ℓ, j corresponds to presignatures from pairs of batches whose bias has not yet been specified (and whose corresponding random nonces 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 nonces ρ_k 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

$$\pi^{-1}(\mathcal{R}^*) = \underbrace{\{a + \sum_{k,i}\bar{c}_{k,i}z_{k,i}\}}_{=z^*} + \underbrace{\{b - \sum_{k,i}\bar{c}_{k,i}h_{k,i}\}}_{=-h^*}\mathtt{D} + \sum_{k,i}\underbrace{\{\bar{d}_{k,i} - \bar{c}_{k,i}\delta_{k,i}\}}_{=0}\bar{\mathtt{S}}_{k,i}. \tag{21}$$

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 3.4. 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 signing queries that define the $h_{k,i}$'s must already be preordained. Therefore, to forge a signature, the adversary must essentially win Preimage Attack I or II on H (as in Section 3.2.2, 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 nonce 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 $\bar{\mathsf{S}}_{\ell,i}$ 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 inputs to Δ so that $\bar{d}_{\ell,i} - \bar{c}_{\ell,i}\delta_{\ell,i} = 0$ for each i will be negligible.

4.4.1 Summary

Putting all of the above together, one sees that the same security bounds as in Theorem 3 hold here as well. The argument is also easily adapted to handle additive key derivation, obtaining the same security bounds.

Acknowledgments

Thanks to Fabrice Benhamouda for pointing out that the ROS attack on blind Schnorr signatures in [BLL+22] is easily adapted to an attack on Schnorr with presignatures. This work was partially done while the author was employed at DFINITY.

References

[BFP21] B. Bauer, G. Fuchsbauer, and A. Plouviez. The one-more discrete logarithm assumption in the generic group model. In *Asiacrypt 2021*, pages 587–617, 2021. Also at https://eprint.iacr.org/2021/866.

[BHK+24] F. Benhamouda, S. Halevi, H. Krawczyk, Y. Ma, and T. Rabin. SPRINT: High-throughput robust distributed Schnorr signatures. In *Eurocrypt 2024*, 2024. Also at 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.

[BLL+22] F. Benhamouda, T. Lepoint, J. Loss, M. Orrù, and M. Raykova. On the (in)security of ROS. *J. Cryptol.*, 35(4):25, 2022. Also at https://eprint.iacr.org/2020/945.

[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.

[BS23] D. Boneh and V. Shoup. *A Graduate Course in Applied Cryptography (v0.6)*. 2023. Available at https://toc.cryptobook.us/.

[BTZ22] M. Bellare, S. Tessaro, and C. Zhu. Stronger security for non-interactive threshold signatures: BLS and FROST. Cryptology ePrint Archive, Paper 2022/833, 2022. URL https://eprint.iacr.org/2022/833.

[Can20] R. Canetti. Universally composable security. *J. ACM*, 67(5):28:1–28:94, 2020. Also at https://eprint.iacr.org/2000/067.

[CDH+21] J. Camenisch, M. Drijvers, T. Hanke, Y.-A. Pignolet, V. Shoup, and D. Williams. Internet computer consensus. Cryptology ePrint Archive, Report 2021/632, 2021. https://ia.cr/2021/632.

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

[CGG+20] R. Canetti, R. Gennaro, S. Goldfeder, N. Makriyannis, and U. Peled. UC non-interactive, proactive, threshold ECDSA with identifiable aborts. In *ACM CCS '20*, pages 1769–1787, 2020. Also at https://eprint.iacr.org/2021/060.

[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.

[CL99] M. Castro and B. Liskov. Practical byzantine fault tolerance. In M. I. Seltzer and P. J. Leach, editors, *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA, February 22-25, 1999*, pages 173–186. USENIX Association, 1999. URL https://dl.acm.org/citation.cfm?id=296824.

[CP17] A. Choudhury and A. Patra. An efficient framework for unconditionally secure multiparty computation. *IEEE Trans. Inf. Theory*, 63(1):428–468, 2017.

[CP23] B. Y. Chan and R. Pass. Simplex consensus: A simple and fast consensus protocol. In G. N. Rothblum and H. Wee, editors, *Theory of Cryptography - 21st International Conference, TCC 2023*, volume 14372 of *Lecture Notes in Computer Science*, pages 452–479. Springer, 2023. Also at https://eprint.iacr.org/2023/463.

[DDL+24] S. Das, S. Duan, S. Liu, A. Momose, L. Ren, and V. Shoup. Asynchronous consensus without trusted setup or public-key cryptography. In *ACM CCS*, pages 3242–3256, 2024. Also at https://eprint.iacr.org/2024/677.

[DEF+19] M. Drijvers, K. Edalatnejad, B. Ford, E. Kiltz, J. Loss, G. Neven, and I. Stepanovs. On the security of two-round multi-signatures. In *IEEE Symposium on Security and Privacy, SP 2019*, pages 1084–1101, 2019. Also at 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.

[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.

[Gro21] J. Groth. Non-interactive distributed key generation and key resharing. Cryptology ePrint Archive, Report 2021/339, 2021. https://ia.cr/2021/339.

[GS22a] 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.

[GS22b] J. Groth and V. Shoup. On the security of ECDSA with additive key derivation and presignatures. In O. Dunkelman and S. Dziembowski, editors, *Advances in Cryptology - EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 - June 3, 2022, Proceedings, Part I*, volume 13275 of *Lecture Notes in Computer Science*, pages 365–396. Springer, 2022. Full version in Cryptology ePrint Archive, Report 2021/1330, https://ia.cr/2021/1330.

[GS24] J. Groth and V. Shoup. Fast batched asynchronous distributed key generation. In *Eurocrypt 2024*, 2024. Also at 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. In *Selected Areas in Cryptography - SAC 2020*, pages 34–65, 2020. Also at 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. https://eprint.iacr.org/2022/550.

[Sho00] V. Shoup. Practical threshold signatures. In B. Preneel, editor, *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, volume 1807 of *Lecture Notes in Computer Science*, pages 207–220. Springer, 2000. Also at https://eprint.iacr.org/1999/011.

[Sho24] V. Shoup. A theoretical take on a practical consensus protocol. Cryptology ePrint Archive, Paper 2024/696, 2024. URL https://eprint.iacr.org/2024/696.

