SPRINT: High-Throughput Robust Distributed Schnorr Signatures

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

[1]Algorand Foundation, USA
[2]University of Pennsylvania, USA

March 23, 2023

Abstract

We describe high-throughput threshold protocols with guaranteed output delivery for generating Schnorr-type signatures. The protocols run a single message-independent interactive ephemeral randomness generation procedure (e.g., DKG) followed by a *non-interactive multi-message* signature generation procedure. The protocols offer significant increase in throughput already for as few as ten parties while remaining highly-efficient for many hundreds of parties with thousands of signatures generated per minute (and over 10,000 in normal optimistic case). These protocols extend seamlessly to the dynamic/proactive setting, where each run of the protocol uses a new committee, and they support sub-sampling the committees from among an effectively unbounded number of nodes. The protocols work over a broadcast channel in both synchronous and asynchronous networks.

The combination of these features makes our protocols a good match for implementing a signature service over an (asynchronous) 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 using broadcast bandwidth of only $O(n^2)$ group elements and scalars. For example, we can sign about $n^2/16$ messages using just under $2n^2$ total bandwidth while supporting resilience against $n/4$ corrupted parties, or sign $n^2/8$ messages using just over $2n^2$ total bandwidth with resilience against $n/5$ corrupted parties.

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

[*] Work done partially while at the Algorand Foundation.

Contents

1 Introduction **4**

1.1 Our Techniques . . . 5

1.2 Prior Work . . . 7

1.3 Organization . . . 8

2 Technical Overview **8**

2.1 Starting Point: The GJKR Protocol . . . 9

2.2 The Agreement Protocol . . . 10

2.3 Signing Many Messages . . . 11

2.4 Using Super-Invertible Matrices . . . 11

2.5 Using Packed Secret Sharing . . . 12

2.6 More Efficient Signing . . . 13

2.7 The Dynamic/Proactive Setting . . . 13

2.8 Sub-sampling the Committees . . . 14

2.9 More Optimizations . . . 15

2.10 Parameters and Performance . . . 17

2.11 Mixed Adversary Model and Dishonest Majority . . . 20

3 The SPRINT Protocols **20**

3.1 Static-Committee Setting . . . 20

3.2 The Dynamic/Proactive Setting . . . 22

4 The Agreement Protocol **22**

4.1 Agreement in SPRINT . . . 26

5 Deploying in a Blockchain Environment **28**

5.1 Committee Sizes . . . 28

5.2 Uses of Additive Key Derivation . . . 30

5.3 Recovery from Catastrophic Failures . . . 30

A Security Proof **33**

A.1 Centralized Schnorr . . . 34

A.2 Threshold Schnorr for a single message . . . 35

A.3 Threshold Schnorr for multiple messages . . . 39

A.4 Threshold Schnorr with a Super-Invertible Matrix . . . 40

A.5 Threshold Schnorr with Packing . . . 40

A.6 Putting it All Together . . . 40

A.7 Security for the Dynamic Setting . . . 44

B Refreshing Packed Secrets **44**

B.1 Method One: Sharing One Polynomial . . . 44

B.2 Method Two: Sharing a Polynomials . . . 45

C Faster Multiplication by a Super-Invertible Matrix **46**

D The Parameter-Finding Utility **46**

D.1 Example Parameters 49

1 Introduction

In this work we describe a suite of protocols that we call SPRINT[1], aimed at generating many Schnorr signatures at a low amortized cost where a single message-independent interactive run of ephemeral randomness generation is followed by a *non-interactive and robust* signature generation procedure for signing multiple messages.

The initial motivation for this work was implementing distributed Schnorr signatures on a public blockchain, where the number of participants could be very large (e.g., in the hundreds). Crucially, such systems require *robust* protocols with guaranteed output delivery, as it is not feasible in such systems to restart the protocol and purge misbehaving parties after each failure. On the other hand, public blockchains provide tools such as a broadcast channel and PKI, which can simplify the design of higher level protocols. Moreover, the large number of parties makes it reasonable to assume a large honest majority, a significant advantage when building robust protocols. Our initial goal, then, was to design practical robust distributed Schnorr signature protocols that scale efficiently to hundreds of parties, using PKI and a broadcast channel in a realistic asynchronous regime.

To achieve acceptable performance at the large scale that we wanted, we had to combine many techniques, mostly centered around effective use of amortization. The protocols that we describe can produce thousands of signatures in each run, "almost for the price of one". We introduce an efficiency parameter a, such that each run of the protocol produces $a(n-2t)$ signatures. We pay for this with a slightly reduced 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.[2] (Even without giving up any robustness, i.e. in the case of $a=1$ and $n=3t+1$, we can still sign $n-2t=t+1$ messages rather than a single one, with very little increase in complexity.)

The amortization techniques that we use can apply for as few as $n=10$ parties, while the protocols remain feasible with over a thousand parties. Our protocols are useful in the fixed-committee setting where the same set of parties is used over and over again, but extend seamlessly to the dynamic/proactive setting where each run of the protocol is done by a different committee with refreshed shares. They also naturally support huge systems, where committees are periodically sub-sampled from among the overall population of parties and the required secret state is transferred to the selected parties. Finally, the protocols are modular and not sensitive to the details of the communication model. They do not require synchrony and can work over any mechanism that provides certain agreement properties (see more details below).

Asymptotically, our protocols are bandwidth-optimal upto some not-too-large constants: With n parties, they can provide 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. (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. In the other direction, reducing the resilience to $t=n/10$ allow us to increase the efficiency parameter to $a=7n/20$, resulting in broadcast bandwidth of 9 scalars/group-

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

[2] Since our technique apply in the asynchronous setting, they inherently require $n \geq 3t+1$; but see Section 2.11.

elements per signature. (This is just 4.5 times the bandwidth that it takes to communicate the signatures themselves!!) See more details in Section 2.10.

1.1 Our Techniques

Achieving high efficiency requires the use of many ideas and techniques. Below is a list of the main ones (in no particular order), many of which could be useful also in other contexts. See Section 2 for a detailed overview of the entire protocol and the roles that these techniques play.

As background, recall that Schnorr-type signatures work over a group of prime order p with a generator G; a signature on a message M relative to secret key s and public key $S = s \cdot G$, has the form $(R, r + e \cdot s)$, where r is an ephemeral random secret, $R = r \cdot G$, and $e = \mathsf{Hash}(S, R, M)$. A standard way to compute *robust* threshold Schnorr signatures among n parties that share a long-term secret key s is to run a *distributed key generation (DKG)*[3] procedure [10] that produces a value $R = r \cdot G$ where r is a fresh random value secret shared among the parties. 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 then the DKG procedure that has quadratic (in n) cost both in terms of bandwidth and computation. Our approach is to amortize the cost of DKG by producing $O(n^2)$ signatures per DKG run.

Less than complete sharing. Many DKG threshold systems require *complete secret-sharing*, i.e., all honest parties must receive shares of all the relevant secrets. This means that honest parties cannot terminate until they ensure that all other honest parties will eventually learn their shares of all secrets. This requirement often adds significant complexity to the protocol, and is not needed in our setting. All we need is for sufficiently many honest parties to learn their shares so that they can generate signatures, there is no need to ensure that all honest parties get them. (As a technical comment, our use of the underlying broadcast channel obviate the need to find a biclique of dealers and shareholders, which is sometimes needed when giving up complete sharing, see Sections 2.2 and 4.)

Extreme packing. We use a combination of packed secret sharing [8] and super-invertible matrices [15] to share $a(n-2t)$ fresh random scalars in a single DKG run, at the price of reducing the resilience from $t < n/3$ to $t < (n - a + 1)/3$. This is described in Sections 2.4 and 2.5. We also describe some optimizations related to faster multiplication by super-invertible matrices in Section 2.4.1 and Appendix C.

Local SIMD computation. Working with packed secret sharing increases the number of secret shared, but current MPC solutions for using packed secret sharing entail non-trivial protocols, even for simple functions. 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 seem to require interaction, since it includes a product. Furthermore, when using simple Shamir sharing for s, some joint processing is needed to create the multiple signatures.

To do better, we introduce an interesting idea (which to the best of our knowledge has not appeared before) to be able to take full advantage of packing without any interaction: 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

[3] Throughout the paper, we use a DKG protocol for different purposes, including ephemeral Schnorr signature generation, long-term key generator and proactive refreshment; we use the DKG term in all these cases.

(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 Section 2.6.

Refreshing packed secrets. In the dynamic/proactive setting, we need to refresh the sharing of the packed vector $(s, \ldots, s)$. This requires a small generalization to the GRR protocol from [11], see Section 2.7 and Appendix B.

An optimized QUAL agreement protocol with verifiable complaints. A main component in our solutions is the Distributed Key Generation (DKG) protocol from [10] which we use for the generation of shared *ephemeral* randomness for the Schnorr protocol. The protocol crucially relies on an agreement on a set of qualified dealers QUAL (those that followed the protocol correctly). In this work, we describe a simple modular QUAL-agreement protocol over broadcast, with a clean separation between the underlying broadcast channel, the QUAL-agreement above it, and the high-level DKG protocol on top.

One advantage of using an underlying broadcast channel, is that it enables "verifiable complaints" by shareholders, i.e., NIZK proofs that certain dealers sent bad shares. This technique simplifies the agreement protocol and saves one round of broadcasts, see Sections 2.2 and 4 for details.

Optimistic runs with pessimistic fallback. While it is crucial that the system is resilient against upto t corrupted parties, in "normal operations" we often have very few (if any) actual misbehaving parties. In Section 2.9 we describe a few optimizations that take advantage of this characteristic, using smaller parameters and signing many more messages in the optimistic case, and falling back on larger parameters and/or fewer signatures otherwise.

In the best case, the optimistic path can reduce complexity by more than a 3× factor, while at the same time increasing the number of messages signed by almost 6×. Importantly, these optimizations will not violate safety along the optimistic paths, even if the optimistic assumptions that they make are violated. It is only liveness that is (partially) impacted in that case, so the system can fall back on the pessimistic path to complete the protocol, if needed.

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 would be to use for this purpose self-selection via VRFs (as done, e.g., in [4]). However, this results in a 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. We therefore augment the signature protocol to implement also this beacon, which turns out to be almost for free in our case. See Section 2.8 for more details.

Security of distributed parallel Schnorr signatures. Our base protocol is similar (though not quite identical) to the GJKR distributed Schnorr signature protocol from [10], and we then extend and optimize it to sign many messages. However, GJKR-like protocols are known to fail in the concurrent setting where the protocol is run in parallel for multiple messages; specifically, such protocols are open a ROS-type attacks [3]. Our protocols do not address concurrent security in general but allow for multiple messages to be signed in parallel in a *single run* of the protocol. To enable such multi-message security, we use a mitigation technique from prior work (e.g., [17, 13]). As far as we know, that technique was only analyzed in the generic group model for ECDSA signatures [13]. In our case, we show it sufficient for proving the security of our protocols (for signing a given set of messages) by reduction to the hardness of the discrete logarithm problem in the random-oracle model. See Section 2.3 and Appendix A.

Robust threshold signatures. All our protocols provide robustness in a strong sense. They terminate with signatures for all a^2 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 synchronous case, parties output their shares after just two rounds of broadcast.

1.2 Prior Work

Recent years saw a lot of activities trying to improve efficiency of VSS, DKG, and signatures protocols, much of which focused on asynchronous protocols and some emphasizing guaranteed output delivery. Below we focus on some of the more recent works on these subjects.

Threshold Signatures. Komlo and Goldberg described FROST [18], a non-robust threshold Schnorr signature protocol that requires a single-round signing protocol after a single-round pre-processing 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 (but see Section 2.11).

ROAST [24] presents a wrapper technique that can transform concurrent-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; in particular, it involves quadratic (or more) per-signature bit-communication ($O(tn^2+tn\lambda)$, λ a security parameter) compared to $O(\lambda)$ in our case.

Garillot et al. [9] implement a threshold Schnorr signature based on deterministic signing, e.g., EdDSA, in order to avoid potential dangers of randomness reuse. They present a dishonest-majority non-robust scheme using MPC (i.e., garbled circuits) and NIZKs techniques that while optimized for this specific application still results in significant more complexity and cost per signature relative to our solution.

Lindell [19] 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 sharings, 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 [12] recently described a rather efficient ECDSA signing protocol, with emphasis on guaranteed output delivery over asynchronous channels. (The underlying VSS in that work achieves completeness, which is not needed in our case.) They use verifiable complaints in a manner somewhat similar to ours, but do not disqualify dealers upon a verifiable complaint, instead relying on other shareholders in an attempt to still distribute their shares.

Joshi et al. [16] address the lack of concurrent security in the basic threshold Schnorr scheme from [10] 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).[4] Neji et al. [20] design a DKG intended to avoid the need to reconstruct shares of inactive (or slow) shareholders as required in the GJKR [10] solution. However, they do so at a high computational and communication cost. We achieve a similar effect at a much lower cost by avoiding the additive sharing approach of GJKR.

Yurek et al. [26] described a randomness-generation protocol over asynchronous communication channels, in the context of the offline phase of generic secure-MPC. They provide *complete* secret sharing (i.e. requiring that all honest parties have shares on the polynomial), which is needed for their MPC application. As in [12], this work also uses verifiable complaints, yet unlike our paper, they do not disqualify dealers upon a verifiable complaint, instead they complete the set of shares. We can deliver our solution without needing all honest parties to hold valid shares.

Abraham et al. [1] describe Bingo, 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 uses pairings and bivariate polynomial to get completeness (which we don't need and do not provide), and has concrete complexity significantly higher than ours. 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 additional papers deal with the question of Asynchronous DKG. However, they do not relate directly to our paper as the main thrust of their work is reaching agreement in the asynchronous setting, e.g. [5, 6]. In contrast, we assume an underlying broadcast channel, simplifying agreement significantly.

1.3 Organization

The rest of this manuscript is organized as follows: In Section 2 we provide a high-level step-by-step overview of our protocols and the various components that are used in them. In Section 3 we describe in more detail our high=level protocol for the fixed-committee and dynamic settings. In Section 4 we describe and prove our agreement protocol. In Section 5 we discuss issue related to using SPRINT in our original motivating application, i.e., to implement a large-scale signature service over a public blockchain. Security proofs for our protocols and some other details are deferred to the appendices.

2 Technical Overview

For ease of exposition, 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). They first run a distributed key-generation (DKG) protocol to generate a sharing of

[4] We use DKG to refer to distributed key generation for long-term keys as well as for the similar process of generating ephemeral randomness as needed in Schnorr signatures.

ephemeral randomness, then use their shares of the long-term secret and ephemeral randomness to generate Schnorr-type signatures on messages. When used in an actual system, 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/proactive setting these will indeed be different parties, but in the static case they are 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) and parameters (a, b, n, t). We denote the set of integers from x to y (inclusive) by $[x, y]$, and also denote $[x] = [1, x]$. Groups elements are denoted by uppercase English letters (G, S, R, etc.), sometimes with a hat ($\hat{F}, \hat{H}$). Polynomials are denoted by bold Uppercase English letters ($\mathbf{F}, \mathbf{H}, \mathbf{I}, \mathbf{Y}, \mathbf{Z}$).

