SPRINT: High-Throughput Robust Distributed Schnorr Signatures

Fabrice Benhamouda[1(✉)], Shai Halevi[1], Hugo Krawczyk[1], Yiping Ma[2], and Tal Rabin[1,2]

[1] AWS, New York, NY, USA
fabrice.benhamouda@gmail.com, shai.halevi@gmail.com, hugokraw@gmail.com
[2] University of Pennsylvania, Philadelphia, PA, USA
{yipingma,talr}@seas.upenn.edu

Abstract. We describe robust high-throughput threshold protocols for generating Schnorr signatures in an asynchronous setting with potentially hundreds of parties. The protocols run a single message-independent interactive ephemeral randomness generation procedure (i.e., DKG) followed by *non-interactive* signature generation for multiple messages, at a communication cost similar to one execution of a synchronous non-robust protocol in prior work (e.g., Gennaro et al.) and with a large number of parties (ranging from few tens to hundreds and more). Our protocols extend seamlessly to the dynamic/proactive setting where each run of the protocol uses a new committee with refreshed shares of the secret key; in particular, they support large committees periodically sampled from among the overall population of parties and the required secret state is transferred to the selected parties. The protocols work over a broadcast channel and are robust (provide guaranteed output delivery) even over asynchronous networks.

The combination of these features makes our protocols a good match for implementing a signature service over a public blockchain with many validators, where guaranteed output delivery is an absolute must. In that setting, there is a system-wide public key, where the corresponding secret signature key is distributed among the validators. Clients can submit messages (under suitable controls, e.g., smart contracts), and authorized messages are signed relative to the global public key.

Asymptotically, when running with committees of n parties, our protocols can generate $\Omega(n^2)$ signatures per run, while providing resilience against $\Omega(n)$ corrupted nodes and broadcasting only $O(n^2)$ group elements and scalars (hence $O(1)$ elements per signature).

We prove the security of our protocols via a reduction to the hardness of the discrete logarithm problem in the random oracle model.

F. Benhamouda, S. Halevi, H. Krawczyk and T. Rabin—Work done prior to joining Amazon, partially while at the Algorand Foundation.

M. Joye and G. Leander (Eds.): EUROCRYPT 2024, LNCS 14655, pp. 62–91, 2024.
https://doi.org/10.1007/978-3-031-58740-5_3

1 Introduction

In this work, we describe a suite of protocols that we call[1], aimed at generating many Schnorr signatures at a low amortized cost. SPRINT consists of a single interactive distributed key generation (DKG) for generating message-independent ephemeral randomness, followed by a *non-interactive and robust* signature generation for many messages. Here, *robustness* means that with a sufficient number of honest parties, the protocol is guaranteed to output the requested signatures.

Threshold Schnorr signature schemes have seen a revival due to applications in the blockchain space. However, the bulk of existing work focuses on the case of a small number of signers, targeting applications such as key custody and multi-signatures. For those cases, one can afford a non-robust scheme where a single misbehaving party can cause the protocol to abort: If the misbehaving party can be identified, then it can be removed before re-running the protocol. This is indeed the approach most recent schemes embrace (e.g., [8,15,27,28]). However, the remove-and-restart approach does not scale well with the number of signers, since the protocol may need to be restarted as many times as the number of misbehaving parties. Also, this approach cannot be used in a fully asynchronous setting, where there is no distinction between a malicious party that refuses to participate and an honest party that is just slow. Here, we study *robust* threshold Schnorr signatures in scenarios with many messages and many signers (possibly hundreds of them), in an asynchronous setting.

One of the motivating scenarios for considering a large set of signers signing many messages is provided by blockchain settings, where the validator nodes should generate signatures on behalf of the blockchain (see more below). That use case precludes non-robust protocols, as it requires an asynchronous protocol that remains feasible for many signers. At the same time, public blockchains provide tools such as a broadcast channel and PKI, which can simplify the design of the signature protocol. Moreover, the large number of parties makes it reasonable to assume a large honest majority, a significant advantage when building robust protocols.

Let us recall Schnorr-type signatures. They work over a group of prime order p with a generator G; a signature on a message M relative to secret key $s \in \mathbb{Z}_p$ and public key $S = s \cdot G$, has the form $(R, r + e \cdot s)$, where $r \in \mathbb{Z}_p$ is an ephemeral random secret, $R = r \cdot G$ is ephemeral randomness, and $e = \mathsf{Hash}(S, R, M) \in \mathbb{Z}_p$. A standard way to compute *robust* threshold Schnorr signatures among n parties who secret-share a long-term secret key s is to run a *distributed key generation (DKG)*[2] procedure [16] that produces a message-independent ephemeral randomness $R = r \cdot G$ where r is a fresh random value secret-shared among the parties. This phase is often called preprocessing or just DKG, and the message-

[1] SPRINT is a permuted acronym for "Robust Threshold Schnorr with Super-INvertible Packing".

[2] Throughout the paper, we use a DKG protocol for different purposes, including ephemeral Schnorr randomness generation, long-term key generation, and proactive refreshment.

independent ephemeral randomness is often called presignatures. Then, the parties use their shares of s and r to produce signature shares that can be combined into a single standard Schnorr signature. The bulk of the cost for signature generation is the DKG procedure that has $O(n^2)$ cost both in terms of bandwidth and computation.

Robust threshold Schnorr schemes have been known for over 20 years [16,37], but they are less efficient than their non-robust counterparts. These robust protocols include at least 2–3 rounds to generate message-independent ephemeral randomness, and at least one additional round for signature generation. Moreover, the randomness-generation rounds are expensive, using a bandwidth of at least $\Omega(n^2)$ broadcasted group elements. Non-robust schemes can reduce the randomness generation part to a single round, performed before knowing the message to be signed, and a single non-interactive message-dependent round (where parties just output signature shares).

Our robust signature protocol features a two-round message-independent distributed ephemeral randomness generation, followed by a single non-interactive signature generation round. However, the latter non-interactive round can produce signatures for *many messages*, hence amortizing the cost of the randomness generation protocol over many signatures. The protocols we present can produce thousands of signatures in each run, at a communication cost similar to one execution of a synchronous non-robust protocol in prior work [16].

Our protocols are flexible: they are useful in the fixed-committee setting where the same set of parties is used repeatedly, but extends seamlessly to the dynamic/proactive setting where each run of the protocol is done by a different committee with refreshed shares of the secret key. They naturally support large systems, where committees are periodically sub-sampled from among the overall population of parties and the required shared secret state is transferred to the selected parties. The protocols are also *modular*: we present a high-level protocol based on a generic agreement protocol (for the parties to agree on a set of correctly dealt shares) instantiated on an asynchronous broadcast channel. Without tying the high-level signature protocol to the details of the agreement or the communication model, we are able to take advantage of systems (such as blockchain) that natively provide agreement and communication primitives.

This agreement protocol is instrumental in achieving one of our significant design goals, namely, to perform well in the optimistic case of normal network conditions, but also to avoid degrading performance unnecessarily when network delays (possibly adversarially induced) are significant. Crucial for ensuring this property is to achieve agreement as soon as possible among a sufficient number of parties. This calls for forgoing techniques such as complete secret sharing [32] where all honest parties must receive shares of the secret, hence adding longer delays (and latency) to the protocol completion.

We next describe techniques used to achieve the above functional and performance properties of our solution, starting with two main components: (a) an early agreement protocol allowing non-complete sharing and (b) "extreme packing" that combines packed secret sharing [13] with super-invertible matrices [24] to extend the number of signatures we get from a single ephemeral-randomness creation stage.

A Simple Early-agreement Protocol. Many threshold systems require *complete secret sharing*, i.e., all honest parties must receive shares of the secret. This means that honest parties cannot terminate until they ensure that all other honest parties will eventually learn their shares. The completeness requirement often adds significant complexity to the protocol and an opportunity for the adversary to create high-latency executions in the asynchronous setting. In our protocols we forgo completeness and its adverse effects by only requiring that a sufficiently large subset of honest parties learn their shares so that they can generate signatures; there is no need to ensure that *all* honest parties get shares.

Weakening the completeness requirement of secret sharing allows us to use a very simple agreement protocol over the underlying asynchronous broadcast channel. Furthermore, the use of a broadcast channel enables *verifiable complaints* by shareholders, namely proofs that a dealer sent bad shares. Our use of these complaints is markedly different than in prior works. In protocols that aim at complete sharing (such as [19]), a party uses the complaints to inform other parties that it is missing its share, triggering a complex protocol by which the other honest shareholders help them get their missing shares. In contrast, we use the complaint to disqualify the bad dealer, there is no need to help the complaining shareholder get any more shares. This technique simplifies the agreement protocol and saves rounds of broadcast[3] (see Sects. 2.2 and 4 and our full version [4, Appendix E] for details). We believe that this simple agreement protocol could find other uses beyond DKG and threshold signatures.

Extreme Packing. To maximize efficiency, we introduce an efficiency parameter a, such that each run of the protocol produces $a(n-2t)$ signatures where t is the maximal number of corrupted parties supported by the protocol. In more detail, we use super-invertible matrices [24] to get a sharing of at least $n-2t$ random polynomials for every run of the ephemeral randomness generation, and use packed secret sharing [13] to put a random values in each of these polynomials (see Sects. 2.4 and 2.5).[4]

We pay for this extreme packing with a slight reduction in resilience: To withstand t corrupted parties, the number of nodes that we need is $n \geq 3t + 2a - 1$, compared to $n \geq 3t + 1$ for a naive protocol that generates a single signature.[5] The result is a bandwidth-optimal protocol, up to some not-too-large constants: With n parties, it provides resilience against $\Omega(n)$ corrupted parties, using broadcast bandwidth of only $O(1)$ group-elements/scalars per signature, in *both the optimistic and the pessimistic cases* (where the number of faulty parties is small or large, respectively). We stress that the odds of everybody participating

[3] Our use of an underlying broadcast channel also obviates the need to find a biclique of dealers and shareholders, which is sometimes needed when giving up completeness, and which can be computationally hard (cf. [2]).

[4] We also describe some optimizations related to faster multiplication by super-invertible matrices in our full version [4, Appendix B].

[5] Since our techniques apply in the asynchronous setting, they inherently require $n \geq 3t + 1$; see our full version [4, Appendix H].

honestly diminishes as the number of parties grows, so in the large-committee setting it becomes more important to have an efficient pessimistic path. In our protocol, the pessimistic case features additional complaints, but those add at most $O(t/a)$ group-elements/scalars per signature.

For a few examples in the static-committee setting (and assuming no complaints), setting the efficiency parameter at $a = n/5$, they withstand $t = n/5$ corrupted parties and consume broadcast bandwidth of roughly 17.33 scalars/group-elements per signature. To support $t = n/4$ we must reduce the efficiency parameter to $a = n/8$, resulting in a per-signature bandwidth of about 34 scalars/group-elements. This $O(1)$ complexity is to be contrasted with the $O(n^2)$ complexity of the standard threshold Schnorr scheme [16]. See our full version [4, Appendix C] for details.

1.1 Other Techniques

Achieving high efficiency requires the use of many ideas and techniques, beyond the two main ones that we described above. Below is a list of these techniques, in no particular order. See Sect. 2 for a detailed overview of the entire protocol and the roles that these techniques play.