[SS24] V. Shoup and N. P. Smart. Lightweight asynchronous verifiable secret sharing with optimal resilience. *J. Cryptol.*, 37(3):27, 2024. Also at https://eprint.iacr.org/2023/536.

[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). In *Advances in Cryptology - CRYPTO 2022*, pages 66–96, 2022. Also at 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 Fig. 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 $i \notin \mathit{Domain}(\pi)$:
      i. $\mathcal{P} \xleftarrow{\$} E$; if $\mathcal{P} \in \mathit{Range}(\pi)$ then `abort`

ii. add $(-i, -\mathcal{P})$ and $(i, \mathcal{P})$ to π

(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 π

(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 π

(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 $a + b\mathtt{D}$, where $a, b \in \mathbb{Z}_q$ and $\mathtt{D}$ 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 $\texttt{abort}$

      ii. add $(-i, -\mathcal{P})$ and $(i, \mathcal{P})$ to π

   (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 $\texttt{abort}$

      ii. add $(-i, -\mathcal{P}_j)$ and $(i, \mathcal{P}_j)$ to π

   (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 $\texttt{abort}$

   (e) add $(-r, -\mathcal{R})$ and $(r, \mathcal{R})$ to π

   (f) return $(\mathcal{R}, z)$

Note that while the simulator may invoke $\texttt{map}$ 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 $\mathbb{Z}_q[\texttt{D}]$, we define $[P] \in \mathbb{Z}_q$ to be the value of P with $\texttt{D}$ replaced by d. For a set S of such polynomials, we define $[S] \coloneqq \{[P] : P \in S\}$. 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 Fig. 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 Fig. 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) $k \leftarrow 0$; $\mathcal{K} \leftarrow \emptyset$
   (f) 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 π
   (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 π
   (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) $k \leftarrow k + 1$; $\mathcal{K} \leftarrow \mathcal{K} \cup \{k\}$
   (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 $k \in \mathcal{K}$:
   (a) $\delta_k \xleftarrow{\$} \mathbb{Z}_q$; $r'_k \leftarrow r_k + \delta_k$; if $r'_k \in \mathit{Domain}(\pi)$ then `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) $\mathcal{K} \leftarrow \mathcal{K} \setminus \{k\}$; 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 $k \in \mathcal{K}$:
   (a) $\delta_k \xleftarrow{\$} \mathbb{Z}_q$; $r'_k \leftarrow r_k + \delta_k$
   (b) $\mathcal{R}'_k \xleftarrow{\$} E$
   (c) if $r'_k \in \mathit{Domain}(\pi)$ 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) $z_k \leftarrow r_k + \delta_k + h_k d$
   (g) $\mathcal{K} \leftarrow \mathcal{K} \setminus \{k\}$; 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) $k \leftarrow 0$; $\mathcal{K} \leftarrow \emptyset$
   (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 π
   (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 π
   (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) $k \leftarrow k + 1$; $\mathcal{K} \leftarrow \mathcal{K} \cup \{k\}$
   (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 $k \in \mathcal{K}$:

   (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) $\mathcal{K} \leftarrow \mathcal{K} \setminus \{k\}$; 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 $\{\mathtt{R}_k\}_{k \in \mathcal{K}}$, and the coefficients of these polynomials are independent of d and $\{r_k\}_{k \in \mathcal{K}}$.

It should also be clear that this symbolic attack game is equivalent to the one in Fig. 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.

C Implementing threshold Schnorr signatures

We assume we have n parties $\mathfrak{P}_1, \ldots, \mathfrak{P}_n$, at most $t < n/3$ of which may be corrupt. We also assume static corruptions; that is, at the very beginning of the attack, the adversary corrupts some subset of $t^* \leq t$ parties.

C.1 An MPC engine geared towards Schnorr

A fruitful approach to designing a threshold Schnorr signing protocol in the asynchronous communication setting is to build it on top of a highly optimized **MPC (multi-party computation) engine**. That is, rather than designing and analyzing a monolithic protocol, one can design an MPC engine that supports operations well suited to threshold Schnorr signatures, and that can be efficiently implemented while providing robustness and optimal resilience in an asynchronous communication setting.

We can conveniently define the required functionality and security properties of this MPC engine in the universal composability framework [Can20]. At a high level, our MPC engine is an ideal functionality $\mathfrak{F}_{\text{MPC}}$ supporting the following operations:

- $([r], \mathcal{R}) \leftarrow \mathsf{RandomKeyGen}()$:

  $\mathfrak{F}_{\text{MPC}}$ chooses $r \in \mathbb{Z}_q$ at random, computes $\mathcal{R} \leftarrow r\mathcal{G} \in E$, and gives $\mathcal{R}$ to the adversary and to each party as a delayed output (i.e., the adversary indicates when the output is given to any given party). The ideal functionality $\mathfrak{F}_{\text{MPC}}$ also stores r for future use.

- $[z] \leftarrow \mathsf{LinearOp}([x], [y], a, b)$:

  For public inputs $a, b \in \mathbb{Z}_q$, and previously stored values $x, y \in \mathbb{Z}_q$, $\mathfrak{F}_{\text{MPC}}$ computes $z \leftarrow ax + by \in \mathbb{Z}_q$ and stores z for future use.

- $z \leftarrow \mathsf{Open}([z])$:

  For a previously stored value $z \in \mathbb{Z}_q$, $\mathfrak{F}_{\text{MPC}}$ gives z to the adversary and to each party as a delayed output.

We have left out some details about how these operations are locally initiated. Each party locally initiates each operation, and the adversary is informed of this immediately. No corresponding data — i.e., the value $\mathcal{R}$ in RandomKeyGen and the value z in Open — is given to the adversary until at least one honest party has initiated the operation. Also, when a party receives the value $\mathcal{R}$ in RandomKeyGen, this means the corresponding value r has been generated and is available for future use. We assume that LinearOp is a local operation, which means that in the ideal functionality, the adversary is not informed when a party performs that operation, and that in an implementation, no communication takes place.