2.1 Starting Point: The GJKR Protocol

Our starting point is the protocol of Gennaro et al. [10] for distributed key-generation (DKG), and a variation on their use of that protocol for Schnorr signatures. In their DKG protocol, each dealer uses VSS to share a random value to everyone, and then all these sharings are added. Specifically, each dealer D_i shares a random ephemeral secret using a degree-d' polynomial $\mathbf{H}_i$ (for some degree d' that we define later), and commits publicly to this polynomial.

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 sub-shares that they received from these qualified dealers. Namely, each shareholder can add the sub-shares 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 $\mathbf{H}$ (the ephemeral randomness) 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 [10]. 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 depoly 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.

2.1.1 Pedersen vs. Feldman Commitments

It was pointed out by Gennaro et al. [10] that sharing randomness usually requires the dealers to commit to their sharing polynomials using statistically-hiding commitments such as Pedersen's [21]. The less expensive Feldman commitments[5], where dealers commit to each coefficient h_{ij} of their polynomials by broadcasting a single group element $h_{ij} \cdot G$, are open to rushing attacks in general. Luckily, Gennaro et al. prove in [10, Sec 5] that for the purpose of generating the

[5] Strictly speaking, the so called Feldman commitments do not provide the hiding security of regular cryptographic commitments, yet they are colloquially referred to as such.

ephemeral randomness for Schnorr signatures, it is safe to use Feldman commitments, 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 (see Section 2.9).

2.2 The Agreement Protocol

Once the dealers distributed their shares and committed to their polynomials, the shareholders engage in a protocol to agree on a large enough set of "qualified" dealers QUAL and "secret-holding" shareholders HOLD, such that every shareholder in HOLD received valid shares from every dealer in QUAL. This protocol is parameterized by some d_0, d_1 (to be defined later), and it ensures that $|\mathsf{HOLD}| \geq d_0$ and $|\mathsf{QUAL}| \geq d_1$.

In this work we describe an efficient implementation of the QUAL-agreement protocol over a total-order broadcast channel, using a PKI (PKI is used for enhanced performance but not in an essential way). It begins with each dealer D_i broadcasting their shares, 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 sub-shares that are inconsistent with the commitments can broadcast a *verifiable complaint* against a dealer, consisting of a NIZK proof that the dealer has sent them bad shares.

Each shareholder keeps a local candidate for the set QUAL, which is increased with every new dealer message and decreased with verifiable complaints. Specifically, it consists of all the dealers who broadcasted their message, and for which no verifiable complaint was received (so far). If a shareholder receive a dealer message with bad sub-shares, it still adds that dealer to the local QUAL candidate, but at the same time it sent a verifiable complaint against them on the broadcast channel. The dealer will be removed from QUAL once a verifiable complaint appears on the broadcast channel. Note that since the QUAL candidates are determined solely by the messages on the broadcast channel, then all honest shareholders that read up to some point in the channel will agree on their QUAL candidates.

Every so often, a shareholders P_j will broadcast a "support message", indicating that their QUAL candidate is large enough, and that they have no outstanding complaints. (Outstanding complaints are verifiable complaints that P_j sent, but that still did not appear on the broadcast channel.) In more detail, P_j records the last point in time τ in which their QUAL reaches size d_1. This recorded time will remain fixed as long as QUAL remains at size d_1 or larger, and will reset whenever QUAL drops below size d_1. P_j will send a message "support at τ" whenever the following conditions are met:

- τ is set (which means that their QUAL candidate has size $\geq d_1$);
- P_j has no outstanding complaints against dealers in QUAL; and
- P_j never sent a support message with this τ value before.

In other words, P_j will send a "support at τ" message when QUAL grows to size d_1 in step τ, provided that they have no outstanding complaints against any of those dealers. Otherwise, P_j will wait until all these bad dealers are purged from QUAL, and then will send a support message if QUAL is still large enough.

We note again that since the point in which QUAL reaches size d_1 is the same for all honest dealers, then they will all send the same support message, except those that are still waiting on outstanding complaints. Similarly, when τ is reset, they all will rest it at the same point in the broadcast channel.

Finally, P_j counts the number of support messages on the broadcast channel that carry the same τ as the one that it currently has recorded (whether or not P_j itself sent such a support message). Again we note that all the honest parties have the same counter value at the same point in the broadcast channel. If at any time the counter reaches the value d_0, then P_j terminates, outputting the agreed-upon set QUAL at the time of termination. (Note that this may be different from what it was at the point τ.)

Since the set QUAL and the counters are deterministic functions of the content of the broadcast channel, then it is clear that upon termination all honest parties will agree on QUAL. Moreover, once a single honest party terminates, they all do. To see that some honest party must terminate, observe that it must have terminated by the time that all the dealers messages and all the verifiable complaints have arrived. Detailed proof is provided in Theorem 1.

2.3 Signing Many Messages

Our large-scale signature service needs to handle signing many messages in parallel, which brings up a security problem: The proof of security from [10, 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 [3] 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 that was used in [17, 13], 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 $e_v = \mathsf{Hash}(S, R_v, M_v)$ (Hash maps arbitrary strings into $\mathbb{Z}_p$). 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 $\delta = \mathsf{Hash}(S, (M_1, R_1), (M_2, R_2), \ldots)$ and $\Delta = \delta \cdot G$. The signature on M_v is then set as $(R'_v, r'_v + e'_v \cdot s)$, where $r'_v = r_v + \delta$, $R'_v = R_v + \Delta = r'_v \cdot G$ and $e'_v = \mathsf{Hash}(S, R'_v, M_v)$. With this mitigation technique, the reduction only needs to guess the random-oracle query in which δ is computed, recovering the argument from [10, Sec 5] and reducing security to the hardness of computing discrete logarithms in the random-oracle model. See Appendix A.3. We note that this proves the security of 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.

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 super-invertible matrix [15] 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 $\mathbf{H}'_u$. 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 $\mathbf{H}^u = \sum_{i \in \mathsf{QUAL}} \psi_i^u \mathbf{H}_i$ gor 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 log 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 $\delta = \mathsf{Hash}(S, \mathsf{QUAL}, (M_1, R_1), (M_2, R_2), \ldots)$. The reason is that in the asynchronous case we cannot guarantee that all honest parties 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.

2.4.1 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 require that each shareholder multiply their sub-shares by that super-invertible matrix "in the exponent".[6] To make these operations more efficient, we therefore would like to make the matrix as sparse as possible. In Appendix C we prove that any matrix of the form $S = (I|H)$ is super-invertible, with I the $b \times b$ identity matrix and H a $b \times (n-b)$ hyper-invertible matrix [2] (e.g., a Vandermonde sub-matrix). In our protocol, the matrix H would be a $b \times t$ matrix, which means that multiplying by S takes t products per row (rather than $b+t$). Moreover, for our parameter regime, using Strassen algorithm (or maybe even FFT-based techniques) could further reduce the computational load.

2.5 Using Packed Secret Sharing

Similarly to above, we can also assume many honest parties among the set HOLD of secret-holding shareholders, and use packed secret sharing [8] 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 $\mathbf{H}^u(0), \mathbf{H}^u(-1), \ldots, \mathbf{H}^u(-a+1)$. 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.

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

In terms of security, we can reduce security of the packed version in a black-box manner to that of simply running a copies of the non-packed protocol,[7] with the v'th copy encoding its secrets at evaluation point $1-v$. See Theorem 4.

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 requite 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 $(R_v, r_v + e_v \cdot s)$. Our shareholders hold Shamir sharings of the secret key s and the vector $(r_1, r_2, \ldots, r_a)$ of ephemeral secrets. Also, S and the M_v's and 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 $d = t + a - 1$. 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 operation $(e_1, e_2, \ldots, e_a) \odot (s, s, \ldots, s)$ seems like it still requires a nontrivial protocol, even for a known vector of e_v's. But we can eliminate even this little protocol, by assuming an even 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 $(e_1, \ldots, e_a)$. Call this polynomial $\mathbf{Z}$. 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 $(e_1 \cdot s, \ldots, e_a \cdot s)$.

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 $d' + 1 = 2t + 2a - 1$ of these broadcast values, everyone can reconstruct the polynomial and read out all the scalars $\phi_v = r_v + e_v s$ that are needed for these a signatures.

2.7 The Dynamic/Proactive Setting

We are now ready to present the additional components that we need in the dynamic case, 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-in also in the dynamic setting. What is missing is a share-refresh protocol where the dealers can pass to the shareholders also a sharing of the long-term secret s. Here we essentially just use the

[7] All these copies share the same δ in the mitigation technique described in Section 2.3.

GRR protocols of Gennaro et al. from [11], with a minor adaptation since we need to share it in a packed manner.[8]

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 sharing the random degree-d' polynomial $\mathbf{H}_i$, and the shareholders then engage in an agreement protocol to determine QUAL and HOLD for both the $\mathbf{F}$'s and the $\mathbf{H}$'s. Specifically, we run the agreement protocol from Section 2.2 to get $|\mathsf{QUAL}| \geq d' + 1 = t + 2a - 1$ and $|\mathsf{HOLD}| \geq d' + 1 + t = 2t + 2a - 1$.

Once QUAL is set, every shareholder $P_j \in \mathsf{HOLD}$ sets QUAL' to be the first $d+1 = t+a$ dealers in QUAL. Having received $\sigma_{ij} = \mathbf{F}_i(j)$ from each dealer $D_i \in \mathsf{QUAL}'$, P_j then computes their share of the long-term secret as $\sigma'_j = \sum_{i\in\mathsf{QUAL}'} \lambda_i \sigma_{ij}$. The λ_{ij}'s are the Lagrange coefficients for recovering $Q(0)$ from $\{Q(i) : i \in \mathsf{QUAL}'\}$ for degree-d polynomials Q. As usual, denoting $\mathbf{F}' = \sum_{i\in\mathsf{QUAL}'} \lambda_i \mathbf{F}_i$, the shares of shareholders in HOLD satisfy $\sigma'_j = \mathbf{F}'(j)$, and also

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

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 $\mathbf{F}'$.

When used in a proactive system, the system's time is split into epochs, where in each epoch there is a current-epoch committee that holds the long-term secret signing key in shared form, as well as some ephemeral randomness, and utilizes them to sign messages. Between epochs the parties will run an update protocol where the current-epoch committee refreshes the sharing of the global long-term secret key to the next committee, and at the same time also shares to them ephemeral secrets. Then, the next committee uses their shares of the long-term and ephemeral secrets to sign messages as needed. The membership in the current- and next-epoch committees could be determined by some auxiliary mechanism (or they may even remain fixed), or they can be sub-sampled as described below.

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 sufficiently large honest majority whp.

One way of implementing this sub-sampling would be to use VRFs, but this would result is a 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 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 to that value 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

[8] 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 Appendix B.

expected committee size). On the other hand, the number of honest parties in a committee when using the randomness beacon follow 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 sortition. One reason for the tighter concentration is that the variance of the Binomial is $f(1 - f)n$, which is smaller than the variance $(1 - f)n$ of the Poisson.

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 $U_{T,i} = \sigma_i \cdot E$. Once the qualified set QUAL' 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'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 setting there are additional optimizations that can significantly improve the performance. A few are described below.

Committing to evaluation points. Clearly, from a security perspective there is no difference if the commitment to the polynomials $\mathbf{H}_i, \mathbf{F}_i$ is done by committing to their coefficients or to their values at some points (or any mix therefore). But different choices have an effect on the computational complexity of the protocols. In particular, our protocol embed the various secrets at evaluation points $0, -1, -2, \ldots, -a + 1$, so it makes sense to commit to these values instead of the coefficients, this will allow everyone to verify these values without having to compute them every time.

We use the convention that a commitment to a degree-$(d + a)$ polynomial F consists of the $d + a + 1$ group elements $F(v) \cdot G$ for all $v \in [-a + 1, d + 1]$. This way, we directly get $s_u \cdot G$ for all the a embedded secrets s_v, and also $\sigma_i \cdot G$ for the shares σ_i of the first $d + 1$ parties. For the shares of the other $n - d - 1$ parties, $i > d + 1$, obtaining $\sigma_i \cdot G$ requires computing "linear combinations in the exponent" using the appropriate Lagrange coefficients.

Also, in some cases we know that some embedded secrets are equal, so we only need to commit to them once. In particular, our share-refresh protocol uses polynomials F_i such that $F_i(-a + 1) = \cdots = F_i(0) = \sigma_i = F(i)$. Since $\sigma_i \cdot G$ is known ahead of time, there is no need for D_i to send that group element again. Hence, even though F_i is a polynomial of degree $t + a - 1$, the dealer D_i only needs to send t group elements to commit to it.

Optimistic parameters. When sub-sampling the committees from a large population, we need them to be large enough to ensure both safety and liveness with high probability. However, we may set difference confidence levels for safety and liveness. In particular, we require that safety holds except with a negligible probability (statitical security of $1 - 2^{-80}$), but allow liveness to be violated with small but not quite non-negligible probability (99.95% of progress), and re-run the protocol with a new committee in the very rare cases that it failed to make progress. Moreover, it may make sense to start from a relatively weak liveness guarantee, in the hope that the protocol completes, and only switch to larger committees and stronger liveness guarantee if the initial optimistic attempt fails to produce signatures.

For example, our "main parameters" (see Section 2.10) are chosen to ensure 2^{-80} assurance for liveness and 2^{-11} assurance for safety when 80% of the population are honest. But we can start from smaller committee that still gives 2^{-80} assurance for safety at 80% honest, but only (say) 2^{-8} liveness assurance at 95% honest. Namely, it gives the same assurance as before against the adversary learning the secret, but even an adversary controlling only 5% of the parties already has about 1/300 chance of preventing the generation of signatures. The parties can run the protocol with these parameters for a while, and if the signatures are not generated for a while, then timeout and fall back on the more conservative parameters above.

Importantly, in the proactive case, this optimistic approach still ensures the following weak liveness guarantee: the adversary cannot push the system into an unrecoverable state where honest parties do not have enough shares to reconstruct the long-term secret key. Concretely, if the protocol executed with the optimistic parameters succeeds, then at least $d + 1$ honest shareholders managed to reconstruct the new shares of the long-term secret key.

Signing more messages. Our QUAL-agreement protocol for DKG ensures at least $n - t$ qualified dealers, but in "normal operation" we can expect more dealers to behave honestly, perhaps even all of them. In practical deployments we can modify our QUAL-agreement protocol, letting shareholders wait a little for more dealers to send shares, before broadcasting their support messages. (For example, wait a few more blocks if we use a blockchain as our broadcast channel.) This way, if many more dealers are honest and synchronized, we can end up with a larger QUAL set. This in turn will let us sign $a \cdot (|\mathsf{QUAL}| - t) > a \cdot b$ messages in this run. In the best case we could have as many as $|\mathsf{QUAL}| = n$, letting us sign $a(n - t)$ messages.