Local SIMD Computation. Working with packed secret sharing increases the number of secrets shared, but current MPC solutions for using packed secret sharing entail non-trivial protocols, even for simple functions [18]. For Schnorr signatures we need to compute $s \cdot (e_1, \ldots, e_a) + (r_1, \ldots, r_a)$ where s and the r_v's are secret and the e_v's are public. While simple, an MPC protocol for computing that function still seems to require interaction, since it includes a product. Furthermore, when using simple Shamir sharing for s, some joint processing is needed to create multiple signatures.

To enable a more efficient protocol with full advantage of packing and to avoid interaction, we introduce the following technique. We share the long-term secret key in a packed vector $(s, \ldots, s)$ instead of just the single scalar s. This enables SIMD generation of the partial signature, with each party using only a local multiplication (without degree reduction), with randomization done locally as well. Using this technique, signature generation becomes non-interactive: The only communication required is for the party to broadcast their partial signature, after which anyone can assemble the signatures themselves. The cost is a reduction in the resilience to $t < (n - 2a + 2)/3$. See Sect. 2.6 for details.

Refreshing Packed Secrets. In the dynamic/proactive setting, we need to refresh the sharing of the packed vector $(s, \ldots, s)$. This requires a generalization to the GRR protocol [17], see Sect. 2.7 and our full version [4, Appendix I]. We remark that in the current version of the writeup we only prove security of the static protocol. The proof for the dynamic/proactive protocol should be a fairly straightforward extension, using the same techniques. See a brief discussion in our full version [4, Appendix G.7].

Security of Distributed Parallel Schnorr Signatures. The starting point for our protocol is similar (though not quite identical) to the GJKR distributed Schnorr signature protocol from [16], which we extend and optimize to sign many messages. However, GJKR-like protocols [16] are known to fail in the concurrent setting where the protocol is run in parallel for multiple messages; specifically, such protocols are open to ROS-type attacks [5,11]. Our work focuses on signing a given set of messages (a batch) in parallel. To enable this parallelism and avoid ROS-type attacks, we use a mitigation technique similar to prior work (e.g., [20,27]). As far as we know, prior to our work this specific technique was only analyzed in the generic group model for ECDSA signatures [20]. In our case, we show it is sufficient for proving the security of our protocols (for signing a single batch of messages) via reduction to the hardness of the discrete logarithm problem in the programmable random-oracle model. See Sect. 2.3 and our full version [4, Appendix G]. These techniques do not guarantee concurrent security for signing multiple batches in parallel. For this, Shoup [36] shows that technique from FROST can be combined with our protocols to obtain full concurrent security (see detailed discussion on this in Sect. 1.3).

Robust Threshold Signatures. Our protocols provide robustness in a strong sense. They terminate with signatures for all $a(n-2t)$ input messages as soon as $t+2a-2$ honest parties output their shares. Invalid shares can be identified based on public information and discarded. This holds in both synchronous and asynchronous networks. In the former case, after two rounds of broadcast for generating ephemeral randomness, parties generate *non-interactively* the shares from which all signatures are recovered.

Smaller Sub-sampled Committees Using a Beacon. To use our protocols in massive systems with a huge number of nodes, one needs some mechanism to sub-sample the committees from among all the nodes in the system. One natural approach is to use self-selection via verifiable random functions (VRFs), as done, e.g., in [7]. However, this results in somewhat loose tail bounds and thus somewhat-too-big committees.

Instead, we note that we can get smaller committees by using a randomness beacon to implement the sub-sampling, resulting in better bounds and smaller committees. Thus, when acting in this large dynamic committee settings, we augment the signature protocol to implement this beacon, which turns out to be almost for free in our case. See Sect. 2.8 for more details. See also our full version [4, Appendix A] for an additional optimization in this setting: using smaller optimistic parameters by default with a safe fallback mechanism to pessimistic parameters.

1.2 Prior Work

Recent years saw a lot of activity trying to improve the efficiency of threshold signature schemes, including underlying techniques such as verifiable secret

sharing (VSS) and distributed key generation (DKG), much of which focused on asynchronous protocols and some emphasizing robustness (guaranteed output delivery). Below we focus on some of the more recent works on these subjects.

Threshold Signatures. Komlo and Goldberg described FROST [27], a non-robust threshold Schnorr signature protocol that requires a single-round signing protocol after a single-round preprocessing phase. The improved round complexity comes at the expense of robustness, as it uses additive sharings and requires correct participation of all prescribed signers. In our case, we use two rounds of interaction in a message-independent phase but can then generate multiple signatures non-interactively and with guaranteed output delivery. Our schemes are designed to work in an asynchronous regime hence requiring a super-majority of honest parties (see details in our full version [4, Appendix H]).

ROAST [35] presents a wrapper technique that can transform concurrently secure non-robust threshold signature schemes with a single signing round and identifiable abort into a protocol with the same properties but also robust in the asynchronous model. In particular, this applies to the FROST protocol resulting in a scheme with concurrent security for any threshold $t < n$ and optimal robustness for up to $n - t$ parties. The price for this strengthening is significant: it involves $O(tn^2 + tn\lambda)$ per-signature transmitted bits (λ is a security parameter) assuming a trusted coordinator and $O(tn^3 + tn^2\lambda)$ without the coordinator; whereas we only require $O(\lambda)$ broadcasted bits (strictly better even when considering a quadratic overhead of the underlying broadcast).

Garillot et al. [15] implement a threshold Schnorr signature based on deterministic signing, e.g., EdDSA, in order to avoid the potential risks of randomness reuse. They present a dishonest-majority non-robust scheme using zero-knowledge proof and garbling techniques that, while optimized for this specific application, is much more expensive than protocols that do not offer deterministic signing (like FROST and our SPRINT protocols).

Lindell [28] presents a threshold Schnorr signature scheme proven under standard assumptions in the UC model. The focus of that work was conceptual simplicity and UC security rather than optimal efficiency. As in FROST, it utilizes additive sharing, hence necessitating the choice of a new set of signers when a chosen set fails to generate a signature.

For ECDSA signatures, Groth and Shoup [19] recently described a rather efficient ECDSA signing protocol, with emphasis on guaranteed output delivery over asynchronous channels. (The underlying VSS in their work achieves completeness, which is not needed in our case.) They use verifiable complaints in order to notify other parties that they do not have a share. These complaints trigger a complex protocol, by which honest shareholders help each other to get all their missing shares.

Joshi et al. [25] address the lack of concurrent security in the basic threshold Schnorr scheme from [16] by running two DKG executions per signature and using a mitigation technique similar to the one we use here to bind a batch of

messages to be signed. However, while our solution generates multiple signatures with a single DKG run, theirs requires two such runs per single signed message.

Distributed Randomness Generation (DKG). [6] As we said, a key distinction between our work and previous DKG protocols in the signature setting [1,30,38], is that we *do not require complete sharing* (where all honest parties must receive their shares). While completeness may be desired in traditional MPC applications, eschewing this requirement is not a weakness but a feature in our case, as it enables more efficient signature protocols.

Neji et al. [30] design a DKG intended to avoid the need to reveal the shares of inactive (or slow) shareholders for disqualification as required in the GJKR [16] solution. However, they do so by requiring additional rounds of interaction and significant extra computational cost, namely the party who gets complained does $O(n)$ group additions and each other party does $O(t)$ scalar multiplications where t is the corruption threshold (these costs are merely for handling complaints beyond the verification). We achieve higher performance by using publicly verifiable complaints: in our protocols, each party can verify that a complaint is valid by doing a constant number of group operations and without any additional interaction.

Yurek et al. [38] described a randomness generation protocol over asynchronous communication channels, in the context of the offline phase of generic secure MPC. They provide completeness for secret sharing needed for their MPC applications. As in a recent work by Groth and Shoup [19], they use verifiable complaints, yet unlike our work, they do not disqualify dealers upon a verifiable complaint—they instead complete the set of shares. Their asynchronous VSS has an amortized network bandwidth $O(n \log n)$ in the optimistic case and $O(n^2 \log n)$ in the pessimistic case.

Abraham et al. describe Bingo [1], a packed method for asynchronous secret sharing that allows a dealer to generate many sharings at an amortized communication cost of $O(\lambda n)$ per secret. This solution requires KZG-style polynomial commitments [26] to get completeness (and thus relies on pairing-friendly groups). Specifically, the dealer performs a KZG commitment to a polynomial of degree $2t$ (where $n = 3t+1$), which concretely is slightly more expensive than our protocol. Also, our agreement sub-protocol makes a more direct usage of the underlying broadcast channel than the agreement in Bingo, and is more efficient.

Various other papers (e.g., [9,10]) deal with the question of asynchronous DKG. However, they do not directly relate to our paper as the main thrust of their work is reaching an agreement in the asynchronous setting. In contrast, we assume an underlying broadcast channel, simplifying the agreement significantly.

[6] Recall we use DKG to refer to distributed key generation for long-term keys, for generating ephemeral randomness as needed in Schnorr signatures, and also for proactive refreshment.

1.3 Subsequent Work

There have been several papers published after our paper was first made public.

Shoup's Many Faces of Schnorr. In [36], Shoup presents a unifying framework for obtaining robust concurrently-secure threshold Schnorr signatures combining techniques from our work and FROST [29]. This framework applies to two-phase protocols, like ours, consisting of an offline phase for generating "presignatures" (a.k.a., ephemeral randomness), and then an online phase for generating signatures from those presignatures. The concurrent-security aspect of these protocols means that many copies of the online phase can be run concurrently, as long as sufficiently many unused presignatures are available. Shoup shows that concurrent security can be added to any protocol within this framework (including ours) in one of two ways: either using two fresh DKG-generated secret sharing of ephemeral randomness à-la-FROST (thus doubling the cost), or using a randomness beacon (which adds rounds of communication).

Groth-Shoup Asynchronous Robust DKG. In [21], Groth and Shoup present an asynchronous robust DKG protocol which can be used as a basis for a threshold signature protocol, that require a total of $O(n\lambda)$ bits of point-to-point communication per signature over the optimistic path (roughly when all parties behave honestly), amortized over $O(n^2)$ signatures. The optimistic path communication complexity matches (asymptotically) our communication complexity of $O(\lambda)$ bits broadcast per signature.[7]

However, the Groth-Shoup protocol is a lot less efficient on the pessimistic path, when parties misbehave: Its communication complexity increases by a factor $O(t')$ where t' is the number of actual misbehaving parties. In contrast, the communication complexity of our protocol increases by at most a small constant factor, no matter how many parties misbehave (as long as there are at most t of them). On the other hand, the Groth-Shoup protocol can withstand up to $(n-1)/3$ misbehaving parties, compared to our $t \leq (n-2a+1)/3$. Our protocol is therefore a better choice in the large-committee setting, where consistent performance also on the pessimistic path is important, and where it is reasonable to assume a larger honest majority. The Groth-Shoup protocol may be better in the small-committee setting, where higher resilience is more important and assuming the optimistic path makes more sense.

The main difference between our protocol and Groth-Shoup stems from the fact that the latter requires complete secret sharing, where all the honest parties get their shares. In particular, if a dealer misbehaves and does not appropriately distribute shares to some honest parties, these honest parties need other honest parties to help them reconstruct their shares, whereas our protocol just disqualifies that dealer. On the other hand, the Groth-Shoup protocol uses complete secret sharing to eliminate the need for polynomial commitments in the

[7] Broadcasting messages of size $\ell \geq n\lambda$ bits, as done in our protocol, can be achieved using a total point-to-point communication of $O(\ell n)$ bits [14,29].

sharing phase. Instead, they use error correction to reconstruct signature shares at the end of the protocol without having to check validity against some public commitment.