We have also left out details of liveness or completeness. Following, for example, [SS24], we do not attempt to capture these properties in an ideal functionality. Rather, these are best left as properties specific to concrete protocols. In a nutshell, these properties say that if all honest parties initiate a particular operation, all honest parties will eventually finish that operation — if and when all protocol messages between honest parties have been delivered.

The ideal functionality $\mathfrak{F}_{\text{MPC}}$ may be implemented using well-known techniques for secret sharing and threshold cryptography. See, for example, [GS22a] for a presentation of one particular approach along with discussions of other approaches. In all of these approaches, a $(t+1)$-out-of-n secret sharing scheme is used to represent the secret values stored by the ideal functionality. The operation RandomKeyGen is essentially a distributed key generation protocol, and is typically a quite expensive operation. The operation Open typically is fairly simple and takes just one round of communication (but possibly two if certain batching techniques are used to reduce communication complexity — see, for example, [CP17] for details).

C.2 A modular implementation of threshold Schnorr signatures

Using the above MPC engine, we can easily implement threshold Schnorr signatures as follows. The protocol to generate the key is:

$([d], \mathcal{D}) \leftarrow \mathsf{RandomKeyGen}()$

The protocol to sign a message m is:

$([r], \mathcal{R}) \leftarrow \mathsf{RandomKeyGen}()$
$h \leftarrow H(\langle \mathcal{D} \rangle \parallel \langle \mathcal{R} \rangle \parallel m) \in \mathbb{Z}_q$
$[z] \leftarrow \mathsf{LinearOp}([r], [d], 1, h)$
$z \leftarrow \mathsf{Open}([z])$
output $(\mathcal{R}, z)$

Here, we have highlighted operations that are non-local computations. All other operations are local.

C.3 Re-randomized presignatures

While the above is very simple, it is typically not very efficient. The problem is that in a typical implementation of $\mathfrak{F}_{\text{MPC}}$, the RandomKeyGen operation is fairly expensive. One way of improving this situation is to observe that the value r is independent of m, and so we might move the computation of $([r], \mathcal{R}) \leftarrow \mathsf{RandomKeyGen}()$ to an "offline" precomputation phase. So the pair $([r], \mathcal{R})$ can be viewed as a "presignature". These presignatures can be generated in batches, which can lead to more efficient implementations as well.

As we saw in Section 2.3, doing this significantly reduces the security of the scheme. To mitigate against this, we introduced the notion of *re-randomized presignatures* in Section 2.4. To implement this, we need to implement a random beacon that outputs a random value in $\mathbb{Z}_q$ — this should have the property that the output of one instance of the random beacon protocol remains unpredictable until at least one honest party chooses to initiate that instance. One way to do this is to use a unique threshold signature scheme such as BLS [BLS01, Bol03] and derive one beacon output by applying a random oracle to one such signature.

A more "lightweight" approach is to augment our MPC engine by adding the following operation to $\mathfrak{F}_{\text{MPC}}$:

- $[r] \leftarrow \mathsf{Random}()$:

  $\mathfrak{F}_{\text{MPC}}$ chooses $r \in \mathbb{Z}_q$ at random and stores r for future use.

Although there is no explicit output for this operation, we assume that the adversary signals each party using a delayed output mechanism to let it know that the value r has been generated and is available for future use. Typically, the implementation of this operation will be less expensive than using BLS signatures to implement a random beacon, especially if optimized using batching techniques. See, for example, [SS24].

With this operation, then with each presignature $([r_k], \mathcal{R}_k) \leftarrow \mathsf{RandomKeyGen}()$, we also associate a shift amount $[\delta_k] \leftarrow \mathsf{Random}()$. Then to sign a message m_k using this presignature and shift amount, the protocol runs as follows:

$\delta_k \leftarrow \mathsf{Open}([\delta_k])$
$\mathcal{R}'_k \leftarrow \mathcal{R}_k + \delta_k \mathcal{G}$
$h_k \leftarrow H(\langle \mathcal{D} \rangle \parallel \langle \mathcal{R}'_k \rangle \parallel m_k) \in \mathbb{Z}_q$
$[z_k] \leftarrow \mathsf{LinearOp}([r_k], [d], 1, h_k)$
$z_k \leftarrow \mathsf{Open}([z_k])$
$z'_k \leftarrow z_k + \delta_k$
output $(\mathcal{R}'_k, z'_k)$

Note that care must be taken to ensure that all parties consistently assign each presignature to a unique signing request. To this end, we envision a signing protocol that is driven by a blockchain or any other consensus mechanism that orders the signing requests, and that assigns a presignature (and shift amount) to each such signing request. It is not important to our discussion here which consensus mechanism is used. One can use a purely asynchronous protocol or a more practical protocol that assumes partial synchrony for liveness. In our formal Attack Game 2 that models re-randomized presignatures, the assignment of presignatures to signing requests is left completely to the adversary. This captures the fact that the adversary may influence the consensus mechanism in such a way that it may determine exactly when a signing request is ordered and which presignature is assigned to it. See Appendix C.7 for more details on modeling a complete signing protocol in this setting.

C.3.1 Reducing latency in the online phase

The approach to implementing re-randomized presignatures requires two consecutive Open operations, since we need must first open δ_k and only then open z_k. In a typical implementation, each of these Open operations require one round of communication. In some implementations, this can be reduced to a single round of communication, as follows. Recall that we are assuming a consensus mechanism to order signing requests. Many such mechanisms (such as PBFT [CL99], ICC [CDH+21], Simplex [CP23], and others) utilize a "two phase" voting strategy to commit blocks. If such a strategy is used, then we can use a unique threshold signature scheme such as BLS to sign a committed block, and then for each signing request triggered by some transaction in the committed block, we can derive a shift value by hashing this signature (plus other domain-separating data). Moreover, by using a high reconstruction threshold (of $n - t$) one can safely piggyback these signature shares with the second phase of voting in the consensus mechanism. This is an example of when a "high threshold" or "dual threshold" security property [Sho00, BTZ22], is required for the threshold signature scheme. Note that for threshold BLS signatures, this property holds in the random oracle model under a certain type of one-more Diffie-Hellman assumption [Gro21, BTZ22]. With this property, we can be sure that if the threshold signature on a block is revealed, then that block will be on the path of committed blocks.