User aggregation. In setting such as proof-of-stake blockchains, we may view each token as a party for the purpose of our protocols, and therefore a node holding many tokens will have to play the role of many parties. In that case, we can generate, verify, and use the shares of all these parties together. For example, all the dealers that are hosted in that node can encrypt their messages together, rather than generating a separate ciphertext per dealer. Also, if any of the shares sent by that node do not match their commitment, then the node itself will be considered corrupted and all the dealers on it disqualified.

This allows us to aggregate the commitment verification by taking a random linear combination of all the commitments. Checking d' commitment from each of x aggregated dealers to each of y aggregate shareholders can be done using only $O(d'x)$ products instead of the naive $O(d'xy)$, and moreover all but $O(d')$ of them could use small scalars (e.g. 80 or fewer bits rather than 256).

A decryption service. In the blockchain setting, it is not hard to configure the parties running our protocols to also provide a decryption service, not just signatures. For safety, a decryption service should use a different secret key for decryption than for signatures. This can be done with a polynomial of degree one larger, using the techniques from Appendix B.

Once the committee has a Shamir sharing of the secret decryption key, they can directly use it to decrypt ElGamal-type ciphertexts: These ciphertexts include an element $R \in \mathbb{G}$, and to decrypt it is sufficient to recover $s \cdot R$ where s if the decryption key. The parties holding Shamir shares σ_i of s can just broadcast partial decryption shares $\sigma_i \cdot R$ (possibly with some proof of correctness), enabling anyone to compute $s \cdot R$ by "interpolating in the exponent."

2.10 Parameters and Performance

The parameters n, t, a and b. For given parameters t and a, the ephemeral secrets r_v are shared via polynomials of degree $t + 2a - 2$, so we need at least $t + 2a - 1$ honest shareholders to obtain shares in order to be able to use them. All these shareholders must be in HOLD, and there could be upto t corrupted shareholders there, so the agreement protocol must ensure that HOLD is of size at least $2t + 2a - 1$. As there could be t corrupted parties during the run of the agreement protocol, then to ensure $|\mathsf{HOLD}| \geq 2t + 2a - 1$ the number of shareholder must be $n \geq 3t + 2a - 1$ (for both the static-committee setting and the dynamic setting). Below we mostly assume $n = 3t + 2a - 1$.

For the parameter b, recall that we need b honest dealers in QUAL for the DKG protocol. We can ensure QUAL as large as $n - t$ (but of course not any larger), so the largest value that we can ensure for $b = n - 2t$ (i.e., $b = (3t + 2a - 1) - 2t = t + 2a - 1$ if n is as small as possible). The number of signatures in each run of the protocol is therefore $a \cdot b = a(n - 2t)$. We note, however, that at the end of each run of the agreement protocol, everyone knows the size of QUAL, and can set the parameter b accordingly. If QUAL happens to be larger than $n - t$ in a particular run, then b can be larger that $n - 2t$, so we can sign more messages in that run. In the best case, we have $|\mathsf{QUAL}| = n$, so we can sign $a(n - t)$ messages.

Bandwidth in the static-committee setting. In the static setting we do not need to re-share the long term secret key, so each dealer only shares a single random polynomial $\mathbf{H}_i$ of degree $d' = t + 2a - 2$. This means broadcasting $t + 2a - 1$ group elements for the commitments, and n ciphertexts encrypting the shares. Assuming ElGamal encryption (with randomness re-use), encryption takes $n+1$ more group elements. The total per dealer is therefore $n+t+2a$, for a grand total of $n(n + t + 2a)$ group elements.

In the optimistic case where (almost) everyone is synchronized, the agreement protocol requires a single broadcast from each shareholder (after the dealers sent their messages). If all the dealers are good then these messages are short, otherwise each message can include upto t complaints, each with just a few group elements, for an additional bandwidth of $O(nt)$.

For the signature generation part, we have $|\mathsf{HOLD}| = 2t + 2a - 1$ and each shareholder in HOLD broadcasts $b = n - 2t$ shares, for a total broadcast bandwidth of $(2t + 2a - 1)(n - 2t)$ scalars, enabling the generation of upto $a(n - 2t)$ signatures.

The total bandwidth (both group elements and scalars) is therefore $n(n + t + 2a) + (2t + 2a - 1)(n - 2t) = O(n^2)$, plus at most $O(t^2)$ for all the complaints. The amortized bandwidth per signature (not counting complaints) is about

$$\frac{n \cdot (n + t + 2a) + (2t + 2a - 1) \cdot (n - 2t)}{a \cdot (n - 2t)} \quad = \quad \frac{n^2 + nt}{a(n - 2t)} + \frac{2n}{n - 2t} + \frac{2t - 1}{a} + 2.$$

Asymptotically, setting $t = a = \Omega(n)$ we get $O(1)$ group-elements/scalars per signature.

For a few examples, setting $t = a = n/5$ we get about 17.33 scalars/group-elements per signature. To get resilience of $t = n/4$ we can only get $a = n/8$, yielding about 34 scalars/group-elements per signature. In the other direction, reducing the resilience to $t = n/10$ we can set $a = 7n/20$, getting an amortized 9 scalars/group-elements per signature. (That is 4.5 times the bandwidth needed to communicate just the signatures themselves.)

Computation in the static-committee setting. In terms of computation (and counting only scalar-by-element products), each dealer needs to compute $d' = t + 2a - 2$ products to commit to their polynomial and $n + 1$ more for the encryptions.

Each shareholder must verify all the shares that they received, each costing about $d'+1$ products, so $n \cdot (d' + 1)$ products for each shareholder. For complaints, each shareholder needs to generate at most t of them, and verify at most $2t$ of them, since at most t dealers and t shareholders can be bad. (In the static committee case, only t complaints need to be verified.) Verifying each complaint can take another $d' + O(1)$ products (d' to compute the commitment and $O(1)$ for the proof of decryption.) Hence, the agreement can take up to (about) $(n + t)(d' + 1)$ products for each party, regardless of how many messages will be signed.

For generating the signatures, the shareholders must first compute all the R_v's to be hashed, then compute their own shares. By our convention, commitments to the $\mathbf{H}_i$'s include in particular the elements $\mathbf{H}_i(1-v) \cdot G$ for $v \in [a]$. Hence, for any $v \in [a]$, computing the elements $R_{u,v} = \mathbf{H}^u(1-v)$ for all $u \in [b]$ requires multiplying the vector $(\mathbf{H}_i(1-v) : i \in \mathsf{QUAL})$ by the super invertible matrix Ψ of dimension $b \times (b + t)$. Doing it for all $v \in [a]$ means computing the matrix product

$$\left[R_{u,v}\right]_{u\in[b],v\in[a]} \;=\; \Psi \times \left[\mathbf{H}_i(1-v) \cdot G\right]_{i\in\mathsf{QUAL},v\in[a]}.$$

Using the construction from Section 2.4.1, the first b columns in Ψ are the identity, so even using naive matrix multiplication this would take a total of $a \cdot b \cdot t$ products to get all the $a \cdot b$ elements $R_{u,v}$. Using Strassen algorithm (or FFT-based techniques) this can be improved further. Computing the shares involves only field operations.

Once, everyone broadcast their shares, verifying the shares and assembling the signature can be done by anyone who sees the broadcast channel, not necessarily the parties themselves, and certainly not all parties need to carry out that verification. [9] Verifying each share takes $d' + 1$ products, and at most $d' + t + 1$ of them need to be verified before we have $d' + 1$ valid ones, for a total of $(d' + 1)(d' + t + 1)$ products, but since not every party needs to carry it out we do not include these products in the tally below.

Substituting $d' = t + 2a - 2$ and $b = n - 2t$, the overall number of scalar-by-element products for each party is therefore $(n + t + 2a - 1) + (n + t)(t + 2a - 1) + a(n - 2t)t$ products, and the per-signature number is

$$\frac{(n+t+2a-1) + (n+t)(t+2a-1) + a(n-2t)t}{a(n-2t)} \;=\; t + \frac{(n+t)(t+2a)}{a(n-2t)} + \frac{2-\frac{1}{a}}{n-2t}.$$

With $a = \Omega(n)$ and $n-2t = \Omega(n)$, and when using naive matrix multiplicaiton, this yields complexity of $t+O(1)$ products per signature. (In the example of $t = a = n/5$, we get about $t+6$ products for each signature.)

Performance in the dynamic/proactive setting. In this setting we also need to refresh the long-term secret. This means that each dealer broadcasts $n + t$ additional group elements, and

[9] Verification can even be avoided in the optimistic case, by just trying to reconstruct a polynomial and see that (almost) all the shares agree with it.

shareholders may need to broadcast more complaints, but the bandwidth for signature generation remains unchanged. Therefore, the overall bandwidth in this case (not counting complaints) becomes $2n(n+t+a)+(2t+2a-1)(n-2t)$, increasing the per-signature bandwidth by $\frac{n(n+t)}{a(n-2t)}$. (For example, when $t=a=n/5$, the bandwidth increases from 17.33 to 27.33 scalars/group-elements per signature.)

Computation likewise increases much less than 2×. Each dealer performs only t additional products for the commitments.[10] Verifying the sub-shares takes only $n(t+1)$ more products by each shareholder, and checking each complaint is only $t+O(1)$ more products.

Some numerical examples. The techniques in this paper are useful even for fairly small committees. For example:

- With $n=10$, we need $t \leq 3$ even to sign just a single message, but setting for $t=2$ allows us to set $a=2, b=6$ and sign $a \cdot b=12$ messages for the price of a single message. That's a 12× performance improvement for a small drop in resilience.
- For $n=16$ and $t=3$ we can set $a=4, b=10$ and generate 40 signatures.
- For $n=64$ and $t=15$ we can set $a=10, b=34$ and generate 340 signatures.
- In Section 5.1 we analyze the committee sizes that we need when sub-sampling the committees in a few settings. For example, when assuming 80% honest majority in the overall population and shooting for a 2^{-80} probability of safety failure and 2^{-11} probability of liveness error due to sub-sampling, we can use committees of size $n=992$ with $t=336$ and $a=40$ (and $b=320$). We note that here we have $n<3t+1$, since we use different thresholds for liveness and safety errors.

  In this setting, we can sign $40 \cdot 320=12800$ messages per run, and each run consumes broadcast bandwidth of $2n(n+t+a)+(2t+2a-1)(n-2t)=2,954,432$ scalars and group elements, or about 95 megabytes. In terms of computation, each party (playing first a shareholder and then the dealer in the next epoch) needs to perform about $(n+2t+2a-1)+(n+t)(t+2a-1)+n(t+1)+a(n-2t)t=5,187,967$ scalar-element products. Moreover, most of the products are actually dot product between a vector of scalars and a vector of group elements, which can be done perhaps 3× faster than performing each product separately. On contemporary servers, even a single-threaded implementation can perform this number of products in under three minutes, yeilding an amortized rate better than 4000 signatures per minute.
- Running the protocol with optimistic parameters, we may try for a setting that still provides 2^{-80} safety error with 80% honest but liveness-failure probability of 2^{-8} with 95% honest. To get roughly the same number of signatures per run we set $a=64$, thus getting $n=676$, $t=250$, and $b=176$. This yields more or less a 2× lower complexity, with bandwidth of 1,448,832 scalars/group-elements and 3,336,081 scalar-by-element products, while producing 11264 signatures. Moreover, in the even-more-optimistic case where all the dealers happens to be honest, we can sign as many as $a(n-t)=27264$ messages with the same parameters, another 2× improvement (and a rate of more than 10,000 signatures per minute).

[10] Using hybrid encryption does not take any more products to encrypt longer messages.

2.11 Mixed Adversary Model and Dishonest Majority

Focusing on the honest super-majority case (i.e., $n > 3t$) in our analysis is necessary to preserve security of SPRINT in the asynchronous setting, including robustness of signatures and the safety of the global secret across refreshes. Here, we remark on the security of the SPRINT protocol from Figs. 1 and 2 in some cases where $n \leq 3t$ or even with dishonest majority $n < 2t$. Specifically, we consider the *mixed adversary* model of Hirt et al. [14] that distinguishes between malicious parties (that can deviate from the protocol in arbitrary ways) and honest-but-curious ones (that run the protocol as specified but whose internal secrets are available to the attacker). In this model one can get more relaxed bounds on the number of dishonest parties, including security against a dishonest majority (involving both malicious and honest-but-curious parties), while preserving robustness.

Denote by h (a lower bound on) the number of honest parties, and m, c (upper bounds on) the number of malicious and honest-but-curious parties, respectively (here $t = m + c$). We focus on the static case, Fig. 1, as the bounds apply to the dynamic case too. Given these parameters (with $n = h + c + m$) and the packing parameter a, we run the protocol from Fig. 1 using polynomials of degrees $d = m + c + a - 1$ (for the long-term secret) and $d' = m + c + 2a - 2$ (for the ephemeral randomness), and with $d_1 = |\mathsf{QUAL}| = d_0 = |\mathsf{HOLD}| = n - m$ for the agreement-protocol parameters. To be able to sign, we must ensure that $|\mathsf{HOLD}| - m = d_0 - m \geq d' + 1$. Substituting $d' = m + c + 2a - 2$ and $d_0 = n - m$ we get $n - 2m \geq m + c + 2a - 1$, which means $a \leq (n - c - 3m + 1)/2$. For the parameter b, we need b honest parties in QUAL so $b = |\mathsf{QUAL}| - m - c = n - 2m - c$.

Consider the dishonest-majority example $n = 100, h = 49, m = 20, c = 31$, then we can set $a = 5, b = 29$ and we can produce 145 signatures in one run of the protocol with 49% of honest parties. In another example: $n = 16, h = 9, m = 3, c = 4$, we get $a = 2, b = 6$ so 12 signatures. This is just over one quarter of what can be achieved with $n = 16, m = 3$ but security here withstands 4 additional honest-but-curious participants.

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 Section 2 (particularly, till Section 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 keys $S = s \cdot G$ 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 any other secure initialization procedure.

A second component is the *generation of ephemeral randomness for the Schnorr signatures.* Following the DKG blueprint of [22, 10], 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 commitments [7]. In our

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 $\hat{H}_i(v) = \mathbf{H}_i(v) \cdot G$ for $v \in [-a+1, t+a-1]$, and sends $\rho_{ij} = \mathbf{H}_i(j)$ privately[a] to $P_j, \forall j \in [n]$.
2. $P_1, \ldots, P_n$ run an agreement protocol to agree on sets $\mathsf{QUAL}, \mathsf{HOLD} \subseteq \{P_1, \ldots, P_n\}$ with $d_0 = |\mathsf{HOLD}| = n - t$, $d_1 = |\mathsf{QUAL}| = n - t$, and every $P_j \in \mathsf{HOLD}$ holds valid shares from all the dealers in QUAL.[b]
3. Set $b = |\mathsf{QUAL}| - t$; $\Psi = [\psi_i^u] \in \mathbb{Z}_p^{b \times |\mathsf{QUAL}|}$ a super-invertible matrix.