Another difference is that [21] uses a construction based on Pascal triangle for super-invertible matrices, which is better than the small Vandermonde construction (see details in [4, Appendix B]). This way, they reduce the cost of evaluating the product by the super-invertible matrix from $\approx (b-1)n \log n / \log p$ scalar-element products in that solution to $\approx b(n-(b+1)/2)+1$ group additions (which correspond to about $(b(n-(b+1)/2)+1)/(1.5 \log p)$ scalar-element products). Our proposal to use the ECFFT-EXTEND algorithm (see Sect. 2.4) is more efficient asymptotically ($O(k \log k)$ scalar-element products, for $k = \max(b, n-b)$) but the Pascal solution would most likely perform better up to $n \approx 8000$.

1.4 Organization

The rest of this manuscript is organized as follows: In Sect. 2, we provide a high-level step-by-step overview of our protocols and the various components that are used in them. In Sect. 3, we describe in more detail our high-level protocol for the static (fixed-committee) and dynamic settings. In Sect. 4, we describe the basic agreement protocol that we use in the static-committee setting, the agreement in the dynamic setting can be found in our full version [4, Appendix E]. Security proofs and additional details are deferred to appendices. In particular, in the full version [4, Appendix D], we discuss how to use SPRINT in one of our motivating applications to implement a large-scale signature service over a public blockchain.

2 Technical Overview

We consider a static setting where the set of parties (a shareholder committee) is fixed and a dynamic one where shareholder committees change over time while keeping the system's signing key (in particular, its public verification key) unchanged. In the latter case, shares are refreshed and proactivized between committees. We begin by describing our protocols in the static committee setting, and discuss only towards the end the extra components for the dynamic/proactive settings. The basic protocols for these two settings are shown in Figs. 1 and 2.

In the static case, we have a committee that holds shares of the long-term secret key s, shared via a degree-d polynomial $\mathbf{F}(X)$ with party i holding $\sigma_i = \mathbf{F}(i)$ (for some degree d that we determine later) and where $s = \mathbf{F}(0)$. They first run a distributed key-generation (DKG) protocol to generate a sharing of ephemeral randomness, then use their shares of the long-term secret and ephemeral randomness to generate Schnorr-type signatures on messages. The DKG and signature protocols can be pipelined, where the committee uses the randomness that was received in the previous run to sign messages, and at the same time prepares the randomness for the next run.

While the static setting features just a single committee, we still often refer to parties as *dealers* when they share secrets to others, and as *shareholders* when

they receive those shares. In the dynamic setting, these will indeed be different parties, but in the static case, they may be the same.

Notations. We use Greek letters (e.g., σ, ρ, π, ϕ) and lowercase English letters (e.g., e, r, s) to denote scalars in $\mathbb{Z}_p$, and also use some English lowercase letters to denote indexes (i, j, k, ℓ, u, v) and parameters (a, b, n, t). We denote the set of integers from x to y (inclusive) by $[x, y]$, and also denote $[x] = [1, x]$. We rely on a group of prime order p generated by G. We use the additive notation for this group. Group elements are denoted by uppercase English letters (G, S, R, etc.). Polynomials are denoted by bold Uppercase English letters ($\mathbf{F}, \mathbf{H}, \mathbf{I}, \mathbf{Y}, \mathbf{Z}$), and commitments to them are sometimes denoted with a hat ($\hat{\mathbf{F}}, \hat{\mathbf{H}}$).

2.1 Starting Point: The GJKR Protocol

Our starting point is the protocol of Gennaro et al. [16] for distributed key generation (DKG), and a variation on their use of that protocol for Schnorr signatures. In their DKG protocol, each dealer uses Verifiable Secret Sharing (VSS) to share a random value; parties then add all the shares from dealers that shared their values correctly (thus requiring an agreement protocol on which dealers fall in this set, denoted QUAL). Specifically, each dealer D_i shares a random ephemeral secret (which is later used to compute ephemeral randomness and partial signatures) using a degree-d' polynomial $\mathbf{H}_i$ (for some degree d' that we define later), and commits publicly to this polynomial. Concretely, D_i shares the random ephemeral secret $\mathbf{H}_i(0)$ by sending shares $\mathbf{H}_j(i)$ to each shareholder P_i.

The shareholders then agree on a set QUAL of "qualified dealers" whose values will be used, and a corresponding shareholder set HOLD that were able to receive valid shares. Shareholders in HOLD can compute shares for the ephemeral secrets from the shares that they received from these qualified dealers. Namely, each shareholder can add the shares (i.e., the Shamir shares of ephemeral secrets of dealers) that they received from dealers in QUAL, and the resulting ephemeral secret is shared via the polynomial $\mathbf{H} = \sum_{i \in \mathsf{QUAL}} \mathbf{H}_i$.

In our protocol, shareholders use their shares on polynomials $\mathbf{H}$ (the ephemeral secret) and $\mathbf{F}$ (the long-term secret) to compute Shamir shares of the signatures, and then reconstruct the signatures themselves. We note that this is somewhat different from the signature protocol in [16]: there, it is the dealers in QUAL that generate the signature (and HOLD is only used as a backup to reconstruct the input of misbehaving dealers), whereas we let the shareholders in HOLD generate the signature directly. Our variant could be more round-efficient in some cases, and is easier to deploy in a proactive setting where the long-term key is shared using Shamir sharing (as opposed to additive sharing as used in the GJKR protocol). But otherwise these protocols are very similar.

Pedersen vs. Feldman Commitments. It was pointed out by Gennaro et al. [16] that sharing randomness usually requires the dealers to commit to their sharing polynomials using statistically-hiding commitments such as Pedersen's

[33]. Using the less expensive Feldman secret sharing, where dealers commit to coefficients h_{ij} of their polynomials by broadcasting the group elements $h_{ij} \cdot G$, are susceptible to rushing attacks in the DKG setting. Luckily, Gennaro et al. prove in [16, Sec 5] that for the purpose of generating the ephemeral randomness for Schnorr signatures, it is safe to use Feldman secret-sharing, and their proof techniques extend to our signature protocol as well.

We note that for efficiency reasons, in our protocols we use commitments to the value of the polynomials at certain evaluation points rather than to the coefficients as done in [16] (see details in [4, Appendix A]).

2.2 The Agreement Protocol

We utilize the QUAL-agreement protocol in two different settings: for generation of ephemeral randomness (in both the static and dynamic setting), and for re-sharing of the long-term key (in the dynamic setting only). We observe that randomness generation is less demanding of the agreement protocol than key-refresh: For key-refresh we need the shareholders to have shares from at least $d + 1$ dealers (d is the degree of the sharing polynomials), whereas randomness generation can work even with a single honest dealer. Therefore, in the static setting we use a weaker (and more efficient) agreement protocol than in the dynamic setting. Both protocols use PKI, and both operate over a total-order (aka atomic) broadcast channel, providing eventual delivery of messages from honest parties, sender authentication, and prefix consistency (i.e., the views of any two honest parties are such that one is a prefix of the other).

We start with the more efficient (but weaker) protocol for the static setting. The protocol begins with the dealers distributing their shares, and then the shareholders engage in a protocol to agree on sets of "qualified" and "bad" dealers $\mathsf{QUAL}, \mathsf{BAD}$, and a set of shareholders HOLD. We want the following properties: (i) every shareholder in HOLD received valid shares from every dealer in QUAL, and (ii) BAD consists entirely of dishonest dealers. This protocol is parameterized by d_0, d_1 (to be defined later as a function of the number of corrupt parties and some additional parameters), and it ensures that $|\mathsf{HOLD}| \geq d_0$ and $|\mathsf{QUAL}| + |\mathsf{BAD}| \geq d_1$.

In more detail, each dealer D_i broadcasts all the shares that it deals, encrypted under the keys of their intended recipients, together with commitments to the sharing polynomial $\mathbf{H}_i$. As this information is visible to all, shareholders that receive shares that are inconsistent with the commitments can broadcast a *verifiable complaint* against a dealer, consisting of a proof that the dealer has sent them a bad share.

The shareholders initially set QUAL to the first d_1 dealers whose messages appeared on the broadcast channel. Then each shareholder broadcasts verifiable complaints if they have any, and otherwise they broadcast the empty set (signifying that they have all the shares from dealers in QUAL). Now, QUAL contracts by eliminating all the dealers who have a valid verifiable complaint against them on the broadcast channel, moving them to the set BAD. The set HOLD is fixed to the first d_0 shareholders who broadcast verifiable complaints (or the empty set)

that were verified as valid complaints. By construction, we have $|\mathsf{HOLD}| \geq d_0$ and $|\mathsf{QUAL}| + |\mathsf{BAD}| \geq d_1$, and the set BAD contains only (verifiably) dishonest dealers. Also, since QUAL, BAD, and HOLD are determined by what is visible on the broadcast channel, then all honest shareholders that read up to some point in the channel will agree on these sets. This protocol's specification can be found in Fig. 3, and the proof is provided in Theorem 4.1.

In the dynamic setting (that includes also key-refresh), we need to ensure $|\mathsf{QUAL}| \geq d_1$ (as opposed to just $|\mathsf{QUAL}| + |\mathsf{BAD}| \geq d_1$). To that end, we run iterations of the basic protocol above. At the beginning of the $i+1$'st iteration, we add to QUAL as many new dealers as the number of dealers that were added to BAD in the i'th iteration. Once no more dealers are added to BAD, we have $|\mathsf{QUAL}| \geq d_1$, and we are done. A full specification is in our full version [4, Appendix E].

2.3 Signing Many Messages in Parallel

Our large-scale signature service needs to handle signing many messages in parallel, which brings up a security problem: The proof of security from [16, Sec 5] when using Feldman commitments for Schnorr signatures, requires that the reduction algorithm makes a guess about which random oracle query the adversary intends to use for the signature. When signing many messages in parallel, the reduction will need to guess one random-oracle query per message, leading to exponential security loss. Moreover, Benhamouda et al. demonstrated in [5] that this is not just a problem with the reduction, indeed this protocol is vulnerable to an actual forgery attack when many messages are signed in parallel. To fix this problem, we use a mitigation technique somewhat similar to [20,27], where the ephemeral secrets are all "shifted" by a public random value δ, which is only determined after all the messages and commitments are known.

As recalled in the introduction, a Schnorr signature on a message M^v relative to secret key s and public key $S = s \cdot G$, has the form $(R^v, r^v + e^v \cdot s)$, where r^v is an ephemeral random secret, $R^v = r^v \cdot G$, and $e^v = \mathsf{Hash}(S, R^v, M^v)$, where Hash maps arbitrary strings into $\mathbb{Z}_p$. (We are using a superscript v to indicate a plurality of messages and their respective signatures.) In our context, we first run DKG to generate all the required r^v's and corresponding R^v's, and get from the calling application all the messages M^v's to be signed. Then we compute $\delta = \mathsf{Hash}(S, (R^1, M^1), (R^2, M^2), \ldots)$ and $\Delta = \delta \cdot G$. The signature on M^v is then set as $(R^v + \Delta, r^v + \delta + e^v \cdot s)$, where $e^v = \mathsf{Hash}(S, R^v + \Delta, M^v)$.