For the security analysis to remain valid in this setting, it is important that the assignment of a presignature to a signing request is already determined by the time the shift value is revealed.

C.4 Re-randomizing presignatures via hashing

Now suppose we wish to re-randomize via hashing as in Section 2.5. In this case, for each signing request, instead of a presignature $([r_k], \mathcal{R}_k)$, we need a presignature pair $([r_k], \mathcal{R}_k)$ and $([s_k], \mathcal{S}_k)$, where both pairs are obtained from the RandomKeyGen. As above, these presignature pairs may be generated in advance and in batches, but then all parties must consistently assign each presignature pair to a unique signing request. If that request is to sign the message m_k, then the protocol to generate that signature runs as follows:

$\delta_k \leftarrow \Delta(\langle \mathcal{D} \rangle \parallel \langle \mathcal{R}_k \rangle \parallel \langle \mathcal{S}_k \rangle \parallel m_k)$
$\mathcal{R}'_k \leftarrow \mathcal{R}_k + \delta_k \mathcal{S}_k$
$h_k \leftarrow H(\langle \mathcal{D} \rangle \parallel \langle \mathcal{R}'_k \rangle \parallel m_k) \in \mathbb{Z}_q$
$[r'_k] \leftarrow \mathsf{LinearOp}([r_k], [s_k], 1, \delta_k)$
$[z'_k] \leftarrow \mathsf{LinearOp}([r'_k], [d], 1, h_k)$
$z'_k \leftarrow \mathsf{Open}([z'_k])$
output $(\mathcal{R}'_k, z'_k)$

The latency of this online protocol is just that of one Open operation. This can be an improvement on the latency associated with re-randomized presignatures — but it may not be, if the derivation of the random shift amounts can be folded into the consensus mechanism as outlined above in Appendix C.3.1.

C.5 Biased presignatures and more efficient distributed key generation

As already mentioned, implementing the RandomKeyGen operation is typically quite expensive. Even if we move this operation to an offline phase using presignatures, its cost may still limit the overall throughput of the system. One can reduce this cost by using a simpler protocol that achieves a somewhat weaker security property, as follows.

Suppose $\mathfrak{F}_{\text{MPC}}$ includes the following operation:

- $([r_i], \mathcal{R}_i) \leftarrow \mathsf{InputKey}_i(r_i)$:

  Party $\mathfrak{P}_i$ inputs $r_i \in \mathbb{Z}_q$ to $\mathfrak{F}_{\text{MPC}}$, who computes $\mathcal{R}_i \leftarrow r_i \mathcal{G} \in E$, immediately gives $\mathcal{R}_i$ to the adversary, and gives $\mathcal{R}_i$ to each party as a delayed output. $\mathfrak{F}_{\text{MPC}}$ also stores r_i for future use.

This operation is essentially a variation on verifiable secret sharing, allowing a party $\mathfrak{P}_i$ to share a secret r_i, but where the value $\mathcal{R}_i \coloneqq r_i \mathcal{G}$ is public.

Given this operation, we can implement a weaker kind of distributed key generation as follows. Each party $\mathfrak{P}_i$ inputs a random secret key r_i to the ideal functionality via $\mathsf{InputKey}_i$ so that every party, including the adversary, learns the public key $\mathcal{R}_i$. Note that while honest parties input random secret keys, the corrupt parties may choose arbitrary secret keys in a way that depends on the public keys of the honest parties. The parties then use a consensus protocol to agree on a set $\mathcal{I}$ of $t+1$ indices, where for each $i \in \mathcal{I}$, the operation $\mathsf{InputKey}_i$ has successfully completed. [4] The resulting key pair is $([r'], \mathcal{R}')$, where

[4] More specifically, the parties run an ACS (Agreement on a Common Set) protocol. See [DDL+24, Sho24] for a state-of-the art ACS protocol and for useful historical perspective on the ACS problem. With such a protocol, ACS will not be the bottleneck in any of our protocols, in terms of either communication or computational complexity. In a practical implementation, any consensus mechanism may be used.

$r' = \sum_{i \in \mathcal{I}} r_i$ and $\mathcal{R}' = \sum_{i \in \mathcal{I}} \mathcal{R}_i$. This is computed locally, using $\mathsf{LinearOp}$ to compute $[r']$ from $\{[r_i]\}_{i \in \mathcal{I}}$, and computing $\mathcal{R}'$ directly using the known values $\{\mathcal{R}_i\}_{i \in \mathcal{I}}$ as output by $\{\mathsf{InputKey}_i\}_{i \in \mathcal{I}}$.

One can show that the above protocol realizes the following idealized operation (which we can then effectively add to $\mathfrak{F}_{\mathrm{MPC}}$):

- $([\bar{r}], \bar{\mathcal{R}}) \leftarrow \mathsf{BiasedKeyGen}()$:

  $\mathfrak{F}_{\mathrm{MPC}}$ chooses $r \in \mathbb{Z}_q$ at random, computes $\mathcal{R} \leftarrow r\mathcal{G} \in E$, gives $\mathcal{R}$ to the adversary.

  The adversary later responds with a "bias" $(u, u') \in \mathbb{Z}_q^* \times \mathbb{Z}_q$.

  $\mathfrak{F}_{\mathrm{MPC}}$ then computes $\bar{r} \leftarrow ur + u' \in \mathbb{Z}_q$ and $\bar{\mathcal{R}} \leftarrow \bar{r}\mathcal{G} \in E$, and gives $\bar{\mathcal{R}}$ to each party as a delayed output. $\mathfrak{F}_{\mathrm{MPC}}$ also stores $\bar{r}$ for future use.

That this protocol securely realizes $\mathsf{BiasedKeyGen}$ is fairly straightforward to see (this was observed implicitly in [Gro21] and more explicitly in Section A.3.6 of [GS22a], but in a slightly different setting). For completeness, we recall the argument here, which use the random self-reducibilty property of the discrete logarithm.