   For $u \in [b]$, $v \in [a]$, define $\mathbf{H}^u(\cdot) = \sum_{i \in \mathsf{QUAL}} \psi_i^u \mathbf{H}_i(\cdot)$, $r^{u,v} = \mathbf{H}^u(1-v)$, $R^{u,v} = r^{u,v} \cdot G$.[c]

   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 shares can be sent via private channels, or encryption over broadcast.

[b] For a specific instantiation of this protocol over broadcast see Section 4.1.

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

Figure 1: SPRINT Scheme in the Static-Committee Setting

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 a large enough set of dealers (denoted QUAL) that shared their polynomials correctly, and a large enough set of parties (denoted HOLD) that received correct sharings from all parties in QUAL. In Section 4.1 we describe an implementation of such a QUAL-agreement protocol over an asynchronous atomic broadcast channel, as needed in our motivating applications. However, SPRINT can be used in other communication settings, possibly with a different agreement protocol.

The source of efficiency for SPRINT is the use of packing to share a secrets at little more cost than sharing just one and attaining further amplification, by a factor of b, using super-invertible matrices [15] (see Section 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), $b = |\mathsf{QUAL}| - t$ (smaller values of b can be used too, if less 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 on $a \cdot b$ signatures at the cost of a single execution of the (interactive) randomness generation procedure.

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 $\pi^u_j, u \in [b]$ and publishes them. Reconstructing the signature for each batch of a message $M^{u1}, \ldots M^{ua}$ can be done by interpolation from any $d' + 1$ correct signature shares π^u_j. Moreover, signature shares can be verified individually by a Schnorr-like validation $\pi^u_j \cdot G = \mathbf{Z}^u(j) \cdot S_j + \rho^u_j \cdot G$, 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 Section 2.3.

Security of the SPRINT protocol from Fig. 1 is proven in Appendix A (cf. Theorem 4).

3.2 The Dynamic/Proactive Setting

The adaptation of SPRINT to the dynamic setting is shown in Fig. 2. See also Section 2.7. It requires two types of sharings. One is signature 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 from Fig. 3 (with the same set HOLD and two QUAL sets for the two sharings.). Proving security of this protocol is very similar to the static case; see some more details in Appendix A.7.

4 The Agreement Protocol

We now turn to our agreement protocol where, as explained in Sections 2.2 and 3, parties need to *agree* on sets QUAL and HOLD that contain enough information to ensure robust generation

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

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

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 $\mathbf{F}(0) = \mathbf{F}(-1) = \dots = \mathbf{F}(-a+1)$. 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 sends $\rho_{ij} = \mathbf{H}_i(j)$ and $\sigma_{ij} = \mathbf{F}_i(j)$ privately to P_j $\forall j \in [n]$.

2. $P_1, \dots, P_n$ run an agreement protocol to agree on $\mathsf{HOLD} \subseteq \{P_1, \dots, P_n\}$, $\mathsf{QUAL}_1, \mathsf{QUAL}_2 \subseteq \{D_1, \dots, D_n\}$ 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 $\sigma_j' = \sum_{i \in \mathsf{QUAL}_2} \lambda_i \mathbf{F}_i(j)$, the λ_i's are the Lagrange coefficients for QUAL_2.

    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 $\mathbf{F}_i(0) = \cdots = \mathbf{F}_i(1-a) = \sigma_i = F(i)$.

Figure 2: SPRINT Scheme in the Dynamic-Committee Setting

of signatures. This protocol is designed to work over an asynchronous total-order (aka atomic) broadcast channel, using a PKI. We start with an abstract description of this protocol in Fig. 3, and then explain in Section 4.1 the specifics of how it is used in the context of our signature protocol (in either the fixed-committee or the dynamic setting).

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 view of the other.
- *Authenticity.* Messages that are received on behalf of honest parties were indeed sent by those honest parties.

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 which 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 th e honest parties will take the same action, and they will all know that they did it at "step T".

PKI. We assume that each party has an encryption public key associated with it known to all parties.

The protocol below uses only the broadcast channel for communication, private messages are sent by encrypting them and broadcasting the ciphertext. In the description below we distinguish between two types of parties, that we call dealers and shareholders (after the role that they play in our protocol). The protocol begins with the dealers broadcasting messages, then the shareholders will 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 it complains about it. The goal of this protocol is to agree on a set of dealers QUAL and a set of shareholders HOLD, such that every shareholder in HOLD accepts all the messages from every dealer in QUAL. (In some cases, the dealers will send more than one type of messages, and the shareholders will output a different QUAL for each type separately, all for the same set HOLD.)

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 a NIZK proof 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.

In its most general form, the protocol can deal with m types of messages, with m some parameter. (In our setting we will only use $m = 1$ and $m = 2$.) Each type k of message is associated with some finite set of n_k dealers that send that type of message, at least d_k of them are assumed honest. The protocol itself is run among a set of n_0 shareholders, at least d_0 of them are assumed honest. We

Parameters: $d_0, d_1, \ldots, d_m$ (should agree on $|\mathsf{HOLD}| \geq d_0$, $|QUAL_i| \geq d_i$).
Note: In our application $m = 2$ for two types: one for sharing of ephemeral randomness and one for share refreshment. Or $m = 1$ when only ephemeral randomness is generated.

Precondition: For $k = 1, \ldots, m$ we have upto n_k dealers, at least d_k of which are honest, broadcasting messages of type k.

Shareholder P_j: (out of n_0 shareholders)
Initialize $\mathsf{HOLD} = \mathsf{QUAL}_1 = \cdots = \mathsf{QUAL}_m = \emptyset, T = 0, \mathsf{sent} = \mathsf{false}$.
Execute until $|\mathsf{HOLD}| \geq d_0$:

Enlarge QUAL_k. When receiving first broadcast message of type k from dealer D_i at some step τ:

1. Set $\mathsf{QUAL}_k := \mathsf{QUAL}_k \cup \{D_i\}$;
2. If $T = 0$ and $|\mathsf{QUAL}_{k'}| \geq d_{k'}$ *for all* k', set $T = \tau$;
3. If the message is locally bad, broadcast a verifiable complaint against it.

Contract QUAL_k. When receiving a verifiable complaint against a type-k message of some dealer D_i:

1. Set $\mathsf{QUAL}_k := \mathsf{QUAL}_k \setminus \{D_i\}$;
2. If $|\mathsf{QUAL}_k| < d_k$ set $\mathsf{HOLD} = \emptyset, T = 0, \mathsf{sent} = \mathsf{false}$.

Broadcast Approval. After processing any message, if $|\mathsf{QUAL}_k| \geq d_k$ for all k and $\mathsf{sent} = \mathsf{false}$, and you didn't send verifiable complaints against any dealer in any of the current QUAL_k's, then broadcast "P_j approves in time T" and set $\mathsf{sent} = \mathsf{true}$.

Enlarge HOLD. When receiving message "$P_{j'}$ approves in time T":
If $T' = T$:

1. Set $\mathsf{HOLD} = \mathsf{HOLD} \cup \{P_{j'}\}$;
2. If $|\mathsf{HOLD}| \geq d_0$ then output HOLD, $\mathsf{QUAL}_1, \ldots, \mathsf{QUAL}_m$ and terminate.

Figure 3: Agreeing on QUAL, Shareholder Actions

require that the protocol terminates, and that all honest shareholders outputs the same sets HOLD, $\mathsf{QUAL}_1, \ldots, \mathsf{QUAL}_m$, with $|\mathsf{HOLD}| \geq d_0$ and $|\mathsf{QUAL}_i| \geq d_i$ for $i = 1, 2, \ldots, m$.

The protocol is described in Fig. 3, and proven in Theorem 1. In the protocol, each shareholder continuously update its candidates for the QUAL_i's by adding dealers whose broadcast message they receive, and removing them if a verifiable complaint is lodged against them. Every so often a shareholder may send a message "approving" some QUAL_i candidates, once they all reach the desired sizes d_i. Once enough shareholders approve, the protocol terminates and the sets HOLD, QUAL_i are determined.

Theorem 1. *Consider an execution of the 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. For all $k = 1, 2, \ldots, m$, assume that at most some n_k type-k messages were sent on the channel, at least d_k of which were sent by honest dealers, and that no verifiable complaint can be constructed against any honest* dealer message. Then all honest shareholders will eventually terminate, all outputting the same sets with $|\mathsf{HOLD}| \geq d_0$ and $|\mathsf{QUAL}_k| \geq d_k$ for all k. Moreover, for all k, no shareholder in HOLD complained against the type-k message of any dealer in QUAL_k.

Proof. Recall that we refer to shareholders that read τ messages from the channel as being "in step τ".

Agreement is easy to verify: The sets QUAL_k that a shareholder maintains are a deterministic function of the messages on the broadcast channel. Hence, at any given step, all shareholders will have the same QUAL_k's. This means that they also agree on the steps in which $|\mathsf{QUAL}_k|$ grows to size d_k or drops below size d_k. Therefore, all the shareholders at a given step must have the same step value for their step variable T. This, in turn, implies that they will all count the same support message from the channel, and hence will have the same set HOLD. It follows that as soon as one honest party terminates at some step τ, all honest parties will terminate at that step with the same HOLD and QUAL_k's. It can be verified by inspection that when that happens, we have $|\mathsf{HOLD}| = d_0$ and $|\mathsf{QUAL}_k| \geq d_k$ for all k. Also, no shareholder ever broadcasts support for QUAL_k that includes dealers that they complain against, so no dealer in HOLD sent a complaint against any type-k message in QUAL_k (for any k).

Proving termination is a little more subtle. We first note that the only case where an honest shareholder P_j with all $|\mathsf{QUAL}_k| \geq d_k$ since step T did not yet broadcast a support for T, is if there are outstanding verifiable complaints that P_j sent but did not yet hit the broadcast channel. Then, assume towards contradiction that the protocol never terminates. Since there are a finite number of dealer messages and shareholders, then due to eventual delivery (and since the protocol never terminates), there will be some step τ at which:

- All dealer messages have arrived;
- All shareholder complaints have arrived; and
- All sent shareholder support messages have arrived.

Since there are at least d_k honest dealer messages for all k, then at step τ all these dealers are part of QUAL_k for every honest shareholder, which means that they all have $|\mathsf{QUAL}_k| \geq d_k$ at step τ.

Let τ^* be the last time before τ where all the QUAL_k's (which are the same for all honest shareholders) grew to size $|\mathsf{QUAL}_k| \geq d + k$. (That is, in step $\tau^* - 1$ one of the QUAL_k's was still too small.) At that step, all honest shareholders set their step variable to $T = \tau^*$, and since the QUAL_k's never dips below d_k since step τ^*, then all shareholder still have $T = \tau^*$ also at step τ.

We conclude that by step τ, all honest shareholders have $|\mathsf{QUAL}_k| \geq d_k$ for all k, and none of them have outstanding complaints. By the observation above, all of them must have already sent a support message with τ^*, and by our choice of τ all these messages have already arrived. Hence, every honest shareholder must have all the other honest shareholders in HOLD. As there $\geq d_0$ of them, it means that every honest shareholder must have $|\mathsf{HOLD}| \geq d_0$, and therefore must have terminated. Contradiction. □

4.1 Agreement in SPRINT

To use the above agreement protocol in the context of SPRINT, we need to set its parameters and specify how the dealer's messages and verifiable complaints are generated and verified.

Dealer D_i (sharing a degreed-d polynomial, $\mathbf{F}_i(X) = \sum_{k=0}^{d} f_{ik} X^k$):

1. Compute $\hat{F}_{ik} = f_{ik} \cdot G$ for $k = 0, \ldots, d$;
2. Let $\sigma ij = \mathbf{F}_i(j)$ and $E_{ij} = ENC_{PK_j}(\sigma ij)$ for all $j \in [n]$;
3. Broadcast $\big(\{E_{i1}, \ldots, E_{in}\}, \{\hat{F}_{i0}, \ldots, \hat{F}_{id}\}\big)$.

Shareholder P_j:

1. Decrypt $\sigma_{ij} = DEC_{SK_j}(E_{ij})$ and verify $\sigma_{ij} \cdot G \overset{?}{=} \sum_{k=0}^{d} j^k \cdot \hat{F}_{ik}$;
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.

Figure 4: Dealer Actions and Shareholder Complaints (in SPRINT the commitments to coefficients are replaced with commitments to polynomial evaluations)

In our protocols, a dealer message is just a Shamir sharing of secret(s) via polynomial(s). Each shared polynomial corresponds to a different type of message and a different QUAL set. (In particular we will have one type for DKG messages and another for share-refresh messages.) 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 polynomial(s) themselves. For simplicity, here we describe these commitments as commitments to the coefficients of the polynomial, but in our protocols, for efficiency reasons, we instead commit to the values of the polynomial at certain points (e.g., a commitment to the value $\mathbf{F}(v)$ will be $\hat{F}(v) = \mathbf{F}(v) \cdot G$). In particular, we use commitments at those evaluation points where the secrets are encoded.

Note that there are checks that all shareholders can perform on public information that the dealer broadcasted. That includes 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, a shareholder is the only one that can check if the share encrypted under their public key is consistent with the committed polynomial.

If the encrypted share *is not consistent* the committed polynomial, the shareholder will create a verifiable complaint. It utilizes 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 and a proof-of-correct-decryption relative to P_j's public key. Once other parties see the decrypted value they can all verify that the share indeed is not consistent with the committed polynomial.

A description of the dealer messages and shareholder complaints are described in Fig. 4. We note that the idea for disqualification via a verifiable complaint which we introduce reduces interaction and bandwidth. One can reduce interaction further by resorting to a non-interactive publicly verifiable secret sharing scheme; however these schemes are much more costly in terms of computation.

Agreement 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 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 $|\mathsf{QUAL}| = d_1 \geq b + t$. To get the best amortized efficiency, we want to make b as large as possible, which means using as large a set QUAL as we can get. Every party can serve as a dealer for the DGK 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 agreement protocol with parameters $d_0 = d_1 = n - t$. (If we have less messages to sign, we can make do with a smaller b, which means smaller d_1, any value $d_1 > t$ would work.)

Agreement in the dynamic/proactive setting. In this setting each dealer shares two polynomials, a random polynomial $\mathbf{H}_i$ of degree $d' = t + 2a - 2$ and a packed re-sharing of its share via $\mathbf{F}_i$ of degree $d = t + a - 1$. Hence, we have two types of messages and two QUAL's, one for $\mathbf{H}$ and the other for $\mathbf{F}$. For $\mathbf{H}$ we have the same parameters as above, $d_0 = d_1 = n - t$. For $\mathbf{F}$, we need $d + 1 = t + a$ dealers in QUAL_2 in order for shareholders in HOLD to be able to recover their shares, so we set $d_2 = t + a$.

(We note that the dealers in QUAL_2 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 $d_0 = n - t$, and t of them could be corrupted, so we cannot set d_2 any larger than $n - 2t$. This implies the constraint $n - 2t \geq t + a$ or $n \geq 3t + a$, which is weaker than the constraint $n \geq 3t + 2a - 1$ that we had above.)