With this mitigation technique, the reduction only needs to guess the random-oracle query in which δ is computed, recovering the argument from [16, Sec 5] and reducing security to the hardness of computing discrete logarithms in the random-oracle model. See our full version [4, Appendix G.3]. We note that our specific mitigation techniques provide security for *a single run of the protocol* on input a set of multiple messages to be signed, but it does not imply concurrent security for multiple parallel runs of the protocol on different sets of messages. Following [36], we can obtain concurrent security by either adopting the FROST mitigation (that requires doubling the DKG cost) or by relying on a beacon (which would add one broadcast round).

2.4 Using Super-Invertible Matrices

As described so far, we would need to run a separate copy of the DKG protocol to generate each ephemeral secret r^v, but we can do much better. For starters, assume that we can ensure many honest dealers in the set QUAL (say at least b of them). Then we can use a (public) super-invertible matrix [24] to generate b random ephemeral values in each run of the protocol.

Recall that the DKG protocol has each dealer D_i share a random polynomial $\mathbf{H}_i$, then the shareholders compute a single random polynomial $\mathbf{H}' = \sum_{i \in \mathsf{QUAL}} \mathbf{H}_i$ and the ephemeral random secret is $\mathbf{H}'(0)$. Intuitively, the polynomial $\mathbf{H}'$ is random if even a single $\mathbf{H}_i$ is random, so a single honest dealer in QUAL is enough to get a random ephemeral value. But if we have many honest dealers in QUAL, then we can get many random polynomials. Specifically, suppose we have b honest dealers in QUAL and let $\Psi = [\psi_i^u]$ be a b-by-n super-invertible matrix, i.e., each b-by-b sub-matrix of Ψ is invertible. Then we still have each dealer D_i share just a single polynomial $\mathbf{H}_i$, but now the shareholders can construct b random polynomials $\mathbf{H}^1, \ldots, \mathbf{H}^b$, by setting $\mathbf{H}^u = \sum_{i \in \mathsf{QUAL}} \psi_i^u \mathbf{H}_i$ for all $u \in [b]$. By the same reasoning as before, if we have b honest dealers in QUAL with random input polynomials $\mathbf{H}_i$, then the b output polynomials will also be random and independent since the b-by-b matrix corresponding to the rows of these b honest dealers is invertible.

The actual proof is more involved since we still use Feldman commitments in the protocol, which means that a rushing adversary can bias the output polynomials somewhat. But using essentially the same reduction as before, we can still reduce the security of the Schnorr signature protocol to the hardness of computing discrete logarithms in the random oracle model. One technical point is that the security proof in the asynchronous communication model requires that the set QUAL is included in the hash function query that determines δ. That is, we compute $\delta = \mathsf{Hash}(S, \mathsf{QUAL}, (R^1, M^1), (R^2, M^2), \ldots)$. The reason is that in the asynchronous case, we cannot guarantee that all honest dealers will be included in QUAL. If we didn't include it in the hash query, then the simulator would have to guess the set QUAL, incurring at least an $\binom{n}{b}$ loss factor in security.

We note that to ensure b honest dealers in QUAL, it is enough to run the "weaker" agreement protocol (Fig. 3) with $d_1 = b + t$, where t is an upper bound on the number of dishonest dealers. Indeed, that protocol ensures that $|\mathsf{QUAL}| + |\mathsf{BAD}| \geq d_1 = b + t$, and BAD contains only dishonest dealers. Therefore, the number of dishonest dealers in QUAL is at most $t - |\mathsf{BAD}|$, and the number of honest dealers is at least $|\mathsf{QUAL}| - (t - |\mathsf{BAD}|) = |\mathsf{QUAL}| + |\mathsf{BAD}| - t = d_1 - t = b$.

Faster Multiplication by a Super-Invertible Matrix. While the use of super-invertible matrices enables us to produce many more random shared secrets without increasing bandwidth, computing all these sharings requires that each shareholder multiply their sub-shares by that super-invertible matrix "in the exponent".[8] The super-invertible matrix multiplication is the most computa-

[8] We use additive notation for group operations, but sometimes use the traditional exponentiation terminology.

tionally intensive operation in the protocol. We thus should carefully implement the matrix multiplication to have good computational efficiency in practice.

We propose two solutions to make these operations more efficient. The first solution, pointed out to us by Victor Shoup, is to use a Vandermonde matrix Ψ corresponding to the powers of small scalars. We show in our full version [4, Appendix B.1], that a variant of the Horner's rule allows to evaluate the multiply-by-Ψ operation using $(b-1)n$ scalar-by-element products with $\log n$-bit scalars (instead of full-length scalars, that is $\log p$-bit scalars). This is equivalent to about $(b-1)n \log n / \log p$ full scalar-by-element product, that is a $\log n / \log p$ speed-up over the naive solution. In practice, p has at least 256 bits, while $n = b+t$ varies but is unlikely to be higher than 10 bits, so this is a more than 25× speed-up.

Our second solution is new and consists of selecting Ψ so that it corresponds to FFT-related operations. However, when implementing Schnorr signatures over the elliptic curve ED25519, the scalar field $\mathbb{Z}_p$ does not even have a 2^3-th root of unity.[9] Instead, we show that we can use the ECFFT-EXTEND algorithm from Ben-Sasson et al. [3], resulting in $O(k \log k)$ scalar-by-element products, where $k = \max(b, t)$ and $n = b+t$. This is asymptotically better than the first solution. Details are provided in the full version [4, Appendix B.2].

We implemented both solutions, benchmarked them, and report results in [4, Appendix B.3]. In short, for ED25519, when $b = t$ is a power of 2, the small Vandermonde matrix solution is better in practice for up to $b = t = 2^8 = 256$, after which the ECFFT solution is more efficient.[10] The benchmarking code is available from https://github.com/fabrice102/ecfft-group, under the MIT license. This code is based on the code [6] and adapts it to work with polynomials with coefficients in a group, instead of in the base field.

2.5 Using Packed Secret Sharing

Similarly to above, we can also assume many honest parties among the set HOLD of shareholders, and use packed secret sharing [13] to get even more ephemeral shared values: If HOLD contains at least $2t + a$ shareholders (for some $a \geq 1$), then we can let each shared polynomial pack a values rather than just one: Each shared polynomial $\mathbf{H}^u$ will have degree $d' \geq t + a - 1$ (rather than $d' = t$) and will encode the a values $\mathbf{H}^u(0), \mathbf{H}^u(-1), \ldots, \mathbf{H}^u(-a+1)$. (Below we denote these scalar values by $r^{u,v} = \mathbf{H}^u(1-v)$, with the corresponding group elements $R^{u,v} = r^{u,v} \cdot G$.)

Importantly, this amplifies the effect of using super-invertible matrices: We have each dealer D_i sharing a single random polynomial $\mathbf{H}_i$ of degree d', packing a values, and we derive b random degree-d' polynomials $\mathbf{H}^u$ from these sharings, which gives us $a \cdot b$ shared random scalars.

[9] $p = 2^{252} + 27742317777372353535851937790883648493$ and the factorization of $p-1$ is $2^2 \times 3 \times 11 \times 1982114232309307540130845257636 97 \times 27660262428164223993721868055713982666 8747$.

[10] The ECFFT solution performs better for $b = t$ is a power of two. But we show in the full version [4, Appendix B.2] that it also works for general b and t, with a cost depending on the smallest power of 2 larger or equal to $\max(b, t)$.

2.6 More Efficient Signing

Once the ephemeral secrets are shared, we use them—together with the shared long-term secret key—to generate many signatures. Computing on the packed ephemeral secrets would generically require a full-blown secure-MPC protocol among the shareholders, but we observe that we can generate all the a signatures from each packed random polynomial with only a single share-reconstruction operation.

To see how, recall again that a Schnorr-type signature has the form $(R^v, r^v + e^v \cdot s)$.[11] Our shareholders hold Shamir sharings of the secret key s and the vector $(r^1, r^2, \ldots, r^a)$ of ephemeral secrets (where $r^v = \mathbf{H}(1-v)$ for $v \in [a]$). Also, the public key S, the messages M^v's, and the group elements R^v's are publicly known, so everyone can compute all the scalars $e^v = \mathsf{Hash}(S, R^v, M^v)$. To improve efficiency, we also share the long-term key s in a packed form, namely the shareholders hold a Shamir sharing of the vector $(s, s, \ldots, s)$, via a polynomial $\mathbf{F}$ of degree $d = t + a - 1$ (i.e., $\mathbf{F}(1-v) = s$ for $v \in [a]$). All they need to do, therefore, is compute the pointwise linear function $(r^1, r^2, \ldots, r^a) + (e^1, e^2, \ldots, e^a) \odot (s, s, \ldots, s)$.

While pointwise addition can be computed locally, computing the pointwise product $(e^1, e^2, \ldots, e^a) \odot (s, s, \ldots, s)$ seems like still requiring a nontrivial interactive protocol, even for a known vector of e^v's. But we can eliminate even this little interaction, by assuming a larger honest majority and using higher-degree polynomials for the ephemeral randomness. Specifically, we assume that HOLD contains at least $2t + 2a - 1$ shareholders (so at least $t + 2a - 1$ honest ones), and modify the DKG protocol so that the sharing of the ephemeral secrets is done with random polynomials of degree $d' = d + a - 1 = t + 2a - 2$ (rather than degree $t + a - 1$).

Since the e^v's are known, each shareholder can interpolate the unique degree-$(a-1)$ polynomial that packs the vector $(e^1, \ldots, e^a)$. Denote this polynomial as $\mathbf{Z}$ (we have $\mathbf{Z}(1-v) = e^v$ for $v \in [a]$). Then each shareholder j with a share $\sigma_j = \mathbf{F}(j)$ for the long-term secret, can *locally* compute $\sigma'_j = \mathbf{Z}(j) \cdot \sigma_j$. Note now that the σ'_j's lie on the polynomial $\mathbf{Z} \cdot \mathbf{F}$ of degree $d + a - 1$ that packs the vector $(e^1 \cdot s, \ldots, e^a \cdot s)$, since $(\mathbf{Z} \cdot \mathbf{F})(1-v) = e^v \cdot s$ for $v \in [a]$.

Each shareholder j, with share ρ_j on an ephemeral-randomness polynomial, computes and broadcasts $\pi_j = \sigma'_j + \rho_j$, and we note that these π_j's lie on a polynomial of degree d' that packs all the values $(r^1 + e^1 s, \ldots, r^a + e^a s)$. Moreover, if the ephemeral secrets were shared via a *random* degree-d' polynomial, then the π_j's constitute *a random sharing* of that vector. After seeing $d' + 1 = t + 2a - 1$ valid shares of these broadcast values, everyone can reconstruct the polynomial and read out all the scalars $\phi^v = r^v + e^v \cdot s$ that are needed for these a signatures.

2.7 The Dynamic Setting

So far, we have described our protocols for the static (fixed committee) setting. Here we present the additional components that we need in the dynamic case,

[11] We suppress here the index u, which is irrelevant for this discussion.

where we have different committees for the dealers and shareholders. Importantly, in all the protocols above we never assumed that the dealers and shareholders are the same committee, so they all still work as-is also in the dynamic setting. What is missing is a share-refresh protocol where the dealers can pass to the shareholders a sharing of the long-term secret s. Here we essentially just use the GRR protocols of Gennaro et al. from [17], with a minor adaptation since we need to share it in a packed manner.[12]

Each dealer D_i begins with a share σ_i of the long-term secret key s, shared using a "packed" polynomial $\mathbf{F}(X)$ of degree $d = t+a-1$. Namely, $\sigma_i = \mathbf{F}(i)$, and $\mathbf{F}(0) = \mathbf{F}(-1) = \cdots = \mathbf{F}(1-a) = s$. In addition, everyone knows a commitment to $\mathbf{F}$. D_i reshares its share using a fresh random degree-d polynomial $\mathbf{F}_i$ with $\mathbf{F}_i(0) = \mathbf{F}_i(-1) = \cdots = \mathbf{F}_i(1-a) = \sigma_i$, and also commits publicly to $\mathbf{F}_i$.