We want to give a simulator $\mathfrak{S}$ (i.e., ideal-world adversary) that interacts with $\mathfrak{F}_{\mathrm{MPC}}$ with the idealized operation $\mathsf{BiasedKeyGen}$ and an adversary $\mathcal{A}$ that is interacting with the $\mathsf{InputKey}$ operations that are used to implement it. So $\mathfrak{F}_{\mathrm{MPC}}$ first chooses $r \in \mathbb{Z}_q$ at random, computes $\mathcal{R} \leftarrow r\mathcal{G} \in E$, gives $\mathcal{R}$ our simulator $\mathfrak{S}$. Now, for each honest party $\mathfrak{P}_i$, the simulator chooses $s_i \in \mathbb{Z}_q$ at random, computes $\mathcal{R}_i \leftarrow \mathcal{R} + s_i\mathcal{G}$, and can give $\mathcal{R}_i$ to $\mathcal{A}$ as if from the operation $\mathsf{InputKey}_i$. Similarly, $\mathcal{A}$ may give an input r_j to the operation $\mathsf{InputKey}_j$ from various corrupt parties $\mathfrak{P}_j$. After the index set $\mathcal{I}$ is agreed upon, the resulting secret key is

$$\bar{r} = \sum_i (r + s_i) + \sum_j r_j,$$

where the first sum is over all indices $i \in \mathcal{I}$ belonging to honest parties, and the second sum is over all indices $j \in \mathcal{I}$ belonging to corrupt parties. Therefore, $\mathfrak{S}$ selects the "bias" (u, u'), where u is the number of terms in the first sum, modulo q, and

$$u' \coloneqq \sum_i s_i + \sum_j r_j.$$

Since the number of terms in the first sum is nonzero, we have $u \in \mathbb{Z}_q^*$, assuming (reasonably) that $n < q$. It is easy to verify that this simulation is perfect.

In implementing a threshold Schnorr signature protocol, we may use $\mathsf{BiasedKeyGen}$ in place of KeyGen for generating presignatures, our analysis in Section 3.3 (and, specifically, the discussion in Section 3.3.3) guarantees security when using re-randomized presignatures. Also note we can generate presignatures in batches, which means we can implement a batch version of the $\mathsf{InputKey}$ — this can be exploited to obtain even greater efficiency improvements, such as those in [GS24]. As it turns out, we can also use $\mathsf{BiasedKeyGen}$ for generating the signing key itself — although we have not analyzed this explicitly in this paper.

C.6 Batch randomness extraction

In Appendix C.5, we saw that by allowing some bias in the resulting, we could use a simplified distributed key generation protocol. Even better performance can be obtained by utilizing the well-known "batch randomness extraction" technique — an idea that goes back at least to [HN06], but first applied to Schnorr signatures in [BHK+24]. The following discussion closely follows [GS24], and is provided here to make the presentation in this paper more self contained.

Here, we choose a certain $P \times N$ matrix W over $\mathbb{Z}_q$ (whose entries are public constants — see below), where $P = n - 2t$ and $N = n - t$. As above (in Appendix C.5), each party $\mathfrak{P}_i$ inputs a random secret key r_i to the ideal functionality via $\mathsf{InputKey}_i$, so that every party, including the adversary, learns the public key $\mathcal{R}_i$. As above, while honest parties input random secret keys, the corrupt parties may choose arbitrary secret keys in a way that depends on the public keys of the honest parties. The parties then use a consensus protocol to agree on a set $\mathcal{I}$ of N indices, where for each $i \in \mathcal{I}$, the operation $\mathsf{InputKey}_i$ has successfully completed. Let us write

$$\mathcal{I} = \{i_1, \ldots, i_N\}.$$

The parties then locally compute P biased key-pairs

$$([\bar{r}_1], \bar{\mathcal{R}}_1), \ldots, ([\bar{r}_P], \bar{\mathcal{R}}_P)$$

where

$$\begin{pmatrix} \bar{r}_1 \\ \vdots \\ \bar{r}_P \end{pmatrix} = W \cdot \begin{pmatrix} r_{i_1} \\ \vdots \\ r_{i_N} \end{pmatrix} \qquad \text{and} \qquad \begin{pmatrix} \bar{\mathcal{R}}_1 \\ \vdots \\ \bar{\mathcal{R}}_P \end{pmatrix} = W \cdot \begin{pmatrix} \mathcal{R}_{i_1} \\ \vdots \\ \mathcal{R}_{i_N} \end{pmatrix}.$$

This is computed locally, using $\mathsf{LinearOp}$ to compute the biased secret keys $[\bar{r}_1], \ldots, [\bar{r}_P]$ from $[r_1], \ldots, [r_N]$, and computing the biased public keys $\bar{\mathcal{R}}_1, \ldots, \bar{\mathcal{R}}_P$ directly using the known values $\{\mathcal{R}_i\}_{i \in \mathcal{I}}$.

The property that the matrix W must satisfy is called **super-invertibility**, which simply means that every subset of P columns of A is linearly independent. For example, one can use a Vandermonde matrix, or (as discussed in [GS24]) a Pascal matrix (which allows for more efficient computation of the biased public keys).

The security property that the above protocol satisfies can be elegantly captured by adding the following operation to our MPC engine, where we define $P \coloneqq n - 2t$ and $Q \coloneqq n - t^*$, where $t^* \leq t$ is the number of actual corrupted parties.

- $(([\bar{r}_1], \bar{\mathcal{R}}_1), \ldots, ([\bar{r}_P], \bar{\mathcal{R}}_P)) \leftarrow$ BatchedBiasedKeyGen():

  $\mathfrak{F}_{\text{MPC}}$ chooses $r_1, \ldots, r_Q \in \mathbb{Z}_q$ at random, computes

  $$\mathcal{R}_1 \leftarrow r_1\mathcal{G}, \ldots, \mathcal{R}_Q \leftarrow r_Q\mathcal{G}$$

  and gives these group elements to the adversary.

  The adversary later responds with a "bias" $(U, \mathbf{u}')$, where $U \in \mathbb{Z}_q^{P\times Q}$ is a full rank matrix and $\mathbf{u}' \in \mathbb{Z}_q^{P\times 1}$ is an arbitrary vector.