5 Deploying in a Blockchain Environment

Next we describe how the protocols from above can be used to implement a large-scale Schnorr-signature service over a public blockchain. Such implementation would use all the techniques from Section 2. Specifically,

- It would use the QUAL-agreement protocol from Section 4, where the blockchain serves as the underlying total-order broadcast channel;
- It includes the proactive share refresh from Section 3.2, which is run periodically every few blockchain rounds (called an epoch);
- The committees in each epoch are sub-sampled from among the validators using the randomness beacon as discussed in Section 2.8;
- It would include all the optimizations from Section 2.9, such as optimistic runs and user aggregation.

Below we discuss a few other aspects of such implementation, such as the required committee sizes, how to use the single blockchain key s to sign on behalf of multiple smart contracts, and plausible mechanisms for recovering from catastrophic failures.

5.1 Committee Sizes

We assume a huge network, consisting of many millions of parties, from which we sub-sample the committees. We note that this model is sometimes applicable even for a blockchain with only a handful of validators. Specifically, in a proof-of-stake blockchain we may want to give high-stake nodes more power in the computation than low-stake ones, to align the trust model of the signature service with that of the underlying consensus protocol. (E.g., both the consensus protocol and the higher-level signature scheme remain secure as long as at least 80% of the stake is controlled by honest parties.)

In that case, we may want to view each token as a party, where a physical node is running the protocol on behalf of all the tokens that it controls. Even if the network only has a few dozen physical nodes, this view may require that we use our protocols with many millions of virtual parties.[11]

Below we denote the overall number of parties by N, and we assume a PKI where we have a list of all N parties, each with their public encryption key. Also set f be (an upper bound on) the fraction of corrupted parties, namely we assume that we have at most fN corrupted parties and at least $(1-f)N$ honest ones.

Recall that we sub-sample each committee based on a (pseudo)random value from the randomness beacon, expanded using a PRG. In more detail, for some parameter n (to be determined below), we expand the latest randomness beacon value U into a (pseudo)random vector of n indexes in $[N]$ (with repetitions), $PRG(U) = (i_1, i_2, \ldots, i_n) \in [N]^n$. The next committee then consists of the n parties that are indexed by $i_1, \ldots, i_n$ in the PKI.[12]

Chosen this way, the committee size will be exactly n, and the number of corrupted parties in the committee will be upper-bounded by $X_C \sim \mathsf{Bin}_{n,f}$, a Binomial random variable with parameters n, f. Similarly, the number of honest parties in the committee is lower-bounded by $X_H \sim \mathsf{Bin}_{n,1-f}$. Conveniently, these two random variables do not depend on N, the total number of parties in the system.

The parameters n, t will be chosen based on a and f (and the required safety and liveness guarantees $\varepsilon_{\text{safety}}, \varepsilon_{\text{liveness}}$), to ensure the following conditions:

Safety. To prevent the corrupted parties from learning the secret key, we need $\Pr[X_C > t] \le \varepsilon_{\text{safety}}$.

Liveness. We need $\Pr[X_H < 2t + 2a - 1] \le \varepsilon_{\text{liveness}}$ to ensure that honest parties can reconstruct the signatures.

To find suitable parameters, we therefore wrote a simple program that takes as input f, a, $\varepsilon_{\text{safety}}, \varepsilon_{\text{liveness}}$, and searches for the smallest values for n, t that satisfy these two conditions. See Appendix D. Once those parameters are set, we instantiate the system with a QUAL-agreement protocol from Fig. 3 with parameters $d_0 = d_1 = n - t$ and $d_2 = t + a$. This implies a parameter b (the dimension of the super-invertible matrix) which is $b = |\mathsf{QUAL}_1| - t = n - 2t$.

5.1.1 Optimistic Parameters

As mentioned in Section 2.9, we can also attempt to run with optimistic parameters, i.e., smaller committee, as long as it is only liveness that we sacrifice, not safety. For that purpose, we modified the parameter-searching program to take another set of parameters $f', \varepsilon'_{\text{liveness}}$, then define $X'_H \sim \mathsf{Bin}_{n,1-f'}$ and search also for parameters that satisfy $\Pr[X_C > t] \le \varepsilon_{\text{safety}}$ and $\Pr[X'_H < 2t+2a-1] \le \varepsilon'_{\text{liveness}}$.

[11] This is also the setting where the user-aggregation ideas from Section 2.9 will have the most impact.

[12] If an index $i \in [N]$ appears more than once in the vector, then the corresponding party will have more than one seat on the committee and will get more than one share of the relevant secrets.

5.2 Uses of Additive Key Derivation

Additive key derivation (cf. [13]) allow a single secret key to be used to sign on behalf of multiple public keys. Let s be a "master secret key" with corresponding public key $S = s \cdot G$. Then a derived key-pair can be specified by a public "tweak" $u \in \mathbb{Z}_p$, which defines the secret key $s' = s + u$ and the public key $S' = S + u \cdot G$. This is a well known technique, which is used in many blockchains (including Bitcoin [25]). To generate a Schnorr signature on message M relative to the derived key S', one can use the master secret key in conjunction with the public tweak.

In the context of a blockchain signature service, we consider a blockchain master key whose secret key is shared among the validators as described in this work. We can then give each smart contract its own derived key, by setting the tweak value u to be (say) a hash of the smart contract identity I (or code): $u = \mathsf{Hash}(I)$. This way, each smart contract "owns" its own key, and can ask the blockchain to sign messages relative to that key. When a smart contract with identity I asks to sign a message M, the validators will execute the Schnorr signature protocol exactly as specified in this work, except that they use the public scalar $e = \mathsf{Hash}(S', R, M)$ for that message (instead of $e = \mathsf{Hash}(S, R, M)$). This will result in a pair $(R, es + r)$, that can be converted to standard Ed25519 signature by adding $e \cdot \mathsf{Hash}(I)$ to the second entry in the pair.

5.3 Recovery from Catastrophic Failures

In the proactive setting that we consider, the system relies on adequate connectivity to make progress and refresh the secret key from one committee to the next. A plausible attack vector is mounting a network partition attack. This will stall progress and deprive the parties of the ability to refresh the sharing and erase their old shares. If the outage lasts for a long time, it may become necessary for nodes to delete their old shares without refreshing them, for fear that a long-held secret may become an easy target for an attack. If that happens, how can the system recover and return to normal operation once the outage is over?

One recovery approach is to fall back on centralized trust: the master key can be kept in (very) cold storage, perhaps shared among a handful of highly trusted parties, and recovered from them in the event of such a devastating attack. This solution, however, has the drawback that these "highly trusted" parties may not be trustworthy after all: They can recover the key even with no attack, and there would not even be any way of knowing if they did it.

A somewhat better approach would be to equip those highly trusted parties with an independent recovery decryption key, with the corresponding public key embedded in the blockchain code. This key is never used, except when a shareholder needs to delete their share without being able to pass it forward. In that situation, the shareholder will encrypt their share under the recovery public key before deleting it from memory. Once the outage is over, they will run a special recovery protocol, in which the highly trusted parties help them to recover the share and continue where they left of. It is even possible to implement a hybrid approach, where some small number of shares are always encrypted under the public key, but most other shares are only encrypted when an attack happens. This way the highly trusted parties can fill in for some shareholders that lost interest after the attack and did not participate in the recovery process.

Yet another approach, which relies on anonymous public-key encryption, is as follows: When a shareholder needs to delete their share without being able to pass it forward, it chooses a random committee and secret-share its share to them, broadcasting an encryption of the sub-shares under their anonymous-PKE keys. This way, the adversary does not know who is holding what shares, but the parties can still recover those shares when the system resumes.

References

[1] Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, and Gilad Stern. Bingo: Adaptively secure packed asynchronous verifiable secret sharing and asynchronous distributed key generation. Cryptology ePrint Archive, Paper 2022/1759, 2022. https://eprint.iacr.org/2022/1759.

[2] Zuzana Beerliová-Trubíniová and Martin Hirt. Perfectly-secure MPC with linear communication complexity. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 213–230. Springer, Heidelberg, March 2008.

[3] Fabrice Benhamouda, Tancrède Lepoint, Julian Loss, Michele Orrù, and Mariana Raykova. On the (in)security of ROS. *J. Cryptol.*, 35(4):25, 2022.

[4] Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theor. Comput. Sci.*, 2019.

[5] Sourav Das, Zhuolun Xiang, Lefteris Kokoris-Kogias, and Ling Ren. Practical asynchronous high-threshold distributed key generation and distributed polynomial sampling. Cryptology ePrint Archive, Report 2022/1389, 2022. https://eprint.iacr.org/2022/1389.

[6] Sourav Das, Thomas Yurek, Zhuolun Xiang, Andrew K. Miller, Lefteris Kokoris-Kogias, and Ling Ren. Practical asynchronous distributed key generation. In *2022 IEEE Symposium on Security and Privacy*, pages 2518–2534. IEEE Computer Society Press, May 2022.

[7] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th FOCS*, pages 427–437. IEEE Computer Society Press, October 1987.

[8] Matthew K. Franklin and Moti Yung. Communication complexity of secure computation (extended abstract). In *24th ACM STOC*, pages 699–710. ACM Press, May 1992.

[9] François Garillot, Yashvanth Kondi, Payman Mohassel, and Valeria Nikolaenko. Threshold Schnorr with stateless deterministic signing from standard assumptions. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 127–156, Virtual Event, August 2021. Springer, Heidelberg.

[10] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology*, 20(1):51–83, January 2007.

[11] Rosario Gennaro, Michael O. Rabin, and Tal Rabin. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In Brian A. Coan and Yehuda Afek, editors, *17th ACM PODC*, pages 101–111. ACM, June / July 1998.

[12] Jens Groth and Victor Shoup. Design and analysis of a distributed ECDSA signing service. Cryptology ePrint Archive, Report 2022/506, 2022. https://eprint.iacr.org/2022/506.

[13] Jens Groth and Victor Shoup. On the security of ECDSA with additive key derivation and presignatures. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 365–396. Springer, Heidelberg, May / June 2022.

[14] Martin Hirt, Christoph Lucas, and Ueli Maurer. A dynamic tradeoff between active and passive corruptions in secure multi-party computation. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 203–219. Springer, Heidelberg, August 2013.

[15] Martin Hirt and Jesper Buus Nielsen. Robust multiparty computation with linear communication complexity. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 463–482. Springer, Heidelberg, August 2006.

[16] Snehil Joshi, Durgesh Pandey, and Kannan Srinathan. Atssia: Asynchronous truly-threshold schnorr signing for inconsistent availability. In Jong Hwan Park and Seung-Hyun Seo, editors, *Information Security and Cryptology – ICISC 2021*, pages 71–91, Cham, 2022. Springer International Publishing.

[17] Chelsea Komlo and Ian Goldberg. FROST: Flexible round-optimized schnorr threshold signatures. Cryptology ePrint Archive, Report 2020/852, 2020. https://eprint.iacr.org/2020/852.

[18] Chelsea Komlo and Ian Goldberg. FROST: Flexible round-optimized Schnorr threshold signatures. In Orr Dunkelman, Michael J. Jacobson Jr., and Colin O'Flynn, editors, *SAC 2020*, volume 12804 of *LNCS*, pages 34–65. Springer, Heidelberg, October 2020.

[19] Yehuda Lindell. Simple three-round multiparty schnorr signing with full simulatability. Cryptology ePrint Archive, Report 2022/374, 2022. https://eprint.iacr.org/2022/374.

[20] Wafa Neji, Kaouther Blibech, and Narjes Ben Rajeb. Distributed key generation protocol with a new complaint management strategy. *Security and Communication Networks*, 9(17):4585–4595, 2016.

[21] Torben P. Pedersen. A threshold cryptosystem without a trusted party (extended abstract) (rump session). In Donald W. Davies, editor, *EUROCRYPT'91*, volume 547 of *LNCS*, pages 522–526. Springer, Heidelberg, April 1991.

[22] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 129–140. Springer, Heidelberg, August 1992.

[23] David Pointcheval and Jacques Stern. Provably secure blind signature schemes. In Kwangjo Kim and Tsutomu Matsumoto, editors, *ASIACRYPT'96*, volume 1163 of *LNCS*, pages 252–265. Springer, Heidelberg, November 1996.

[24] Tim Ruffing, Viktoria Ronge, Elliott Jin, Jonas Schneider-Bensch, and Dominique Schröder. ROAST: Robust asynchronous schnorr threshold signatures. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 2551–2564. ACM Press, November 2022.

[25] Pieter Wuille. BIP 0032, bitcoin improvement proposal. https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki, 2012.

[26] Thomas Yurek, Licheng Luo, Jaiden Fairoze, Aniket Kate, and Andrew K. Miller. 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*. The Internet Society, 2022.

A Security Proof

We now turn to proving the security of our base threshold signature protocol for the static-committee setting from Fig. 1. We build the proof step by step, starting from the proof for the (centralized) Schnorr signature scheme [23] and a simple threshold Schnorr signature protocol a-la-GJKR, adapting their proof techniques and adding components as needed for our protocols. Specifically, we define a list of variants of the protocol, and explain how the proof is modified from one variant to the next:

1. Centralized Schnorr (Appendix A.1): Pointcheval and Stern [23] proved that, under the discrete log assumption, Schnorr signature scheme is secure in the random oracle model. The key component of their proof is the forking lemma [23, Theorem 10], which we will also use for our proof in the following variants.

2. Threshold Schnorr for a single message (Appendix A.2): This is similar to (but not exactly the same as) the GJKR protocol from [10, Fig.4], using Feldman commitments. While using Feldman allows a rushing adversary to bias the distribution of the ephemeral randomness, Gennaro et al. proved in [10] that their threshold signature protocol with Feldman commitments is still secure. We prove the same for our protocol by adapting their techniques to our needs, see Appendix A.2.

3. Parallel Threshold Schnorr signatures (Appendix A.3): It is known that the GJKR proof for the single-signature threshold scheme does not extend to signing multiple messages in parallel, in fact the resulting scheme is insecure. We therefore add a mitigation technique that allows us to recover the security argument when signing a set of messages in parallel as our protocol does.

4. Non-Packed threshold Schnorr with a super-invertible matrix (Appendix A.4): Our use of a super-invertible matrix allows each dealer to shares only a single polynomial, but we can still derive multiple signatures. The security proof for this variant requires a generalization of the simulation technique, as well as a small change to the protocol itself.

5. Threshold Schnorr with super-invertible matrix and packing (Appendix A.5): This part requires another generalization of the simulation technique.

Combining all these techniques, we describe the final reduction in Appendix A.6. Finally, in Appendix A.7 we then discussed the small changes for the dynamic/proactive setting. We begin with some preliminaries.