This is done in parallel to the sharing of the random, degree-d', polynomial $\mathbf{H}_i$. The shareholders then engage in an agreement protocol (full protocol description can be found in [4, Appendix E]) to determine the sets HOLD of shareholders, $\mathsf{QUAL}_1, \mathsf{BAD}_1$ for the $\mathbf{H}$ dealers, and $\mathsf{QUAL}_2, \mathsf{BAD}_2$ for the $\mathbf{F}$ dealers, with $|\mathsf{HOLD}| \geq n-t$, $|\mathsf{QUAL}_1| \geq n-t$, and $|\mathsf{QUAL}_2| \geq d+1$.[13] Having received $\sigma_{ij} = \mathbf{F}_i(j)$ from each dealer $D_i \in \mathsf{QUAL}_2$, P_j then computes their share of the long-term secret as $\sigma'_j = \sum_{i \in \mathsf{QUAL}_2} \lambda_i \sigma_{ij}$. The λ_i's are the Lagrange coefficients for recovering $Q(0)$ from $\{Q(i) : i \in \mathsf{QUAL}_2\}$ for degree-d polynomials Q. As usual, denoting $\mathbf{F}' = \sum_{i \in \mathsf{QUAL}_2} \lambda_i \mathbf{F}_i$, the shares of shareholders in HOLD satisfy $\sigma'_j = \mathbf{F}'(j)$, and also

$$\mathbf{F}'(0) = \sum_{i \in \mathsf{QUAL}_2} \lambda_i \mathbf{F}_i(0) = \sum_{i \in \mathsf{QUAL}_2} \lambda_i \mathbf{F}(i) = \mathbf{F}(0).$$

Moreover, since all the $\mathbf{F}_i$'s satisfy $\mathbf{F}_i(0) = \mathbf{F}_i(-1) = \cdots = \mathbf{F}_i(1-a)$, then so does $\mathbf{F}'$.

2.8 Sub-sampling the Committees

One of the main use cases for our protocol is an open system (such as a public blockchain), which could be very large. In this use case, the committees in each epoch must be sub-sampled from the entire population, and be large enough to ensure a sufficiently large honest majority with overwhelming probability.

One way of implementing this sub-sampling would be to use verifiable random functions (VRFs), but this would result in rather loose tail bounds and large committees. We can get smaller committees by having the committees implement also a randomness beacon, outputting a (pseudo)random value that the adversary cannot influence at the end of each run of the protocol. At the

[12] As described here, the protocol only works for resharing a packed vector of the form $(s, s, \ldots, s)$. But it is not very hard to extend it to reshare arbitrary packed vectors (using somewhat higher-degree polynomials), see the full version [4, Appendix I].

[13] Recall that in the dynamic setting we use an agreement protocol that provides stronger guarantees about the size of QUAL, than in the static setting. Namely $|\mathsf{QUAL}| \geq d_1$ instead of just $|\mathsf{QUAL}| + |\mathsf{BAD}| \geq d_1$. See Sect. 2.2.

beginning of the $T+1$'st protocol, everyone therefore knows the value U_T that was produced by the beacon in the T'th run. Members of the $T+1$'st committee determine the members of the $T+2$'nd committee by applying a PRG on U_T.

To see why this helps, note that when the total population is very large, the number of honest parties in a committee chosen by VRFs is approximated by a Poisson random variable with parameter $\lambda = (1-f)n$, where f is the fraction of faulty parties in the overall population (and n is the expected committee size). On the other hand, the number of honest parties in a committee when using the randomness beacon follows a Binomial distribution with parameters $n, p = 1-f$. The Binomial turns out to be much more concentrated than the Poisson, hence the number of honest parties is much closer to $(1-f)n$ with the beacon than with the VRF.

Implementing the randomness beacon for our protocol turns out to be very easy. Since the T'th committee held a sharing of the long-term secret scalar s, they could locally compute a "sharing in the exponent" of $s \cdot \mathsf{Hash}'(T)$ (with Hash' hashing into the group). Namely, everyone computes the group element $E = \mathsf{Hash}'(T)$, then each dealer D_i in the T'th committee with share σ_i can compute and broadcast $U_{T,i} = \sigma_i \cdot E$, together with a Fiat-Shamir zero-knowledge proof that $U_{T,i}$ is consistent with the (public) Feldman commitment of σ_i (which is a proof of equality of discrete logarithms).[14] Once the qualified set QUAL_2 is determined, everyone can interpolate "in the exponent" and compute $U_T = \sum_{i \in \mathsf{QUAL}'} \lambda_i \cdot U_{T,i} = s \cdot E$, where the λ_i's are the Lagrange interpolation coefficients. The group element U_T is the next output of the beacon. Note that the adversary has no influence over the U_T's, they are always set as $U_T = s \cdot \mathsf{Hash}'(T)$. On the other hand, before the shares $U_{T,i}$ are broadcast, the value U_T is unpredictable (indeed pseudorandom) from the adversary's point of view.

2.9 More Optimizations

While quite efficient as-is, in many settings there are additional optimizations that can significantly improve the performance of our protocols, such as committing to evaluation points (rather than coefficients) and using optimistic parameters with safe fallback when sub-sampling committees (See details in [4, Appendix A]). Also, in the full version [4, Appendix H] we discuss the dishonest majority case for a mixed malicious/semi-honest adversary model.

2.10 Parameters and Performance

Various parameters and performance analysis are provided in the full version [4, Appendix C], here we give a very brief overview.

To get enough honest parties in HOLD, we need to have $n \geq 3t + 2a - 1$, and we often assume that this holds with equality. Then we set $d_1 = |\mathsf{QUAL}| = n - t$

[14] More precisely, there is a public commitment $\hat{\mathbf{F}}$ of $\mathbf{F}$ from which anyone can derive a Feldman commitment $\sigma_i \cdot G$ of σ_i. See Sect. 2.1.

and get $b = n - 2t$, hence we can get as many as $ab = a(n - 2t)$ signatures for each run of the protocol. Some example numbers are $n = 10, t = 2, a = 2, b = 6$ (12 signatures per run), or $n = 64, t = 15, a = 10, b = 34$ (340 signatures per run). In the setting of a large open system where committees are sub-sampled, we can even sign more messages in each run without reducing resiliency: for example, assuming 80% honest parties, we can sub-sample a committee of size $n = 992$ with $t = 336, a = 40, b = 320$, and sign 12880 messages in each run.[15]

If we set $t = a = n/5$, we can sign $3n^2/25$ messages per run, with an amortized bandwidth of fewer than 35 scalars/group elements broadcasted per signature. For the sub-sampling parameters above with $n = 992$ (with a group of size $\approx 2^{256}$), the total broadcast bandwidth is only under 100MB.

Given parameters n, t, a, the parties broadcast less than $4n^2$ scalars and group elements (in total). If we set $t = a = n/5$, we can sign $3n^2/25$ messages per run, with an amortized bandwidth of fewer than 35 scalars/group elements broadcasted per signature. For the sub-sampling parameters above with $n = 992$ (with a group of size $\approx 2^{256}$), the total broadcast bandwidth is only under 100MB.

In terms of computation, the most expensive part is multiplying the super-invertible matrix in the exponent (which is needed to compute the public R's). This part takes at most $at(n - 2t)$ products (using a naive algorithm), which is t scalar-elements multiplications per signature. But as we explain in Sect. 2.4, we can use much more efficient matrix-multiplication to reduce it, or just use small scalars. With the small-scalar Vandermonde optimization from above, the computation is about 1 min.

Since the super-invertible matrix multiplication is the most expensive part of the protocol, we wrote code to benchmark actual performances for both our possible optimizations from Sect. 2.4. In the full version [4, Appendix B.3], we show the results for various values of b. For $b = t = 256$, our first solution provides a 29× speed up compared to the naive solution and only takes 682ms when $a = 1$ (on a single core of a 2.20GHZ AMD EPIC 7601 CPU). Even with $a = 64$, the total super-invertible matrix multiplication would take less than 1 min on a single-core. In addition, this operation is trivially parallelizable, computations for each of the a packed values are completely independent of each other and can be run on different threads.

For $b = t = 512$, our second solution becomes faster and provides a 28× speed up compared to the naive solution. It only takes 2.80 s to compute the super-invertible matrix multiplication in that setting for $a = 1$, on a single core.

[15] We need $n \geq 657$ to get (statistical) safety failure $< 2^{-80}$ (and liveness failure $< 2^{-11}$), without packing (i.e., $a = 1$). Setting $a = 40$ only requires $n \geq 992$ while multiplying the number of messages that can be signed by 40 and while providing the same safety guarantees. This is because we have less than $(n - 1)/3$ corrupted parties selected in each committee with overwhelming probability. See details in the full version [4, Appendix D.1].

3 The SPRINT Protocols

3.1 Static-Committee Setting

We begin with our base protocol shown in Fig. 1, namely, a robust threshold Schnorr signature scheme for the static-committee case where the set of parties is fixed. It follows the design and rationale presented in Sect. 2 (particularly, till Sect. 2.6), resulting in a two-round ephemeral randomness generation phase (dependent on the number of messages to be signed but not on the messages themselves) followed by a *non-interactive* signing procedure. It considers n parties of which at most t are corrupted, and is given a packing parameter a and an amplification (via a super-invertible matrix) parameter b. It assumes an asynchronous broadcast channel. The protocol consists of three parts. An initial setup stage where parties obtain shares σ_i of a long-term secret key s, and corresponding public key $S = s \cdot G$, and $S_i = \sigma_i \cdot G$ are made public. We assume that sharing the secret key uses packed secret sharing, namely, the parties' shares σ_i lie on a polynomial $\mathbf{F}$ of degree $d = t + a - 1$, such that $\mathbf{F}(0) = \mathbf{F}(-1) = \ldots = \mathbf{F}(-a+1) = s$. This initial setup can be done via a distributed key generation (DKG) protocol or another secure procedure.

The second part is the *generation of ephemeral randomness for Schnorr signatures.* Following the DKG blueprint of [16,34], each party P_i shares a random polynomial $\mathbf{H}_i$ by transmitting the value $\mathbf{H}_i(j)$ to each other party P_j and committing to $\mathbf{H}_i(\cdot)$ over a public broadcast channel. Our application allows for the use of the more efficient Feldman secret sharing [12]. In our case, parties commit to their polynomials $\mathbf{H}$ by broadcasting values $\mathbf{H}(v) \cdot G$ for $d' + 1$ different evaluation points v where d' is the degree of $\mathbf{H}$ (specifically, in our case, this set is defined as the interval $[-a+1, t+a-1]$).

A central part of such a protocol is for the parties to agree on sets of dealers (denoted $\mathsf{QUAL}, \mathsf{BAD}$) that shared their polynomials correctly/badly, and a large enough set of parties (denoted HOLD) that received correct sharings from all parties in QUAL. In Sect. 4.1 we describe an implementation of such a protocol over an asynchronous atomic broadcast channel.