  $\mathfrak{F}_{\text{MPC}}$ then computes

  $$\begin{pmatrix} \bar{r}_1 \\ \vdots \\ \bar{r}_P \end{pmatrix} = U \cdot \begin{pmatrix} r_1 \\ \vdots \\ r_Q \end{pmatrix} + \mathbf{u}'$$

  and

  $$\bar{\mathcal{R}}_1 \leftarrow \bar{r}_1\mathcal{G}, \ldots, \bar{\mathcal{R}}_P \leftarrow \bar{r}_P\mathcal{G}$$

  and gives $(\bar{\mathcal{R}}_1, \ldots, \bar{\mathcal{R}}_P)$ to each party as a delayed output. $\mathfrak{F}_{\text{MPC}}$ also stores the values $\bar{r}_1, \ldots, \bar{r}_P$ for future use.

It is straightforward to show that the above protocol securely implements the BatchedBiasedKeyGen operation. The proof is similar to that in Appendix C.5. After the set of indices $\mathcal{I}$ is determined, the simulator $\mathfrak{S}$ chooses the matrix U as follows: for each index $i \in \mathcal{I}$ belonging to an honest party, $\mathfrak{S}$ sets the column of U corresponding to $\mathfrak{P}_i$ to the corresponding column in W; all other columns in U are set to zero. The vector $\mathbf{u}'$ is calculated as the sum over all $j \in \mathcal{I}$ belonging to corrupt parties of r_j times the corresponding column of W. It is easily seen that U has full rank (by virtue of the fact that W is super-invertible). It is easy to verify that this simulation is perfect.

Note that one can define this operation more generally, in that the parameter P could be set dynamically to a larger value determined by the adversary. This can be useful to model variations on the above protocol for batch randomness extraction. For example, it is possible that the ACS protocol happens determine a set of size greater than $n - t$, in which case P can be set to a correspondingly larger value.

In implementing a threshold Schnorr signature protocol, we may use BatchedBiasedKeyGen along the lines outlined in Section 4 for generating presignatures in batches. Our analysis in Section 4.1 guarantees security when using re-randomized presignatures. Similarly, our analysis in Section 4.4 guarantees security when re-randomizing via hashing. As discussed in Section 4.4, after generating a pair of batches of presignatures, a random nonce ρ_k is generated and published. In the threshold setting, this is accomplished using a random beacon that is only revealed after the two sets of indices that determine the biased presignatures in the pair of batches are agreed upon.

See [GS24] for a number of important efficiency improvements to implementing a batch version of InputKey, as well for the matrix-vector computations involving super-invertible matrices. By using a batch version of InputKey, one can generate larger "batches of batches" of presignatures. The paper [GS24] relies on the results in this paper to justify the protocols given there.

C.7 How to model a complete signing protocol

In this section, we show how to model a complete signing protocol. This modeling applies to Schnorr signatures with re-randomized presignatures, but can easily be adapted to any of the other variations discussed in this paper. While it would be possible to present a single, general model that encompasses all such variations, doing so would be quite cumbersome. We leave the modeling of these other variations to the reader. The discussion here follows [GS22a] fairly closely, and is provided mainly for completeness. Note that [GS22a] was in the setting of ECDSA signatures, rather than Schnorr signatures. Our presentation here has also been adjusted in a few ways to make it easier to adapt to other variations (and we comment on how to do so for a couple of variations).

We again work in the UC framework, giving an ideal functionality that models the signing process using re-randomized presignatures. Security is captured by the fact that the ideal functionality only generates signatures that are explicitly requested by honest parties. As discussed in Appendix C.3, we want to model a system which is driven by a consensus mechanism that (among other things) consistently pairs presignatures with signing requests. In our model, this pairing left to the environment, and we therefore constrain the environment in an appropriate way.

C.7.1 Ideal world

We first describe the ideal world. As usual, we have an environment $\mathfrak{Z}$, an ideal-world adversary, i.e., simulator, $\mathfrak{S}$, and an ideal functionality $\mathfrak{F}_{\text{rrp}}$. We assume static corruptions, and that $\mathfrak{Z}$, $\mathfrak{S}$, and $\mathfrak{F}_{\text{rrp}}$ are aware of the identities of the corrupt parties.

The environment $\mathfrak{Z}$ may give inputs to *honest* parties (the environment gives inputs *only* to honest parties in our model). In the ideal world, when an honest party $\mathfrak{P}_i$ receives an input from $\mathfrak{Z}$, that input is forwarded directly to the ideal functionality $\mathfrak{F}_{\text{rrp}}$. Similarly, when an honest party receives an output from $\mathfrak{F}_{\text{rrp}}$, that output is forwarded directly to $\mathfrak{Z}$. Each input to an honest party $\mathfrak{P}_i$ is either an *initialization request*, a *presignature request*, or a *signature request*.

- An *initialization request* is of the form (`init`).

  In response to such a request, $\mathfrak{P}_i$ may at a later time output a message of the form (`output-pk`, ...) to $\mathfrak{Z}$.

- A *presignature request* is of the form (`presig`, *presigID*), where *presigID* is an identifier.

  In response to such a request, $\mathfrak{P}_i$ may at a later time output the message (`output-presig`, *presigID*) to $\mathfrak{Z}$.

- A *signature request* is of the form (`sig`, *sigID*, *presigID*, m), where *sigID* and *presigID* are identifiers, and m is a message.

  In response to such a request, $\mathfrak{P}_i$ may at a later time output a message of the form (`output-sig`, ...) to $\mathfrak{Z}$.

We shall assume that $\mathfrak{Z}$ is **locally consistent**, which means that is it satisfies the following constraints for each honest party $\mathfrak{P}_i$.

- $\mathfrak{P}_i$ is given an initialization request from $\mathfrak{Z}$ only once. Moreover, this must be first input that $\mathfrak{P}_i$ receives from $\mathfrak{Z}$, and it must not receive any further inputs from $\mathfrak{Z}$ until it outputs a message of the form $(\texttt{output-pk}, \ldots)$ to $\mathfrak{Z}$.

- If $\mathfrak{P}_i$ receives a presignature request of the form $(\texttt{presig}, \mathit{presigID})$, it should not receive the same presignature request again. In addition, if $\mathfrak{P}_i$ receives a signature request of the form $(\texttt{sig}, \mathit{sigID}, \mathit{presigID}, m)$ from $\mathfrak{Z}$, this must happen only after it has output the message $(\texttt{output-presig}, \mathit{presigID})$ to $\mathfrak{Z}$. Moreover, $\mathfrak{P}_i$ should never receive another signature request with the same $\mathit{presigID}$.