Assumption 1 (The Discrete-Logarithm Assumption)**.** Let $\mathbb{G}$ be a group of prime order $p \in \mathsf{PRIME}(\lambda)$ where the generator is G, and λ is the security parameter. Define the following attack game for a discrete log adversary $\mathcal{A}$:

- The challenger and the adversary $\mathcal{A}$ take in a description of $\mathbb{G}$, which includes the group order p and the generator G.
- The challenger chooses $s \xleftarrow{\$} \mathbb{Z}_p$, and gives $\mathcal{A}$ the group element $s \cdot G$.
- $\mathcal{A}$ outputs $\bar{s}$.

We say that $\mathcal{A}$ wins the game if $\bar{s} = s$ and we denote the probability of $\mathcal{A}$ winning the game as $\mathcal{E}_{\mathsf{DL}}[\mathcal{A}, \mathbb{G}]$. The discrete logarithm assumption holds if $\mathcal{E}_{\mathsf{DL}}[\mathcal{A}, \mathbb{G}]$ negligible in λ, i.e., $\mathcal{E}_{\mathsf{DL}}[\mathcal{A}, \mathbb{G}] \leq 1/P(\lambda)$ for any polynomial $P(\cdot)$.

Definition 1 (Security of signature scheme)**.** For a signature scheme $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$, define a following game for an adversary $\mathcal{A}$:

- The challenger runs $\mathsf{Gen}(1^\lambda) \to (SK, PK)$ and sends PK to $\mathcal{A}$.
- $\mathcal{A}$ asks the challenger for signatures on $M_1, \ldots, M_q$; and the challenger computes $\mu_i \leftarrow \mathsf{Sign}(SK, M_i)$ for $i = 1, \ldots, q$, and sends the corresponding signatures $\mu_1, \ldots, \mu_q$ to $\mathcal{A}$.
- $\mathcal{A}$ outputs a pair (M, μ).

We say that $\mathcal{A}$ wins the game if $M \notin \{M_1, \ldots, M_q\}$ and yet $\mathsf{Verify}(PK, M, \mu) = 1$. The advantage of $\mathcal{A}$ is the probability that $\mathcal{A}$ wins the game, denoted $\mathcal{E}_{\mathsf{forge}}[\mathcal{A}, \Sigma]$. We say that the signature scheme Σ is secure if, for any polynomial-time $\mathcal{A}$, $\mathcal{E}_{\mathsf{forge}}[\mathcal{A}, \Sigma]$ is negligible in λ.

A.1 Centralized Schnorr

Definition 2 (Schnorr signature scheme)**.** Let Hash be a hash function $\{0,1\}^* \to \mathbb{Z}_p$. Let $\mathbb{G}$ be a group of order p in which discrete log is hard. The Schnorr signature scheme consists of the following algorithms:

- $\mathsf{Gen}(1^\lambda) \to (SK, PK)$: a randomized algorithm run by the signer that takes in a security parameter λ, outputs $SK = s \xleftarrow{\$} \mathbb{Z}_p$ and $PK = S = s \cdot G$.
- $\mathsf{Sign}(s, M) :\to \mu$: a randomized algorithm run by the signer that on input a message M, a secret key $SK = s$, samples $r \xleftarrow{\$} \mathbb{Z}_p$, computes $R = r \cdot G$, $e = \mathsf{Hash}(S, R, M)$, and $\phi = r + es$. Output a signature $\mu := (R, \phi)$.
- $\mathsf{Verify}(S, M, \mu) \to v$: a deterministic algorithm run by the verifier that on input a public key $PK = S$, a message M, and a signature $\mu = (R, \phi)$, computes $e = \mathsf{Hash}(S, R, M)$ and checks if $\phi \cdot G = R + e \cdot PK$. If so, it outputs $v = 1$ (indicating the signature is valid). Otherwise, it outputs $v = 0$ (invalid signature).

Below we briefly recall the security proof of Schnorr signature scheme, because it helps in understanding the security proof of the threshold signing.

Theorem 2 ([23])**.** The Schnorr signature scheme is secure in the random-oracle model under the discrete logarithm assumpiton.

Proof. Assume that there exists an attacker $\mathcal{A}$ that can forge the signature, then we can build an attacker Sim that solves discrete log. The task of the attacker Sim is to simulate the view of $\mathcal{A}$ as in the real-world protocol execution (so that $\mathcal{A}$ can output a forgery), and then Sim transforms $\mathcal{A}$'s forgery into the ability to compute discrete log. We provide details below.

The discrete log challenger first samples a random s and sends $S = s \cdot G$ to Sim. The task for Sim is to utilize $\mathcal{A}$ to find s. In the first step of the simulation, $\mathcal{A}$ sends q messages $M_1, \dots, M_q$ to Sim, and Sim needs to create signatures on these messages to send to $\mathcal{A}$. However, Sim cannot create such signatures "without any help" because it does not have the secret key s. To this end, we assume the hash is a programmable random oracle and Sim programs the random oracle Hash as follows: for each query (S, R, M) to Hash, it samples e, ϕ from $\mathbb{Z}_p$ at random, and then computes $R := \phi \cdot G - e \cdot S$ and set $\mathsf{Hash}(S, R, M)$ to e; and Sim will give $\mathcal{A}$ the tuple (R, ϕ) as the signature to m. When $\mathcal{A}$ verifies the signature, it computes $\phi \cdot G - R$, which equals $e \cdot S$ as random oracle Hash was programmed this way.

Now we show how $\mathcal{A}$ can create a forgery (R^*, ϕ^*) on a message M^*, where $M^* \notin \{M_1, \dots, M_q\}$. In the random oracle model, we assume that $\mathcal{A}$ must have queried Hash on (S, R^*, M^*). Then Sim rewinds $\mathcal{A}$, and answers all the oracle queries before (S, R^*, M^*) with the same value used in the first run, but on the oracle query (S, R^*, M^*), Sim answers with a new e', randomly chosen from $\mathbb{Z}_p$ (note that R^* is the same as in the first run). In the second run, $\mathcal{A}$ outputs a forgery and if it happens to be M^* again, then Sim can compute the secret key s. Here Sim during rewinding can guess which message $\mathcal{A}$ will forge with uniform probability (choose where to rewind) so that the probability of success will be at least $1/\mathcal{Q}$, where $\mathcal{Q}$ is the total number of oracle queries made by $\mathcal{A}$. □

A.2 Threshold Schnorr for a single message

Recall the distributed key generation protocol (called JF-DKG as in GJKR [10]):

1. Each party i (acts as a dealer) chooses a random degree-t polynomial $\mathbf{H}_i(\cdot)$, where $r_i = \mathbf{H}_i(0)$ is the random value that party i wants to (additively) contribute, and $R_i = \mathbf{H}_i(0) \cdot G$ is the corresponding public value. Each party broadcasts 1) $\mathsf{ENC}_{PK_j}(\mathbf{H}_i(j))$, i.e., the encryption of share for party j under party j's public key; and 2) the Feldman commitments $\hat{H}_i$ to $\mathbf{H}_i$.

   (Below we elide the distinction between committing to coefficients of $\mathbf{H}$ or to its values at sufficiently many points, since these are equivalent from a security perspective. We just use the fact that given $\hat{H}_i$, it is possible to compute $\mathbf{H}_i(z) \cdot G$ for every $z \in \mathbb{Z}_p$. In particular R_i is publicly known.)

2. The parties engage in an agreement protocol to agree on sets HOLD and QUAL such that $|\mathsf{HOLD}|, |\mathsf{QUAL}| \geq t + 1$ and every party in HOLD recived from every party in QUAL shares that are consistent with the committed polynomials.

3. Let $\mathbf{H} = \sum_{i \in \mathsf{QUAL}} \mathbf{H}_i$, each party $j \in \mathsf{HOLD}$ compute their share as $\rho_j = \mathbf{H}(j) = \sum_{i \in \mathsf{QUAL}} \mathbf{H}_i(j)$. The parties also compute a Feldman commitment to $\mathbf{H}$, $\hat{H} = \sum_{i \in \mathsf{QUAL}} \hat{H}_i$. The secret that is shared among them is $r = \mathbf{H}(0) = \sum_{i \in \mathsf{QUAL}} \mathbf{H}_i(0) \in \mathbb{Z}_p$. The corresponding public value is $R = \mathbf{H}(0) \cdot G = \sum_{i \in \mathsf{QUAL}} R_i \in \mathbb{G}$, which can be computed from $\hat{H}$.

We use JF-DKG as a main component in our distributed Schnorr signature protocol, specifically it is run to generate the ephemeral randomness that is needed for these signatures. Our protocol, described below, is similar (but not exactly the same as) the protocol from [10].

Protocol inputs. A message M to be signed, a degree-t sharing of a secret key s. Each party i holds a Shamir share of s, denoted as $\sigma_i = \mathbf{F}(i)$, with $\mathbf{F}$ a degree-t polynomial and $\mathbf{F}(0) = s$. Also, $\hat{F}$ if publicly known, from which it is possible to compute $S = s \cdot G$ and $S_i = \sigma_i \cdot G$ for all i

Protocol outputs. A Schnorr signature of the form (R, ϕ) on the message M.

Protocol steps. The threshold signing mainly consists of three parts: first, generate ephemeral randomness for signing using JF-DKG; second, every party locally computes hash and its Shamir share of the signature; finally, the parties combine the shares to reconstruct the signature.

1. The parties run the JF-DKG protocol above. After this, a set HOLD of parties is determined, where each party $j \in \mathsf{HOLD}$ holds a degree-t Shamir share $\rho_j = \mathbf{H}(j)$ of the ephemeral randomness $r = \mathbf{H}(0)$, where $\mathbf{H}$ is a degree-t polynomial, and everyone knows a Feldman commitment $\hat{H}$ to $\mathbf{H}$.

   Let $R = \mathbf{H}(0) \cdot G$ and $\tilde{R_j} = \mathbf{H}(j) \cdot G$ for all $j \in \mathsf{HOLD}$. These can all be computed from the Feldman commitment $\hat{H}$.

2. The parties locally compute $e = \mathsf{Hash}(S, R, M)$.

3. Each party $j \in \mathsf{HOLD}$ broadcasts its share of signature, $\pi_j = \mathbf{H}(j) + e \cdot \sigma_j$. Each share is verified by checking if $\pi_j \cdot G = \tilde{R_j} + e \cdot S_j$.

4. If at least $t + 1$ parties in HOLD broadcast valid shares, use these shares to reconstruct $\phi = r + e \cdot s$. Output a signature on M as (R, ϕ).

The difference between this protocol and the one from [10] is that in our protocol, the parties in HOLD (that hold Shamir sharing of r, s) generate the signature, whereas in [10] it is the parties in QUAL that do it (using additive sharing of r, s). However, the same security challenge resides here as in our protocol, the adversary can play with QUAL for the ephemeral randomness (e.g., kick out a contributed polynomial of an honest party). The security proof is similar to that in [10], but not identical since the protocols are somewhat different.

A.2.1 Proof of Security

Definition 3 (Security of threshold signature)**.** Define the following game for a threshold signature protocol Π and an adversary $\mathcal{A}$ against a distributed signature protocol Π:

- $\mathcal{A}$ while interacting with Π, comes up with $M_1, \ldots, M_q$, and gets the corresponding q signatures $(\mu_1, \ldots, \mu_q)$.
- After interacting with Π, $\mathcal{A}$ outputs a pair (M, μ).

We say that $\mathcal{A}$ wins the game if $M \notin \{M_1, \ldots, M_q\}$ and yet $\mathsf{Verify}(PK, M, \mu) = 1$. The advantage of $\mathcal{A}$ to be the probability that $\mathcal{A}$ wins the game, denoted as $\mathcal{E}_{\mathsf{forge}}[\mathcal{A}, \Pi]$. We say that a threshold signature protocol Π is secure if for any polynomial-time $\mathcal{A}$, $\mathcal{E}_{\mathsf{forge}}[\mathcal{A}, \Pi]$ is negligible.

Theorem 3. The threshold signature protocol from above is secure against a static advesary corrupting upto t parties, in the random-oracle model under the discrete logarithm assumption.

Proof. We describe a reduction, in which an adversary $\mathcal{A}$ against the threshold scheme can be used to solve the discrete logarithm problem. The main difference from the proof for centralized Schnorr is that the simulator must also simulate the adversary view of the protocol, not just of the signatures themselves. The main challenge in this proof is that the adversary can be rushing, which enables it to bias the distribution of the ephemeral randomness, so the simulator cannot just select the R at random.

Let Honest be the set of honest parties, let $\mathsf{Corrupt} = [n] \setminus \mathsf{Honest}$ be the corrupted parties, and let q be a bound on the number of random-oracle queries of the form (S, R, M) that $\mathcal{A}$ makes. The simulator needs to simulate for $\mathcal{A}$ the following aspects:

- The Feldman commitment $\hat{F}$ to the degree-t polynomial $\mathbf{F}$ used to share the long-term key;
- The shares $\sigma_i = \mathbf{H}(i)$ of the long-term secret key for all $i \in \mathsf{Corrupt}$;
- The (encryption of) shares of ephemeral randomness that the honest parties send to the corrupted parties;
- The Feldman commitments $\hat{H}_i$ to the honest parties' ephemeral randomness polynomials $\mathbf{H}_i$;
- The view of the agreement protocol, including the verifiable complaints and support messages;
- The full degree-t signature polynomial $\mathbf{Y}$ such that $\phi = \mathbf{Y}(0)$ is part of the signature (and $\mathbf{Y}$ is consistent with $\hat{F}$ and the $\hat{H}_i$'s);
- In addition to all the above, the simulator also needs to answer random-oracle queries (S, R, M) that $\mathcal{A}$ makes.

The reduction. The discrete log challenger randomly samples s from $\mathbb{Z}_p$ as the secret key and gives Sim the corresponding public key $PK = S = s \cdot G \in \mathbb{G}$. The public key is given to $\mathcal{A}$. Now Sim's task is to find s, by utilizing $\mathcal{A}$.

The simulator begins by choosing t random and independent scalars for the secret-key shares σ_i for all $i \in \mathsf{Corrupt}$ and giving them to $\mathcal{A}$. It also "interpolates in the exponent" a commitment to a degree-t polynomial which is consistent with the public key S and $\sigma_i \cdot G$ for all $i \in \mathsf{Corrupt}$.

In more detail, for each $z \in \mathsf{Corrupt} \cup \{0\}$ let $\mathbf{I}_z$ be the degree-t polynomial satisfying $\mathbf{I}_z(z) = 1$ and $\mathbf{I}_z(y) = 0$ for all $y \in \mathsf{Corrupt} \cup \{0\}$, $y \neq z$. Then the polynomial $\mathbf{F}$ is $\mathbf{F} = s \cdot \mathbf{I}_0 + \sum_{z \in \mathsf{Corrupt}} \sigma_z \cdot \mathbf{I}_z$. Denoting $\mathbf{F}_{\mathsf{Corrupt}} = \sum_{z \in \mathsf{Corrupt}} \sigma_z \cdot \mathbf{I}_z$, we can write $\mathbf{F} = \mathbf{F}_{\mathsf{Corrupt}} + s \cdot \mathbf{I}_0$, where the simulator knows $\mathbf{F}_{\mathsf{Corrupt}}$ in the clear.