The source of efficiency for SPRINT is the use of packing to share a secrets at a little more cost than sharing just one and attaining further amplification, by a factor of b, using super-invertible matrices [24] (see Sect. 2.4). Here, b is the number of rows in the super-invertible matrix Ψ, e.g., a Vandermonde matrix, and is set to its largest possible value (as analysis shows), $b = |\mathsf{QUAL}| - (t - |\mathsf{BAD}|)$. (Smaller values of b can be used too, if fewer messages need to be signed.) Once the randomness generation procedure is completed, each party in HOLD generates (non-interactively) signature shares consisting of a point on a polynomial $\mathbf{Y}$ that when reconstructed (via interpolation of $d' + 1$ signature shares) can be evaluated on a points to achieve a signatures. Remarkably, using super-invertible matrices one can generate b different polynomials $\mathbf{Y}$, hence resulting in $a \cdot b$ signatures at the cost of a single execution of the (interactive) randomness generation procedure.

Parameters: Integers $n, t, a \geq 1, d = t + a - 1, d' = t + 2a - 2$.

Setup: (Parties: $P_1, \ldots, P_n$)

- Each P_i holds a share $\sigma_i = \mathbf{F}(i)$, where $\mathbf{F}$ is a random degree-d polynomial subject to $\mathbf{F}(0) = \mathbf{F}(-1) = \ldots = \mathbf{F}(-a+1)$. Denote $s = \mathbf{F}(0)$.
- Public keys $S = s \cdot G$ and $S_i = \sigma_i \cdot G$ are publicly known.

Ephemeral randomness generation

1. Each $P_i, i \in [n]$, chooses a random degree-d' polynomial $\mathbf{H}_i$; it broadcasts Feldman commitments to $\mathbf{H}_i$ of the form $\hat{\mathbf{H}}_i(v) = \mathbf{H}_i(v) \cdot G$ for $v \in [-a+1, t+a-1]$. Encrypt the share $\rho_{ij} = \mathbf{H}_i(j)$ under the public key of P_j for all $j \in [n]$, and boardcast all the resulting ciphertexts.
2. $P_1, \ldots, P_n$ run the protocol from Fig. 3 to agree on $\mathsf{QUAL}, \mathsf{BAD}, \mathsf{HOLD} \subseteq \{P_1, \ldots, P_n\}$ with $d_0 = |\mathsf{HOLD}| = n - t$, $d_1 = |\mathsf{QUAL}| + |\mathsf{BAD}| = n - t$, and every $P_j \in \mathsf{HOLD}$ holds valid shares from all the dealers in QUAL.
3. Set $b = |\mathsf{QUAL}| - (t - |\mathsf{BAD}|)$; $\Psi = [\psi_i^u] \in \mathbb{Z}_p^{b \times |\mathsf{QUAL}|}$ a super-invertible matrix. For $u \in [b]$, $v \in [a]$, define $\mathbf{H}^u(\cdot) = \sum_{i \in \mathsf{QUAL}} \psi_i^u \mathbf{H}_i(\cdot)$, $r^{u,v} = \mathbf{H}^u(1-v)$, $R^{u,v} = r^{u,v} \cdot G$.[a]
   Each $P_j \in \mathsf{HOLD}$ sets $\rho_j^u = \mathbf{H}^u(j) = \sum_{i \in \mathsf{QUAL}} \psi_i^u \rho_{ij}$ for all $u \in [b]$.

Signature share generation On input messages $M^{u,v}, u \in [b], v \in [a]$:

Each $P_j \in \mathsf{HOLD}$ sets $\delta = \mathsf{Hash}\big(S, \mathsf{QUAL}, \{(R^{u,v}, M^{u,v}) : u \in [b], v \in [a]\}\big)$ and $\Delta = \delta \cdot G$.

Then, it runs the following procedure, in parallel, for each $u \in [b]$:

1. Computes $e^{u,v} = \mathsf{Hash}(S, \Delta + R^{u,v}, M^{u,v})$ for $v \in [a]$;
2. Computes the degree-$(a-1)$ polynomial $\mathbf{Z}^u$, with $\mathbf{Z}^u(1-v) = e^{u,v}$ for $v \in [a]$.
3. Outputs *signature share*: $\pi_j^u = \mathbf{Z}^u(j) \cdot \sigma_j + \rho_j^u$.

   Note: $\pi_j^u = \mathbf{Y}^u(j)$ for the degree-d' polynomial $\mathbf{Y}^u = \mathbf{Z}^u \cdot \mathbf{F} + \mathbf{H}^u$

Schnorr signature assembly (from signature shares)

For each issued signature share π_j^u verify, using commitments to $\mathbf{H}_i, i \in \mathsf{QUAL}$, and public key $S_j = \mathbf{F}(j) \cdot G$, that $\pi_j^u \cdot G = \mathbf{Z}^u(j) \cdot S_j + \mathbf{H}^u(j) \cdot G$.

When collecting $d'+1$ verified shares π_j^u, reconstruct the polynomial $\mathbf{Y}^u$ and for all $v \in [a]$ set $\phi^{u,v} = \mathbf{Y}^u(1-v)$. (Note: $\phi^{u,v} = \mathbf{Y}^u(1-v) = \mathbf{Z}^u(1-v) \cdot \mathbf{F}(1-v) + \mathbf{H}^u(v) = e^{u,v} \cdot s + r^{u,v}$.)

For $v \in [a], u \in [b]$, output the Schnorr signatures $(\Delta + R^{u,v}, \delta + \phi^{u,v})$ on message $M^{u,v}$.

[a] The values $R^{u,v}$ can be computed from commitments to the polynomials $\mathbf{H}_i$, hence public information.

Fig. 1. SPRINT Scheme in the Static-Committee Setting

In all, we have that after the randomness generation procedure, parties generate their shares of the signatures without any further interaction. Each party P_j computes locally their signature shares $\pi_j^u, u \in [b]$ and publishes them. Reconstructing the signature for each batch of a messages $M^{u1}, \ldots M^{ua}$ can be done by interpolation from any $d'+1$ correct signature shares π_j^u. Moreover, signature shares can be verified individually by a Schnorr-like validation $\pi_j^u \cdot G = \mathbf{Z}^u(j) \cdot S_j + \rho_j^u \cdot G$, where all the required information is public. Thus, invalid signature shares can be discarded.

An additional ingredient in the protocol is the use of the "mitigation value" $\delta = \mathsf{Hash}(S, \mathsf{QUAL}, \{(R^{u,v}, M^{u,v}) : u \in [b], v \in [a]\})$ needed to achieve security when running the $a \cdot b$ signatures in parallel, as explained in Sect. 2.3.

We prove the security of the SPRINT protocol in Fig. 1 in the full version [4, Appendix G].

3.2 The Dynamic/Proactive Setting

The adaptation of SPRINT to the dynamic setting is shown in Fig. 2. See also Sect. 2.7. It requires two types of sharings. One is ephemeral randomness generation as in the static setting, where dealers have no input, and they just share random polynomials. The other is a share refresh (i.e., proactive resharing), in which the dealers have shares of the long-term secret, and they refresh the sharing of that secret to the shareholders. These two sharings are enabled by (almost) the same DKG-like protocol, both using the agreement protocol (see details in [4]) with the same set HOLD and two QUAL sets for the two sharings. Note that the use of the same set HOLD for both sharings is crucial to guarantee that enough parties (those in HOLD) have *both* shares of the secret s and of the ephemeral randomness as needed for generating signatures. Proving security of this protocol is very similar to the static case; see more details in our full version [4, Appendix G.7].

A Note on the "Traditional" Proactive Setting. The proactive setting [22,23,31] was originally envisioned as a periodic operation, say every week, in order to heal the system from active and passive corruptions. When running SPRINT in such a scenario, one would not want to perform a share refresh with each run of the signature generation protocol (Fig. 1) but only at the end of a full proactive period. However, by decoupling the two sharings (refresh and randomness generation), we lose the ability to use the same set HOLD for both cases. This raises a liveness issue: If the share refresh ends with a set HOLD of size $n-t$ and a subsequent execution of SPRINT ends with a *different* set HOLD' of the same size, then it may be the case that the intersection of these two sets will have less than $t+1$ uncorrupted parties, hence unable to create signatures.

However, the traditional proactive setting already assumes the share refresh to happen within a more controlled environment.[16] Thus, it makes sense to con-

[16] E.g., it assumes human intervention to replace or reboot servers, to export public keys from new servers or servers that choose new (encryption) keys, etc. (see [23]).

Parameters: Integers $n, t, a \geq 1, d = t + a - 1, d' = t + 2a - 2$.

Parties: Dealers $D_1, \ldots, D_n$, shareholders $P_1, \ldots, P_n$

Setup: (D_i's)

- Each D_i holds a share $\sigma_i = \mathbf{F}(i)$, where $\mathbf{F}$ is a random degree-d polynomial subject to $\mathbf{F}(0) = \mathbf{F}(-1) = \ldots = \mathbf{F}(-a+1)$. Denote $s = \mathbf{F}(0)$.
- Public keys S and $S_i = \sigma_i \cdot G$ are publicly known.

Ephemeral randomness generation and Re-sharing (The D_i's and P_j's)

1. Each $D_i, i \in [n]$, with share $\sigma_i = \mathbf{F}(i)$ chooses:
   - A random degree-d' polynomial $\mathbf{H}_i$;
   - A degree-d polynomial $\mathbf{F}_i$, random subject to $\mathbf{F}_i(0) = \cdots = \mathbf{F}_i(1-a) = \sigma_i$.

   D_i broadcasts Feldman commitments to $\mathbf{F}_i, \mathbf{H}_i$;
   D_i encrypts $\rho_{ij} = \mathbf{H}_i(j)$ and $\sigma_{ij} = \mathbf{F}_i(j)$ under P_j's key $\forall j \in [n]$, and broadcasts all these ciphertexts.
2. $P_1, \ldots, P_n$ run the agreement protocol ([4, Appendix E]) to agree on $\mathsf{HOLD} \subseteq \{P_1, \ldots, P_n\}$, $\mathsf{QUAL}_1, \mathsf{QUAL}_2 \subseteq \{D_1, \ldots, D_n\}$ with $d_0 = |\mathsf{HOLD}| = n - t$, $d_1 = |\mathsf{QUAL}_1| = n - t$, $d_2 = |\mathsf{QUAL}_2| = t + a$, where every $P_j \in \mathsf{HOLD}$ received valid shares ρ_{ij} from all the dealers in QUAL_1 and valid shares σ_{ij} from all the dealers in QUAL_2.[a]
3. Set $b = |\mathsf{QUAL}_1| - t$; $\Psi = [\psi_i^u] \in \mathbb{Z}_p^{b \times |\mathsf{QUAL}_1|}$ a super-invertible matrix.
   For $u \in [b]$, $v \in [a]$, define $\mathbf{H}^u(\cdot) = \sum_{i \in \mathsf{QUAL}_1} \psi_i^u \mathbf{H}_i(\cdot)$, $r^{u,v} = \mathbf{H}^u(1-v)$, $R^{u,v} = r^{u,v} \cdot G$.
   Each $P_j \in \mathsf{HOLD}$ sets $\rho_j^u = \sum_{i \in \mathsf{QUAL}_1} \psi_i^u \rho_{ij}$ for all $u \in [b]$.
4. Each $P_j \in \mathsf{HOLD}$ sets $\sigma'_j = \sum_{i \in \mathsf{QUAL}_2} \lambda_i \mathbf{F}_i(j)$, λ_i's are the Lagrange coefficients.
   Let $\mathbf{F}' = \sum_{i \in \mathsf{QUAL}_2} \lambda_i \mathbf{F}_i$; a commitment to $\mathbf{F}'$ is obtained from those of the $\mathbf{F}'_i$'s.