- If $\mathfrak{P}_i$ receives an input of the form $(\texttt{sig}, \mathit{sigID}, \mathit{presigID}, m)$ from $\mathfrak{Z}$, it should receive no other signature request with the same sigID.

These conditions can be locally checked and enforced by each party using publicly available information, and so it is not a real constraint on $\mathfrak{Z}$.

We also assume that $\mathfrak{Z}$ is **globally consistent**, meaning that if one honest party receives a signature request of the form $(\texttt{sig}, \mathit{sigID}, \mathit{presigID}, m)$, and another receives a signature request of the form $(\texttt{sig}, \mathit{sigID}', \mathit{presigID}', m')$, then

$$[\mathit{sigID} = \mathit{sigID}' \ \vee \ \mathit{presigID} = \mathit{presigID}'] \implies (\mathit{sigID}, \mathit{presigID}, m) = (\mathit{sigID}', \mathit{presigID}', m').$$

This constraint is justified by the assumption that the system is driven by a consensus mechanism that consistently pairs presignatures with signing requests. This is the minimal constraint we need to make to ensure that the security of the distributed system can be reduced to that of an appropriate enhanced attack mode. We could also make a stronger constraint that requires that each honest party receives the same requests from $\mathfrak{Z}$ in the same order (as was done in [GS22a]), but this is not necessary.

Here is how $\mathfrak{F}_{\text{rrp}}$ works.

- Upon receiving an initialization request from $\mathfrak{P}_i$:

  If $\mathfrak{P}_i$ is the first party to send this request, $\mathfrak{F}_{\text{rrp}}$ runs the Schnorr key generation algorithm to generate a public key $\mathcal{D} \in E$ and a secret key $d \in \mathbb{Z}_q$, and then records the tuple $(\texttt{init}, \mathcal{D}, d)$. In any case, it gives $(\texttt{init}, i, \mathcal{D})$ to $\mathfrak{S}$.

- Upon receiving a presignature request $(\texttt{presig}, \mathit{presigID})$ from $\mathfrak{P}_i$:

  If $\mathfrak{P}_i$ is the first party to send this request, $\mathfrak{F}_{\text{rrp}}$ runs the presignature generation algorithm for Schnorr to get a presignature $(\mathcal{R}, r) \in E \times \mathbb{Z}_q$, and then records the tuple $(\texttt{presig}, \mathit{presigID}, \mathcal{R}, r)$. In any case, it gives $(\texttt{presig}, i, \mathit{presigID}, \mathcal{R})$ to $\mathfrak{S}$.

- Upon receiving a signing request $(\texttt{sig}, \mathit{sigID}, \mathit{presigID}, m)$ from $\mathfrak{P}_i$:

  If $\mathfrak{P}_i$ is the first party to send this request, $\mathfrak{F}_{\text{rrp}}$ fetches the corresponding tuple $(\texttt{presig}, \mathit{presigID}, \mathcal{R}, r)$, generates a random shift amount $\delta \in \mathbb{Z}_q$, generates a Schnorr signature $(\mathcal{R}', z)$, where $\mathcal{R}' = \mathcal{R} + \delta\mathcal{G}$, and then records the tuple $(\texttt{sig}, \mathit{sigID}, \mathit{presigID}, m, (\mathcal{R}', z), \delta)$. In any case, it gives $(\texttt{sig}, i, \mathit{sigID}, \mathit{presigID}, m, (\mathcal{R}', z), \delta)$ to $\mathfrak{S}$.

$\mathfrak{F}_{\text{rrp}}$ also responds to "control messages" from $\mathfrak{S}$, which determine when outputs are delivered to honest parties (which are then immediately forwarded to $\mathfrak{Z}$).

- $(\texttt{output-pk}, i)$:

  If $\mathfrak{P}_i$ previously sent an initialization request to $\mathfrak{F}_{\text{rrp}}$, fetch the recorded tuple $(\texttt{init}, \mathcal{D}, d)$ and output $(\texttt{output-pk}, \mathcal{D})$ to $\mathfrak{P}_i$.

- $(\texttt{output-presig}, \mathit{presigID}, i)$:

  If $\mathfrak{P}_i$ previously sent a presignature request $(\texttt{presig}, \mathit{presigID})$ to $\mathfrak{F}_{\text{rrp}}$, output $(\texttt{output-presig}, \mathit{presigID})$ to $\mathfrak{P}_i$.

- $(\texttt{output-sig}, \mathit{sigID}, i)$:

  If $\mathfrak{P}_i$ previously sent a signing request $(\texttt{sig}, \mathit{sigID}, \mathit{presigID}, m)$ to $\mathfrak{F}_{\text{rrp}}$, fetch the recorded tuple $(\texttt{sig}, \mathit{sigID}, \mathit{presigID}, m, (\mathcal{R}', z), \delta)$ and output $(\texttt{output-sig}, \mathit{sigID}, (\mathcal{R}', z))$ to $\mathfrak{P}_i$.

As usual in the UC framework, the environment $\mathfrak{Z}$ and the simulator $\mathfrak{S}$ may freely pass messages back and forth to each other.

NOTES:

1. To support biased presignatures, a bias (u, u') may be supplied as an extra input to each `output-presig` control message. The same bias must be supplied for all such control messages with the same *presigID*. The ideal functionality records the bias, and the initial presignature $(\mathcal{R}, r)$ is replaced by the corresponding biased presignature $(\bar{\mathcal{R}}, \bar{r})$ when this presignature is used in a signing request.
2. To support batch randomness extraction, the ideal functionality is adjusted so that a presignature request generates a batch of initial presignatures $(\mathcal{R}_1, \ldots, \mathcal{R}_Q)$. In addition, the bias is now specified in each `output-presig` control message is a matrix/vector pair $(U, \mathbf{u}')$, which defines the batch of biased presignatures $(\bar{\mathcal{R}}_1, \ldots, \bar{\mathcal{R}}_P)$. Also, each signature request must specify an index $i \in [P]$ within this batch to select an individual biased presignature to be used for signing. The consistency requirements for the environment need to be adjusted accordingly.
3. To support additive key derivation, an additive tweak may be supplied as an extra input to each signature request. The consistency requirements for the environment need to be adjusted accordingly.