Next, the simulator chooses at random an honest party $i^* \in \mathsf{Honest}$, which will be simulated differently than the other honest parties. Sim also chooses a random-oracle query index $\ell \in [q]$, hoping that the random-oracle query where $\mathcal{A}$ asks about (S, R, M) that are used in the signature is the ℓ'th query.

Throughout the simulation, the simulator answers random-oracle queries with independent random scalars $e_1, \ldots, e_q$, but e_ℓ will play a special role. Specifically, Sim chooses e_ℓ at the outset, together with another scalar $\tilde{\phi}$, and sets $R_{i^*} = \tilde{\phi} \cdot G - e_\ell \cdot S$. For the rest of the honest parties ($i \in \mathsf{Honest}$, $i \neq i^*$), Sim chooses r_i's at random and sets $R_i = r_i \cdot G$.

To simulate the Feldman commitments in Step 1 in JF-DKG, Sim simply follows DKG the protocol as prescribed for honest parties other than i^*. For party i^*, Sim needs to generate the commitment $\hat{H}_{i^*}$ and the shares $\rho_{i^*j} = H_{i^*}(j)$ for $j \in \mathsf{Corrupt}$, without knowing the discrete lorarithm of R_{i^*}. These have to remain consistent, namely for all $j \in \mathsf{Corrupt}$ we need

$$\rho_{i^*j} \cdot G = R_{i^*} + \sum_{k=1}^{d} j^k \cdot (h_{i^*,j} \cdot G).$$

While Sim does not know the polynomial $\mathbf{H}_{i^*}$ in the clear (since its free term was chosen based on the unknown secret key), it can still create commitment $\hat{H}_{i^*}$ that satisfies the relation above. Sim chooses at random $\rho_{i^*j} \leftarrow \mathbb{Z}_p$ for all $j \in \mathsf{Corrupt}$, and uses them as the shares of all the corrupted parties. Note that $\{\rho_{i^*j} \cdot G : j \in \mathsf{Corrupt}\}$, together with R_{i^*}, uniquely define the polynomial $\mathbf{H}_{i^*}$, which satisfies $\mathbf{H}_{i^*}(0) = \tilde{\phi} - e_\ell \cdot s$ and $\mathbf{H}_{i^*}(j) = \rho_{i^*j}$ for all $j \in \mathsf{Corrupt}$. Moreover, Sim can generate the commitment $\hat{H}_{i^*}$ by "interpolating in the exponent", without needing to know $\mathbf{H}_{i^*}$ in the clear. Using similar notations to above we can write

$$\mathbf{H}_{i^*} = \sum_{j \in \mathsf{Corrupt}} \rho_{i^*j} \cdot \mathbf{I}_j + (\tilde{\phi} - e_\ell \cdot s) \cdot \mathbf{I}_0 = \underbrace{\textstyle\sum_{j \in \mathsf{Corrupt}} \rho_{i^*j} \cdot \mathbf{I}_j + \tilde{\phi} \cdot \mathbf{I}_0}_{=\mathbf{H}_{\mathsf{Corrupt}}} - e_\ell \cdot s \cdot \mathbf{I}_0, \qquad (1)$$

where the simulator knows $\mathbf{H}_{\mathsf{Corrupt}}$ in the clear.

This completes the simulation of Step 1 of the DKG, the simulator sends all the shares and commitments of honest parties to the adversary. Then $\mathcal{A}$ sends back to Sim the polynomial $\mathbf{H}_i$ for parties $i \in \mathsf{Corrupt}$. (Sim gets the shares ρ_{ij} for $j \in \mathsf{Honest}$, and since it controls $t+1$ or more parties it can recover $\mathbf{H}_i$ in full).

Next the simulator runs the prescribed agreement protocol on behalf of the honest parties, but treating i^* as an honest dealer, even though the ciphertexts that it broadcasts to the honest parties encrypt garbage. At the end of the agreement protocol, QUAL and HOLD are detemined, which in turn defines also $\mathbf{H} = \sum_{i \in \mathsf{QUAL}} \mathbf{H}_i$ and $R = \mathbf{H}(0) \cdot G = \sum_{i \in \mathsf{QUAL}} R_i$. While the simulator does not know $\mathbf{H}$ in the clear, it does know the commitment to it $\hat{H}$.

Sim aborts if either $i^* \notin \mathsf{QUAL}$,[13] or if $\mathcal{A}$ already made the random-oracle query (S, R, M) and it was not the ℓ'th query. (If $\mathcal{A}$ still did not make the query (S, R, M) by the time that R is defined, then the simulator will answer that query when it arrives with e_ℓ, regardless of the index of that query.)

Since there is at least one honest party in QUAL, then the first abort event happens with probability at most $(n-1)/n$. The second abort event happens with probability at most $(q-1)/q$, since there are at most q random-oracle queries. Hence, the simulator will proceed with probability at least $1/qn$. If the simulator did not abort, then we have

$$\mathbf{H} \;=\; \textstyle\sum_{i \in \mathsf{QUAL} \setminus \{i^*\}} \mathbf{H}_i \;+ \mathbf{H}_{i^*} \;=\; \underbrace{\textstyle\sum_{i \in \mathsf{QUAL} \setminus \{i^*\}} \mathbf{H}_i + \mathbf{H}_{\mathsf{Corrupt}}}_{=\mathbf{H}_{\mathsf{CLR}}} - e_\ell \cdot s \cdot \mathbf{I}_0,$$

where the simulator knows $\mathbf{H}_{\mathsf{CLR}}$ in the clear.

Now the simulator proceeds to Step 4 of the signature protocol, where the honest parties broadcast their signature shares. Note that if the simulator did not abort, then we are ensured that

[13] While no shareholder will broadcast a complaint against party i^*, we cannot ensure that it is in QUAL since this is an asynchronous network and the adversary can delay the messages from party i^* until after QUAL is determined.

$\mathsf{Hash}(S, R, M) = e_\ell$. The required signature is therefore $(R, r + e_\ell \cdot s)$, where $r = \mathbf{H}(0)$ is the discrete logarithm of R and $s = \mathbf{F}(0)$ is the discrete logarithm of S.

But the simulator needs to produce more than just the signature, it needs to come up with the full degree-t polynomial $\mathbf{Y} = \mathbf{H} + e_\ell \cdot \mathbf{F}$ (where the commitments $\hat{F}, \hat{H}$ to $\mathbf{F}$ and $\mathbf{H}$ are already fixed). Luckily, this is easy to do: Recall that $\mathbf{F} = \mathbf{F}_{\mathsf{Corrupt}} + s \cdot \mathbf{I}_0$ and $\mathbf{H} = \mathbf{H}_{\mathsf{CLR}} - e_\ell \cdot s \cdot \mathbf{I}_0$, and the simulator knows $\mathbf{F}_{\mathsf{Corrupt}}$, $\mathbf{H}_{\mathsf{CLR}}$ in the clear. Hence,

$$\mathbf{Y} \;=\; \mathbf{H} + e_\ell \cdot \mathbf{F} = \mathbf{H}_{\mathsf{CLR}} - e_\ell \cdot s \cdot \mathbf{I}_0 + e_\ell \cdot (\mathbf{F}_{\mathsf{Corrupt}} + s \cdot \mathbf{I}_0) = \mathbf{H}_{\mathsf{CLR}} + e_\ell \cdot \mathbf{F}_{\mathsf{Corrupt}}$$

which the simulator can output in the clear.

This concludes the simulation portion, and all that is left is to apply the forking lemma exactly as in the proof of the centralized Schnorr signature. Suppose $\mathcal{A}$ creates a forgery on M^* which is (R^*, ϕ^*). In the random oracle model, we assume that in order for $\mathcal{A}$ to generate such forgery it must have queried the oracle on (S, R^*, M^*). Let Sim rewind $\mathcal{A}$, changing the answer to the query (S, R^*, M^*). If $\mathcal{A}$ is still able to forge a signature on M^* with randomness R^*, then the simulator can extract s from those two signatures. □

A.3 Threshold Schnorr for multiple messages

Suppose we wanted to use the protocol from Appendix A.2 to sign multiple messages in parallel. A natural way of doing this would be to let each dealer D_i generate multiple polynomials $H_{u,i}$, $u \in [b]$, in b copies of the single-message protocol. However, the security of the parallel threshold Schnorr cannot be directly derived from the simulation proof for single-message protocol. Recall that in the proof in Appendix A.2.1, in order to generate valid signatures on the b messages $M^1, \ldots, M^b$, the simulator needs to guess the R's for the b messages, which brings down the succeeding probability (for guessing the correct oracle queries) from $1/q$ to $1/q^b$. Below we only give a brief overview of how the proof changes; more details are found in Appendix A.6.

Mitigation and proof technique. To mitigate the security downgrade, we add a shift value δ to the ephemeral randomness where δ is determined after all the R's for the b messages are published. Specifically, let R_i^u be the randomness contributed by party $i \in [n]$ in the u-th copy where $u \in [b]$. After QUAL is determined[14], the randomness for the u-th messages is $R^u := \sum_{i \in \mathsf{QUAL}} R_i^u$. Then each party computes locally

$$\delta = \mathsf{Hash}\big(S, \{(R^u, M^u) : u \in [b]\}\big) \text{ and } \Delta = \delta \cdot G,$$

and the parties use $R^u + \Delta$ and $\phi^u + \delta$ for the signatures.

The intuition here is that $\mathcal{A}$ has very low probability of making a random-oracle query (S, R', M) with $R' = R^u + \Delta$ before δ is computed. The simulator, instead of guessing the queries of the form (S, R, M) for e, now guesses queries for δ of the form $(S, \{(R^u, M^u) : u \in [b]\})$.

The simulator aborts if the guess was wrong, or if $\mathcal{A}$ made a query $(S, R^{u\prime}, M^u)$ with the correct $R^{u\prime} = R^u + \delta \cdot G$ before step 2. (For each $u \in [b]$ the last event happens with probability at most $1/q$, and the total probability for the b messages can be upper bounded by the union bound.) If it did not abort, then the simulator knows all the $R^{u\prime}$'s, so it is free to program the random-oracle answers to all these queries $(S, R^{u\prime}, M^u)$.

[14] Even though multiple polynomials are shared, we assume that the agreement protocol (Fig. 1, Step 2) guarantees the same QUAL for all of them.

A.4 Threshold Schnorr with a Super-Invertible Matrix

When using the super-invertible optimization, we still have each dealer D_i sharing a single polynomial $\mathbf{H}_i$, but we can derive b ephemeral randomness polynomials since we have at least b honest parties in QUAL.

However, having b honest parties in QUAL also means that we cannot use the same simulation strategy as above: Recall that in Appendix A.2.1, the simulator picks an honest party i^* at random, hoping that it will end up in QUAL. Sim simulates the actions of i^* differently from all the other honest partie, embedding a component that depends on S in the randomness R_{i^*}. Trying to do the same here, the simulator would have to guess not one but b honest parties from QUAL. Since in the asynchronous setting we cannot ensure that honest parties end up in QUAL, the probability of guessing correctly is $1/\binom{n}{b}$.

Even worse, the simulator cannot know the linear combination to use for various quantities that the adversary expects to see, until QUAL is determined: without the super-invertible optimization, the linear combination of the contributions from parties in QUAL was always a sum. But with this optimization, the linear combination is determined by a sub-matrix of the super-invertible Ψ, where the column corresponding to each party depends on QUAL.

To overcome these issues, we let the simulator embed S in the randomness $\mathbf{H}_i$ of *all the honest parties*, so it no longer needs to guess which honest parties will end up in QUAL. Moreover, we modify the protocol to includes QUAL in the hash query for δ. This way, once the simulator sees the random-oracle query, it knows QUAL and can determine which party will correspond to what column of Ψ.

A.5 Threshold Schnorr with Packing

The main difference induced by the packed variant is that the degrees of $\mathbf{F}$ and $\mathbf{H}$ are no longer the same. When describing the simulator, and in particular the way it sets up the randomness polynomials $\mathbf{H}_i$ of the honest parties, we can no longer just describe the relevant randomness scalars r and deduce the unique polynomial which is consistent with them. Instead, we construct the polynomials $\mathbf{H}_i$ in a form that would let the simulator cancel out the terms that depends on the secret key (that it doesn't know), and deduce the r's from them.

A.6 Putting it All Together

Combining all the modifications above, we next describe the threshold signature protocol with the super-invertible matrix optimization and packing.

Protocol inputs. $M^{1,1}, \ldots, M^{b,a}$ messages to be signed. Each party i holds a Shamir share of s, denoted as $\sigma_i = \mathbf{F}(i)$ where F has degree-$d = t + a - 1$ polynomial and $\mathbf{F}(0) = \mathbf{F}(-1) = \cdots = \mathbf{F}(1-a) = s$. Also, $S = s \cdot G$, $S_i = \sigma_i \cdot G$ as well as the Feldman commitment to $\mathbf{F}$ are public.

Protocol outputs. Schnorr signatures $(R^{u,v}, \phi^{u,v})$ for messages $M^{u,v}$ for all $u \in [b], v \in [a]$.

Protocol steps.

1. The parties run step 1 and step 2 of JF-DKG protocol, sharing polynomials $\mathbf{H}_i$ of degree $d' = t + 2a - 2$, and the agreement in step 2 ensures $|\mathsf{QUAL}| = t + b$ and $\mathsf{HOLD} = t + d' + 1 =$

$2t+2a-1$. The polynomials $\{\mathbf{H}_i : i \in \mathsf{QUAL}\}$ will be used for generating b ephemeral-randomness polynomials, and we denote $r_{i,v} := \mathbf{H}_i(1-v)$ and $R_{i,v} := r_{i,v} \cdot G$.

2. Let $\Psi = [\psi_i^u]_{i\in\mathsf{QUAL}}^{u\in[b]} \in \mathbb{Z}_p^{b\times(t+b)}$ be a super-invertible matrix.
   For all $u \in [b]$, let $\mathbf{H}^u = \sum_{i\in\mathsf{QUAL}} \psi_i^u \mathbf{H}_i$.
   Each party $j \in \mathsf{HOLD}$ locally computes $\rho_j^u = \mathbf{H}^u(j) = \sum_{i\in\mathsf{QUAL}} \psi_i^u \mathbf{H}_i(j)$.

   From the Feldman commitments $\hat{H}_i$ to $\mathbf{H}_i$ for $i \in \mathsf{QUAL}$, everyone can compute Feldman commitments $\hat{H}^u$ to the $\mathbf{H}^u$'s. For those, anyone can compute the values $\mathbf{H}^u(j)\cdot G$ for all u, j, including $R^{u,v} = \mathbf{H}^u(1-v)\cdot G = \sum_{i\in\mathsf{QUAL}} \psi_i^u \cdot R_{i,v}$.

3. The parties locally compute $\delta := \mathsf{Hash}(S, \mathsf{QUAL}, \{(R^{u,v}, M^{u,v}) : u \in [b].v \in [a]\})$ and $\Delta = \delta \cdot G$. (Note the inclusion of QUAL in this hash query.)