Signature generation and assembly Same as in the static case in Fig. 1 but using polynomial $\mathbf{F}'$ instead of $\mathbf{F}$ in that figure.

[a] Valid σ_{ij} mean in particular that $\mathbf{F}'_i$ indeed has the required format, with $\mathbf{F}_i(0) = \cdots = \mathbf{F}_i(1-a) = \sigma_i = F(i)$.

Fig. 2. SPRINT Scheme in the Dynamic-Committee Setting

sider a more synchronous setting (with monitored and resolved delays) during refresh in which case the share refresh operation can be assumed to be completed after a defined amount of time for non-adversarial servers. In this case, parties that did not make it to HOLD by that time will be disqualified from participating in signature generation until the next proactive execution and be counted towards the bound t on corrupted parties. This guarantees that all honest parties in sets HOLD created by runs of SPRINT until the next refresh period will have valid shares of the secret s.

4 The Agreement Protocol

For agreement, we observe that in the static setting we can have a more efficient agreement protocol than in the proactive/dynamic setting. As a result, we

present two protocols of a very similar flavor for the task of reaching agreement. In this section we describe in detail the base agreement protocol, which achieves the best results for the static setting. Then we sketch the enhancements that we need for the dynamic/proactive setting in what we refer to as the full protocol, which is described in the full version [4, Appendix E].

This protocol is designed to work over an asynchronous total-order (aka atomic) broadcast channel. Recall that a total-order broadcast channel provides the following guarantees:

- *Eventual delivery.* A message broadcasted by an honest party will eventually be seen (unmodified) by all honest parties. However, the adversary can change the order in which messages are delivered to the broadcast channel.
- *Prefix consistency.* Considering the views of the broadcast channel at a given time by two different honest parties, the view of one is a prefix of the other.
- *Authenticity.* Messages that are received on behalf of honest parties were indeed sent by those honest parties.

We also assume a PKI, i.e., each party has an encryption public key that is known to all other parties. The protocol below uses only the broadcast channel for communication, private messages are sent by encrypting them and broadcasting the ciphertext.

Time and Steps. While a total-order broadcast channel is not synchronous, and thus it has no absolute notion of time, we are still ensured that the parties all see the same messages in the same order. We can therefore define a "step T" as the time when the T'th message is delivered. Even though different parties may see it at different times, they will all agree on the message that was delivered at step T. If we have a protocol action that is based only on the messages that appeared on the broadcast channel up to (and including) the T'th message, we are ensured that all the honest parties will take the same action, and they will all know that they did it at "step T".

In the description below we distinguish between dealers and shareholders. The protocol begins with the dealers broadcasting messages, then the shareholders engage in a protocol among themselves based on the dealer messages that they see on the channel. For every dealer message and every shareholder, the shareholder either accepts this message or complains about it.

An important technique in our protocol is the use of "verifiable complaints": This is a complaint by a shareholder about a dealer, that will be accepted by all other honest shareholders. (In our context, it will be implemented by proving that the message sent by that dealer is invalid.) We say that a dealer message is "locally bad" for shareholder P_j, if that shareholder is able to generate a verifiable complaint against it. Importantly, we assume that it is impossible to produce a verifiable complaint against messages sent by honest dealers.

We denote the number of dealers as n_1, at least d_1 of them are assumed to be honest. The protocol is run among a set of n_0 shareholders, at least d_0 of which are assumed to be honest. We require that this base protocol terminates,

Parameters: n_0, d_0, n_1, d_1 (should agree on $|\mathsf{HOLD}| \geq d_0$, $|\mathsf{QUAL}| + |\mathsf{BAD}| \geq d_1$).

Precondition: We have up to n_1 dealers, at least d_1 of which are honest. We also have n_0 shareholders, at least d_0 of them are honest.

Shareholder P_j:

Initialize $\mathsf{HOLD} = \mathsf{QUAL} = \mathsf{BAD} = \emptyset$

1. **Enlarge** QUAL. While $|\mathsf{QUAL}| < d_1$, when receiving the (first) broadcast message of the right format[a] from dealer D_i, set $\mathsf{QUAL} := \mathsf{QUAL} \cup \{D_i\}$.
2. **Broadcast Complaints.** Once $|\mathsf{QUAL}| \geq d_1$, broadcast all the verifiable complaints against dealers in QUAL whose message was locally bad, in a single broadcast message. If this set is empty, broadcast the empty set.
3. **Contract and fix** QUAL, BAD. Collect all the valid complaint-sets (i.e., the ones whose complaints can be verified, or the empty set). Once there are d_0 valid complaint-sets, set $\mathsf{QUAL} := \mathsf{QUAL} \setminus \{D_i\}$ and $\mathsf{BAD} := \mathsf{BAD} \cup \{D_i\}$ for each verifiable complaint against dealer D_i;
4. **Fix** HOLD. To the first d_0 shareholders who broadcasted a valid complaint-set.

[a] In our context, a message has the right format if it contains all the commitments and ciphertexts that it was supposed to have.

Fig. 3. Base protocol for agreeing on QUAL, BAD, HOLD

and that all honest shareholders output the same sets HOLD, QUAL, BAD, where HOLD is a subset of the shareholder set with $|\mathsf{HOLD}| \geq d_0$, and QUAL, BAD are disjoint subsets of the dealer set with $|\mathsf{QUAL}| + |\mathsf{BAD}| \geq d_1$.

The base protocol is described in Fig. 3 and proven in Theorem 4.1 (details in [4, Appendix F]). Here each shareholder initially sets QUAL to the first d_1 dealers whose broadcast message they receive. Then each shareholder broadcasts a message specifying which of these d_1 dealers sent correct shares and complaining about the ones that did not. Thereafter, each shareholder continuously adds to HOLD the shareholders whose message appeared on the channel, and moves dealers from QUAL to BAD when they see a verifiable complaint against them on the channel. The protocol terminates once HOLD reaches size d_0.

Theorem 4.1. *Consider an execution of the base agreement protocol from Fig. 3 over a total-order broadcast channel, among a set of n_0 shareholders of which at least d_0 are honest. Assume that at most n_1 dealers broadcast messages, at least d_1 of these dealers are honest, and no verifiable complaint can be constructed against any honest dealer. Then all honest shareholders will eventually terminate, all outputting the same sets with $|\mathsf{HOLD}| \geq d_0$ and $|\mathsf{QUAL}| + |\mathsf{BAD}| \geq d_1$. Moreover:*

- *No shareholder in* HOLD *complained against any dealer in* QUAL*; and*
- *Every dealer in* BAD *has at least one shareholder in* HOLD *that lodged a verifiable complaint against them.*

Dealer D_i (sharing a degreed-d polynomial, $\mathbf{F}_i(X)$):

1. Compute $\hat{\mathbf{F}}_i = \{\mathbf{F}_i(k) \cdot G : k \in [-a+1, \ldots, d-a]\}$;
2. Let $\sigma ij = \mathbf{F}_i(j)$ and $E_{ij} = ENC_{PK_j}(\sigma ij)$ for all $j \in [n]$;
3. Broadcast $\left(\{E_{i1}, \ldots, E_{in}\}, \hat{\mathbf{F}}_i\right)$.

Shareholder P_j**:**

1. Decrypt $\sigma_{ij} = DEC_{SK_j}(E_{ij})$ and verify $\sigma_{ij} \cdot G \stackrel{?}{=} \sum_{k=1-a}^{d-a} \lambda_{i,j,k} \cdot (\mathbf{F}_i(k) \cdot G)$, with $\lambda_{i,j,k}$ the relevant Lagrange coefficients;
2. If verification failed, create a verifiable complaint against E_{ij}, consisting of the decrypted value σ_{ij} and a proof-of-correct-decryption of E_{ij} relative to PK_j.

Fig. 4. Dealer messages and shareholder complaints

4.1 Agreement in SPRINT, the Static Case

To instantiate the base agreement protocol in SPRINT, we need to set the parameters n_0, d_0, n_1, d_1 and specify how the dealer's messages and verifiable complaints are generated and verified.

In our protocols, a dealer's message is just a Shamir sharing of secrets via polynomials. In the static case, we have one pair of QUAL, BAD for the DKG polynomials. We assume a PKI, and the dealers encrypt and broadcast all the shares under the public keys of their intended recipient, and also broadcast Feldman commitments to the polynomials themselves.

There are checks that all shareholders can perform on public information that the dealers broadcast, i.e. verifying that the committed polynomials are of the right degree, and that the dealer's message includes all the ciphertexts that it is supposed to. However, each shareholder is the only one who can check if the share encrypted under their public key is consistent with the committed polynomial.

If the encrypted share is *not* consistent with the committed polynomial, the shareholder will create a verifiable complaint, using the fact that the dealer's message is visible to all. A verifiable complaint from shareholder P_j, denoted π_{ji}, consists of the decrypted value from the ciphertext that D_i sent to P_j, and a proof-of-correct-decryption relative to P_j's public key.[17] Once other parties see the decrypted value they can all verify that the share indeed is not consistent with the committed polynomial.

The dealer messages and shareholder complaints are described in Fig. 4.

Parameters in the Static-committee Setting. In the static-committee setting, each dealer shares a single random polynomial H_i of degree $d' = t + 2a - 2$. To ensure that the resulting random polynomials can be recovered we need at

[17] The proof-of-decryption can be very simple: a proof of equality of discrete logs if using ElGamal encryption for the shares, or showing an inverted RSA ciphertext if using RSA-based encryption.

least $d'+1$ honest parties in HOLD, so we have to set $d_0 \geq t+d'+1 = 2t+2a-1$. But we can set it even bigger, it can be as large as $n-t$ since we know that there are at least as many honest shareholders. (This implies that we need $n-t \geq 2t+2s-1$, namely $n \geq 3t+2a-1$.)

We note that for the DKG protocol, the size of QUAL is unrelated to the degree of the polynomials H_i. The only constraint on it is that to get b output random polynomials we need $|\mathsf{QUAL}| + |\mathsf{BAD}| = d_1 \geq b+t$. To get the best amortized cost, we want to make b as large as possible, which means using as large an initial set $\mathsf{QUAL} \cup \mathsf{BAD}$ as we can get. Every party can serve as a dealer for the DKG protocol, so we have at least $n-t$ honest dealers and can set $d_1 = n-t$ (and therefore $b = n-2t$).

Hence, we run the agreement protocol with parameters $d_0 = d_1 = n-t$. (If we have fewer messages to sign, we can do with a smaller b, which means smaller d_1, any value $d_1 > t$ would work.)

4.2 Agreement in the Dynamic/Proactive Setting

In this setting, dealers share two types of polynomials, random polynomials $\mathbf{H}_i$ of degree $d' = t+2a-2$ for the DKG, and packed re-sharing polynomials $\mathbf{F}_i$ of degree $d = t+a-1$.

Here we must rely on stronger agreement guarantees. For the static case, it was enough to ensure that in a setting with d_1 honest dealers, we will end up with $|\mathsf{QUAL}| + |\mathsf{BAD}| \geq d_1$, this was enough to ensure $d_1 - t$ honest dealers in QUAL (which is the best we can so in the worst case, and is what's needed for the DKG). Now, however, we need to ensure the stronger condition $|\mathsf{QUAL}| \geq d_1$, since this is what's needed for re-sharing the secret.