C.7.2 Real world

We now describe the real world. As usual, we have an environment $\mathfrak{Z}$ and an adversary $\mathfrak{A}$. As above, we assume that $\mathfrak{Z}$ provides inputs to the honest parties of the same form as in the ideal world, and is both locally and globally consistent, as described above. These inputs, however, are passed to machines that are running the actual protocol, as opposed to being forwarded directly to an ideal functionality. These machines will also produce the outputs $(\texttt{output-pk}, \mathcal{D})$, $(\texttt{output-presig}, \mathit{presigID})$, and $(\texttt{output-sig}, \mathit{sigID}, (\mathcal{R}', z))$ that are passed to $\mathfrak{Z}$. As usual in the UC framework, the environment $\mathfrak{Z}$ and the adversary $\mathfrak{A}$ may freely pass messages back and forth to each other.

Our "real world" is actually a "hybrid world", in which the protocol machines and $\mathfrak{A}$ interact with $\mathfrak{F}_{\text{MPC}}$, including the basic operations RandomKeyGen, LinearOp, Open, as well as the operation random, introduced in Section C.3 to implement a random beacon as needed to create the random shift amounts. We can then implement a distributed signing protocol Π_{rrp} using these operations as outlined in Appendix C.3.

In that particular implementation, a random shift amount $[\delta]$ is generated using the random operation and associated with a presignature, and is then opened to reveal δ when that presignature is used to service a signature request. In Appendix C.3 we also mentioned other implementations of a random beacon, based on BLS signatures, which we could alternatively use to implement Π_{rrp}.

C.7.3 Security theorem

Theorem 4. *Protocol* Π_{rrp} *securely realizes* $\mathfrak{F}_{\text{rrp}}$ *with respect to all consistent environments.*

The statement of this theorem does not rely on any cryptographic assumptions. It does rely on the fact that Π_{rrp} is a hybrid protocol built on $\mathfrak{F}_{\text{MPC}}$. The proof is quite straightforward, and we leave the proof to the reader.

C.7.4 Consequences

The ideal function $\mathfrak{F}_{\text{rrp}}$ was designed to closely align with Attack Game 2, which defines the re-randomized presignatures enhanced attack mode. Indeed, in that attack game, an adversary interacts with a challenger, making presignature and signature requests, and that adversary corresponds directly to the combined entities $\mathfrak{Z}$ and $\mathfrak{S}$ in interacting with $\mathfrak{F}_{\text{rrp}}$ in the ideal world. Theorem 2 therefore implies that in the ideal world, the only signatures that can be produced are ones that were explictly requested by honest parties. Theorem 4 implies that the same holds in the $\mathfrak{F}_{\text{MPC}}$-hybrid world, and the UC composition theorem implies that the same holds using any secure implementation of $\mathfrak{F}_{\text{MPC}}$.

Actually, since Theorem 2 applies only in an idealized model of computation (either GGM or GGM+ROM), one must interpret the above statement with some care. Indeed, one must verify that if the real world adversary operates in this idealized model of computation, then so does the resulting simulator. However, this is straightforward to verify.

C.7.5 Discussion

Our approach to modeling a distributed signing system as an ideal functionality makes explicitly models the use of presignatures and the assumption that the system is driven by a consensus mechanism that consistently pairs presignatures with signing requests. By relegatinh the details of this consensus mechanism to the environment, we can completely abstract away the details of how this pairing is done. However, this necessitates the introduction of the constraint on the environment to be (locally and globally) consistent.

Our approach is somewhat different than other approaches taken in the literature to modeling threshold signing protocols with a presigning phase. For example, the approach taken in [CGG+20] is somewhat different — among other things, their ideal functionality does not seem to explicitly model this notion of consistent pairing of presignatures

with signing requests.[5] There are a number of other differences between our approach and that of [CGG+20]. Overall, we feel that our approach facilitates a more modular analysis: first, as discussed in Section C.7.4, the reduction from the ideal-world distributed model to the enhanced attack mode in the non-distributed setting is essentially immediate; second, the design allows for many possible instantiations of $\mathfrak{F}_{\text{MPC}}$, which can be analyzed independently. That said, the goals in [CGG+20] are actually quite different, so an apples-to-apples comparison is not really possible — indeed, [CGG+20] is focused on the synchronous setting (rather than the asynchronous setting), [CGG+20] is focused on the n-out-of-n setting, which implies no robustness (rather than t-out-of-n and achieving robustness), [CGG+20] is focused on adaptive corruptions and pro-active security (rather than static corruptions with no consideration of pro-active security), and finally, [CGG+20] is focused on ECDSA signatures (rather than Schnorr signatures).

One wrinkle to consider is the fact that our ideal functionality $\mathfrak{F}_{\text{rrp}}$ does not quite allow us to model the optimized implementation of re-randomized presignatures suggested in Appendix C.3.1. The issue is that in this implementation, the random shift amount may be revealed before any honest party explicitly issues a corresponding signing request, even though the logic guarantees all honest parties will *eventually* issue this request. It is perhaps not surprising that our model is inadequate in this instance, as this particular implementation intentionally blurs the boundaries between the consensus mechanism and the signing logic. Nevertheless, it is still possible (with a little extra work) to prove that this implementation of the distributed signing protocol (in the $\mathfrak{F}_{\text{MPC}}$-hybrid model) satisfies an appropriate security property assuming security with respect to re-randomized presignatures in the non-distributed setting. Thus, we still retain the key properties of modular design and analysis.

[5] Indeed, the presentation in [CGG+20] seems a bit inconsistent and incomplete. By inconsistent, we mean that in the protocol description in Fig. 5, signing requests take as input a message, a signature ID, and a presignature ID, while in the ideal functionality in Fig. 15, these requests take as input only a message and a signature ID. By incomplete, we mean that there does not appear to be any explicit discussion of how messages, signature IDs, and presignature IDs are to be consistently associated with one another. Presumably, some such constraint on the environment as we have imposed is necessary, but this is not discussed.