We therefore augment the agreement by running multiple iterations of the base protocol. In every iteration, we enlarge QUAL until it reaches side d_1, then have one round of complaints and potentially move some more dealers from QUAL to BAD. This is repeated until no more dealers are added to BAD, at which point we have $|\mathsf{QUAL}| \geq d_1$. (Note that at the beginning of each iteration, we always have enough honest dealers whose messages were not yet incorporated in the protocol to reach QUAL of size-d_1 in this iteration.)

Another enhancement to the protocol is that we now have two separate QUAL's (and corresponding two BAD's): one pair $\mathsf{QUAL}_1, \mathsf{BAD}_1$ for the H_i's, and another pair $\mathsf{QUAL}_2, \mathsf{BAD}_2$ for the F_i's. We however only have one shareholder set HOLD (since we need the same shareholders to get both a share of the key and a share of the ephemeral secrets). The protocol is in the full version [4, Appendix E].

Parameters in the Dynamic-committee Setting. Here we have parameters n_0, d_0 for HOLD and n_1, d_1 for $\mathsf{QUAL}, \mathsf{BAD}$ as before (for the H_i's), but in addition also n_2, d_2 for $\mathsf{QUAL}', \mathsf{BAD}'$. For the $\mathbf{H}_i$'s we have the same parameters as above, $n_0 = n_1 = n$ and $d_0 = d_1 = n-t$. For the $\mathbf{F}_i$'s, we need $d+1 = t+a$ dealers in QUAL' in order for shareholders in HOLD to be able to recover their shares, so we set $d_2 = t+a$.

All the dealers in QUAL' must have shares of the long-term secret, so they had to be in HOLD in the previous epoch. Hence, the pool of dealers could be as small as $n_2 = d_0 = n - t$, and t of them could be corrupted, so we cannot set d_2 any larger than $n - 2t$. This implies the constraint $d_2 = n - 2t \geq t + a$ or $n \geq 3t + a$. This constraint is weaker than the constraint $n \geq 3t + 2a - 1$ from above.

Acknowledgements. We thank Victor Shoup for mentioning to us the solution using a Vandermonde matrix for fast multiplication by a super-invertible matrix.

References

1. Abraham, I., Jovanovic, P., Maller, M., Meiklejohn, S., Stern, G.: Bingo: adaptivity and asynchrony in verifiable secret sharing and distributed key generation. In: Handschuh, H., Lysyanskaya, A. (eds.) CRYPTO (2023). https://doi.org/10.1007/978-3-031-38557-5_2
2. Ben-Or, M., Canetti, R., Goldreich, O.: Asynchronous secure computation. In: 25th ACM STOC, pp. 52–61. ACM Press (May 1993). https://doi.org/10.1145/167088.167109
3. Ben-Sasson, E., Carmon, D., Kopparty, S., Levit, D.: Elliptic curve fast Fourier transform (ECFFT) part I: low-degree extension in time $O(n \log n)$ over all finite fields. In: Bansal, N., Nagarajan, V. (eds.) SODA 2023, Florence, Italy, January 22–25, 2023, pp. 700–737. SIAM (2023). https://doi.org/10.1137/1.9781611977554.ch30
4. Benhamouda, F., Halevi, S., Krawczyk, H., Ma, Y., Rabin, T.: SPRINT: high-throughput robust distributed Schnorr signatures. Cryptology ePrint Archive (2023). https://eprint.iacr.org/2023/427
5. Benhamouda, F., Lepoint, T., Loss, J., Orrù, M., Raykova, M.: On the (in)security of ROS. J. Cryptol. **35**(4), 25 (2022). https://doi.org/10.1007/s00145-022-09436-0
6. Borgeaud, W.: ECFFT algorithms on the BN254 base field (2023). https://github.com/wborgeaud/ecfft-bn254
7. Chen, J., Micali, S.: Algorand: a secure and efficient distributed ledger. Theor. Comput. Sci. (2019). https://doi.org/10.1016/j.tcs.2019.02.001
8. Crites, E.C., Komlo, C., Maller, M.: How to prove Schnorr assuming schnorr: security of multi- and threshold signatures. Cryptology ePrint Archive (2021). https://eprint.iacr.org/2021/1375
9. Das, S., Xiang, Z., Kokoris-Kogias, L., Ren, L.: Practical asynchronous high-threshold distributed key generation and distributed polynomial sampling. USENIX Security (2023)
10. Das, S., Yurek, T., Xiang, Z., Miller, A.K., Kokoris-Kogias, L., Ren, L.: Practical asynchronous distributed key generation. In: 2022 IEEE Symposium on Security and Privacy, pp. 2518–2534. IEEE Computer Society Press (May 2022). https://doi.org/10.1109/SP46214.2022.9833584
11. Drijvers, M., Edalatnejad, K., Ford, B., Kiltz, E., Loss, J., Neven, G., Stepanovs, I.: On the security of two-round multi-signatures. In: 2019 IEEE Symposium on Security and Privacy (SP), pp. 1084–1101 (2019). https://doi.org/10.1109/SP.2019.00050

12. Feldman, P.: A practical scheme for non-interactive verifiable secret sharing. In: 28th FOCS, pp. 427–437. IEEE Computer Society Press (Oct 1987). https://doi.org/10.1109/SFCS.1987.4
13. Franklin, M.K., Yung, M.: Communication complexity of secure computation (extended abstract). In: 24th ACM STOC, pp. 699–710. ACM Press (May 1992). https://doi.org/10.1145/129712.129780
14. Ganesh, C., Patra, A.: Optimal extension protocols for byzantine broadcast and agreement. Distributed Comput. **34**(1), 59–77 (2021). https://doi.org/10.1007/s00446-020-00384-1
15. Garillot, F., Kondi, Y., Mohassel, P., Nikolaenko, V.: Threshold Schnorr with stateless deterministic signing from standard assumptions. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12825, pp. 127–156. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84242-0_6
16. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Secure distributed key generation for discrete-log based cryptosystems. J. Cryptol. **20**(1), 51–83 (Jan 2007). https://doi.org/10.1007/s00145-006-0347-3
17. Gennaro, R., Rabin, M.O., Rabin, T.: Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In: Coan, B.A., Afek, Y. (eds.) 17th ACM PODC, pp. 101–111. ACM (Jun / Jul 1998). https://doi.org/10.1145/277697.277716
18. Goyal, V., Polychroniadou, A., Song, Y.: Sharing transformation and dishonest majority MPC with packed secret sharing. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO (2022). https://doi.org/10.1007/978-3-031-15985-5_1
19. Groth, J., Shoup, V.: Design and analysis of a distributed ECDSA signing service. Cryptology ePrint Archive, Report 2022/506 (2022). https://eprint.iacr.org/2022/506
20. Groth, J., Shoup, V.: On the security of ECDSA with additive key derivation and presignatures. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part I. LNCS, vol. 13275, pp. 365–396. Springer, Heidelberg (May / Jun 2022). https://doi.org/10.1007/978-3-031-06944-4_13
21. Groth, J., Shoup, V.: Fast batched asynchronous distributed key generation. Cryptology ePrint Archive (2023). https://eprint.iacr.org/2023/1175
22. Herzberg, A., Jakobsson, M., Jarecki, S., Krawczyk, H., Yung, M.: Proactive public key and signature systems. In: Graveman, R., Janson, P.A., Neuman, C., Gong, L. (eds.) ACM CCS 97, pp. 100–110. ACM Press (Apr 1997). https://doi.org/10.1145/266420.266442
23. Herzberg, A., Jarecki, S., Krawczyk, H., Yung, M.: Proactive secret sharing or: how to cope with perpetual leakage. In: Coppersmith, D. (ed.) CRYPTO 1995. LNCS, vol. 963, pp. 339–352. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-44750-4_27
24. Hirt, M., Nielsen, J.B.: Robust multiparty computation with linear communication complexity. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 463–482. Springer, Heidelberg (2006). https://doi.org/10.1007/11818175_28
25. Joshi, S., Pandey, D., Srinathan, K.: Atssia: asynchronous truly-threshold schnorr signing for inconsistent availability. In: Park, J.H., Seo, S.H. (eds.) Information Security and Cryptology - ICISC 2021, pp. 71–91. Springer International Publishing, Cham (2022)
26. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 177–194. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17373-8_11

27. Komlo, C., Goldberg, I.: FROST: flexible round-optimized schnorr threshold signatures. In: Dunkelman, O., Jacobson, Jr., M.J., O'Flynn, C. (eds.) SAC 2020. LNCS, vol. 12804, pp. 34–65. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-81652-0_2
28. Lindell, Y.: Simple three-round multiparty schnorr signing with full simulatability. Cryptology ePrint Archive, Report 2022/374 (2022). https://eprint.iacr.org/2022/374
29. Nayak, K., Ren, L., Shi, E., Vaidya, N.H., Xiang, Z.: Improved extension protocols for byzantine broadcast and agreement. In: Attiya, H. (ed.) 34th International Symposium on Distributed Computing, DISC 2020, October 12-16, 2020, Virtual Conference. LIPIcs, vol. 179, pp. 28:1–28:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020). https://doi.org/10.4230/LIPIcs.DISC.2020.28
30. Neji, W., Blibech, K., Ben Rajeb, N.: Distributed key generation protocol with a new complaint management strategy. Security Commun. Netw. **9**(17), 4585–4595 (2016). https://doi.org/10.1002/sec.1651
31. Ostrovsky, R., Yung, M.: How to withstand mobile virus attacks (extended abstract). In: Logrippo, L. (ed.) 10th ACM PODC., pp. 51–59. ACM (Aug 1991). https://doi.org/10.1145/112600.112605
32. Patra, A., Choudhary, A., Rangan, C.P.: Efficient statistical asynchronous verifiable secret sharing with optimal resilience. In: Kurosawa, K. (ed.) ICITS 09. LNCS, vol. 5973, pp. 74–92. Springer, Heidelberg (Dec 2010). https://doi.org/10.1007/978-3-642-14496-7_7
33. Pedersen, T.P.: A threshold cryptosystem without a trusted party. In: Davies, D.W. (ed.) EUROCRYPT 1991. LNCS, vol. 547, pp. 522–526. Springer, Heidelberg (1991). https://doi.org/10.1007/3-540-46416-6_47
34. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 129–140. Springer, Heidelberg (1992). https://doi.org/10.1007/3-540-46766-1_9
35. Ruffing, T., Ronge, V., Jin, E., Schneider-Bensch, J., Schröder, D.: ROAST: robust asynchronous schnorr threshold signatures. In: Yin, H., Stavrou, A., Cremers, C., Shi, E. (eds.) ACM CCS 2022, pp. 2551–2564. ACM Press (Nov 2022). https://doi.org/10.1145/3548606.3560583
36. Shoup, V.: The many faces of Schnorr. Cryptology ePrint Archive (2023). https://eprint.iacr.org/2023/1019
37. Stinson, D.R., Strobl, R.: Provably secure distributed Schnorr signatures and a (t, n) threshold scheme for implicit certificates. In: Varadharajan, V., Mu, Y. (eds.) ACISP 2001. LNCS, vol. 2119, pp. 417–434. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-47719-5_33
38. Yurek, T., Luo, L., Fairoze, J., Kate, A., Miller, A.K.: hbACSS: How to robustly share many secrets. In: 29th Annual Network and Distributed System Security Symposium, NDSS 2022, San Diego, California, USA, April 24-28, 2022 (2022)