4. For $u \in [b]$, run the packed signature protocol with randomness $\mathbf{H}^u$ and shift scalar δ: Everyone computes $e^{u,v} = \mathsf{Hash}(S, R^{u,v} + \Delta, M^{u,v})$ for all $v \in [a]$. Let $\mathbf{Z}^u$ be the unique degree-$a-1$ polynomials with $\mathbf{Z}^u(1-v) = e^{u,v}$ for all $v \in [a]$. Each party $j \in \mathsf{HOLD}$ sets $\pi_j^{u,v} = \mathbf{H}^u(j) + \mathbf{Z}^u(j)\cdot \mathbf{F}(j)$. Each signature share is verified as $\pi_j^{u,v}\cdot G = R_j^{u,v} + \mathbf{Z}(j)\cdot S_j$.

5. Reconstruct $\mathbf{Y}^u = \mathbf{H}^u + \mathbf{Z}^u \cdot \mathbf{F}$ from $\pi_j^u = \mathbf{Y}^u(j)$ for $j \in \mathsf{HOLD}$. For each $v \in [a]$ let $\phi^{u,v} = \mathbf{Y}(1-v)$, and output the signature as
   $$(R^{u,v} + \Delta, \phi^{u,v} + \delta).$$

Theorem 4. Under the discrete log assumption (Assumption 1), $\Pi_{\mathsf{super,\ pack}}$ (Figure 1) is a secure threshold signature protocol for generating ab signatures in the random oracle model, assuming a malicious adversary corrupting t parties.

Proof. The discrete log challenger randomly samples $s \leftarrow \mathbb{Z}_p$ as the signing secret key and gives the simulator the corresponding public key $S = s \cdot G$. The simulator's task is to find s, by utilizing $\mathcal{A}$. To use $\mathcal{A}$ the simulators needs to simulate for it the following aspects of the protocol:

- The Feldman commitment $\hat{F}$ to the polynomial $\mathbf{F}$ of degree $d = t + a - 1$ that hides the long-term secret key;

- The shares $\sigma_i = \mathbf{F}(i)$ of the long-term secret key for $i \in \mathsf{Corrupt}$;

- The Feldman commitments $\hat{H}_i$ to the polynomial $\mathbf{H}_i$ of degree $d' = t + 2a - 2$ from each party $i \in \mathsf{Honest}$;

- The (encryption of the) shares $\rho_{i,j} = \mathbf{H}_i(j)$ for every $i \in \mathsf{Honest}$ and $j \in \mathsf{Corrupt}$;

- The messages in the agreement protocol that decides on QUAL and HOLD, including the verifiable complaints and the support message;

- The b degree-d' polynomials $\mathbf{Y}_u$, $u \in [b]$, that must be consistent with the shares and with the Feldman commitments $\hat{F}$ and the $\hat{H}_i$'s;

- Sim also must answer all the oracle queries of $\mathcal{A}$, of the forms $(S, \mathsf{QUAL}, \{(R^{1,1}, M^{1,1}), \ldots, (R^{b,a}, M^{b,a})\})$ and (S, R, M).

The simulator needs to first simulate the shares of $\mathbf{F}$ and the Feldman commitments to $\mathbf{F}$, which is done similarly to Appendix A.2.1.

It chooses t random and independent scalars for the shares of the secret key, i.e., $\sigma_i = \mathbf{F}(i)$ for $i \in \mathsf{Corrupt}$, and gives them to $\mathcal{A}$. Now the simulator needs to create the Feldman commitments to F such that the share verification performed by $\mathcal{A}$ will pass. To do this, the simulator uses the public key $S = \mathbf{F}(0) \cdot G = \cdots = \mathbf{F}(1-a) \cdot G$, in conjunction with the t group elements $\sigma_i \cdot G$ for $i \in \mathsf{Corrupt}$: For each $z \in [1-a, 0] \cup \mathsf{Corrupt}$, denote by $\mathbf{I}_z$ the degree-d polynomial satisfying $\mathbf{I}_z(z) = 1$ and $\mathbf{I}_z(y) = 0$ for $y \in [1-a, 0] \cup \mathsf{Corrupt}$ and $y \neq z$. Denoting $\mathbf{I}^* = \mathbf{I}_0 + \mathbf{I}_{-1} + \ldots + \mathbf{I}_{1-a}$ and $F_{\mathsf{Corrupt}} = \sum_{z \in \mathsf{Corrupt}} \sigma_z \mathbf{I}_z$, the simulator defines the polynomial $\mathbf{F} = s \cdot \mathbf{I}^* + \mathbf{F}_{\mathsf{Corrupt}}$, where it knows $\mathbf{F}_{\mathsf{Corrupt}}$ in the clear.

Next, for each party $i \in \mathsf{Honest}$, the simulator picks two random polynomials, $\mathbf{A}_i$ of degree d' and $\mathbf{B}_i$ of degree $a-1$, and computes the commitment to $\mathbf{H}_i$ as

$$\hat{H}_i := \mathbf{A}_i \cdot G - \mathbf{B}_i \cdot \mathbf{I}^* \cdot S.$$

This corresponds to the polynomial $\mathbf{H}_i = \mathbf{A}_i + s \cdot \mathbf{B}_i \cdot \mathbf{I}^*$ (that the simulator does not know in the clear), and to public random elements

$$R_{i,v} = \mathbf{H}_i(1-v) \cdot G = \mathbf{A}_i(1-v) \cdot G - \mathbf{B}_i(1-v) \cdot \mathbf{I}^*(1-v) \cdot S$$

that everyone can compute from $\hat{H}_i$.

Importantly, even though the simulator does not know $\mathbf{H}_i$ in the clear, it can compute $\mathbf{H}_i(j)$ in the clear for all $j \in \mathsf{Corrupt}$, since $\mathbf{I}^*(j) = 0$ and therefore $\mathbf{H}_i(j) = \mathbf{A}_i(j)$. This completes the simulation of step 1 in JF-DKG, and Sim sends to $\mathcal{A}$ the corresponding shares and the commitments.

Let q be the upper bound on the number of random-oracle queries for δ; the simulator chooses a random index $\ell \in [q]$, hoping that this will be the query where $\mathcal{A}$ asked on $(S, \mathsf{QUAL}, \{(R^{u,v}, M^{u,v}) : u \in [b], v \in [a]\})$ that are used for the signatures. All random oracle queries upto and including the ℓ'th query of that form are answered with fresh random scalars from $\mathbb{Z}_p$. Let QUAL^* be the value specified for QUAL in that ℓ'th query, and $R^{*,u,v}$ the groups elements in it. The simulator answers that query with a random $\delta \in \mathbb{Z}_p$, and denote $\Delta = \delta \cdot G$. The simulator aborts if QUAL^* contains less than b honest parties, or if $\mathcal{A}$ made am earlier query $(S, R^{*,u,v} + \Delta, M^{u,v})$ for any $u \in [b], v \in [a]$. (The latter event can be upper bounded by the union bound, in Lemma 4.1 we show that it happens with probability at most abq/p. For the former event, we show bellow that it happens with probability at most $1 - 1/q$.)

The set QUAL^* implies the matrix $\Psi = [\psi_i^u]_{i \in \mathsf{QUAL}^*}^{u \in [b]}$. After Ψ and Δ and the $R^{*,u,v}$'s are defined, the Simulator defines the degree-$(a-1)$ polynomial

$$\mathbf{Z}^u \;=\; \sum_{i \in \mathsf{QUAL}^* \cap \mathsf{Honest}} \psi_i^u \mathbf{B}_i. \tag{2}$$

Random-oracle queries of the form $(S, R^{*,u,v} + \Delta, M^{u,v})$ are answered with $e^{u,v} = \mathbf{Z}^u(1-v)$. Note that the polynomials $\mathbf{Z}^1, \ldots, \mathbf{Z}^b$ are just random and independent polynomials of degree $a-1$: The $\mathbf{B}_i$'s are random and independent, and also independent of the view of $\mathcal{A}$ (due to the $\mathbf{A}_i$'s that hide them). Moreover, there are at least b of them in $\mathsf{QUAL}^* \cap \mathsf{Honest}$, and the matrix Ψ is super-invertible. Hence, the $e^{u,v}$'s are random and independent, and so they are a legitimate programming of the random oracle. All other queries are still answered with fresh random $\mathbb{Z}_p$ elements.

Next the simulator runs the agreement protocol on behalf of honest parties; but treating them as honest dealers even if they broadcast garbage ciphertexts for each other. At the end of the agreement protocol, QUAL and HOLD are determined such that $|\mathsf{QUAL}| \geq t+b$, $|\mathsf{HOLD}| \geq 2t+2a-1$. This in turn defines the ephemeral randomness $R^{u,v} = \sum_{i \in \mathsf{QUAL}} \psi_i^u R_{i,v}$ for each $u \in [b], v \in [a]$. If $\mathsf{QUAL} \neq \mathsf{QUAL}^*$ or $R^{u,v} \neq R^{*,u,v}$ for some u, v, then the simulator aborts. Since $\mathcal{A}$ makes at most q oracle queries (and we can assume w.l.o.g. that one of them is $(S, \mathsf{QUAL}, (R^{1,1}, M^{1,1}), \ldots, (R^{b,a}, M^{b,a})))$, then there is at least a $1/q$ chance that it does not abort. Note that since $|\mathsf{QUAL}|$ includes at least b honest parties, then $\mathsf{QUAL} = \mathsf{QUAL}^*$ implies that the first abort event from above did not happen either. Hence, Sim will proceed to the next steps with probability at least $\frac{1}{q}(1 - \frac{abq}{p}) = \frac{1}{q} - \frac{ab}{p}$.

If the simulator did not abort, we denote $\mathbf{H}^u = \sum_{i \in \mathsf{QUAL}} \psi_i^u \mathbf{H}_i$ for every $u \in [b]$. The simulator does not know the $\mathbf{H}^u$'s in the clear (since they depend on the secret key s). But since $\mathbf{H}_i = \mathbf{A}_i + s \cdot \mathbf{B}_i \cdot \mathbf{I}^*$ for $i \in \mathsf{Honest}$, we can write $\mathbf{H}^u$ as

$$\begin{aligned}
\mathbf{H}^u &= \sum_{i \in \mathsf{QUAL}} \psi_i^u \mathbf{H}_i = \sum_{i \in \mathsf{QUAL} \cap \mathsf{Corrupt}} \psi_i^u \mathbf{H}_i + \sum_{i \in \mathsf{QUAL} \cap \mathsf{Honest}} \psi_i^u \mathbf{H}_i \\
&= \underbrace{\sum_{i \in \mathsf{QUAL} \cap \mathsf{Corrupt}} \psi_i^u \mathbf{H}_i + \sum_{i \in \mathsf{QUAL} \cap \mathsf{Honest}} \psi_i^u \mathbf{A}_i}_{=\mathbf{H}^u_{\mathsf{CLR}}} - s \left(\sum_{i \in \mathsf{QUAL} \cap \mathsf{Honest}} \psi_i^u \mathbf{B}_i \right) \cdot \mathbf{I}^*,
\end{aligned}$$

and Sim knows the full $\mathbf{H}^u_{\mathsf{CLR}}$ in the clear.

To simulate step 4 and 5 of $\Pi_{\mathsf{super,\ pack}}$, Sim needs to create in full the signature polynomials $\mathbf{Y}^1, \ldots, \mathbf{Y}^b$ of degree d'. For each $u \in [b]$, the u-th signature polynomial can be written as

$$\begin{aligned}
\mathbf{Y}^u = \mathbf{H}^u + \mathbf{Z}^u \cdot \mathbf{F} &= \mathbf{H}^u_{\mathsf{CLR}} - s \left(\sum_{i \in \mathsf{QUAL} \cap \mathsf{Honest}} \psi_i^u \mathbf{B}_i \right) \cdot \mathbf{I}^* + \mathbf{Z}^u \cdot (\mathbf{F}_{\mathsf{Corrupt}} + s \cdot \mathbf{I}^*) \\
&= \mathbf{H}^u_{\mathsf{CLR}} + \mathbf{Z}^u \cdot \mathbf{F}_{\mathsf{Corrupt}} + s \underbrace{\left(\mathbf{Z}^u - \sum_{i \in \mathsf{QUAL} \cap \mathsf{Honest}} \psi_i^u \mathbf{B}_i \right)}_{=0} \cdot \mathbf{I}^* = \mathbf{H}^u_{\mathsf{CLR}} + \mathbf{Z}^u \cdot \mathbf{F}_{\mathsf{Corrupt}},
\end{aligned}$$

which Sim known in the clear.

This completes the simulation, finally we apply the forking lemma as in Appendix A.2. □

Lemma 4.1 (Union bound for random oracle queries)**.** *Suppose the random oracle outputs elements in a group $\mathbb{G}$ of order p. Let q be the number of queries of the form (S, R, M) that $\mathcal{A}$ made to the random oracle before δ is computed. Let ab be the number of messages to be signed, and denote by $R^{u,v}$, $u \in [b], v \in [a]$, the group elements that are included in the query where δ is computed. Then the probability that $R^{u,v} + \delta \cdot G$ is included in any $\mathcal{A}$'s queries before δ is computed is at most abq/p.*

Proof. Let $\mathcal{R} := \{R \in \mathbb{F} : R \text{ was included in one of the } q \text{ queries}\}$. By definition, we know that $|\mathcal{R}| \leq q$. For any element $X \in \mathbb{G}$, let $\mathcal{R} - X = \{R - X : R \in \mathcal{R}\}$. Then we have $|\mathcal{R} - X| \leq q$ for every $X \in \mathbb{G}$, and therefore:

$$\begin{aligned}
&\Pr[\exists u \in [b], v \in [a] \text{ such that } R^{u,v} + \delta G \in \mathcal{R}] \\
&= \Pr[\exists u \in [b], v \in [a] \text{ such that } \delta G \in \mathcal{R} - R^{u,v}] \\
&\leq \sum_{u \in [b], v \in [a]} \Pr[\delta G \in \mathcal{R} - R^{u,v}] \ \leq\ ab \cdot (q/p).
\end{aligned}$$

□

A.7 Security for the Dynamic Setting

The main difference between the dynamic and the static cases is that in the dynamic case, it is the shareholders that need to generate the signatures at the end, while the dealers hold the shares of the long-term secret key at the beginning. The sets of dealers and shareholders are arbitrary, they can be the same set, disjoint sets, or anywhere in between. Hence, the dynamic setting requires a re-share protocol in order for the dealers to pass the shared secret key to the shareholders. (In the static case the dealers and shareholders are the same set, hence no re-sharing is needed.)

Note that even in the dynamic case we assume that the secret key is uniformly random and cannot be biased by the adversary. This can be implemented by having a trusted party share it in the first place, or using a non-biased DKG protocol (e.g., the protocol from [10] that uses statistically-hiding commitments). The Shamir sharing of that unbiased key is an input to the protocol. While the adversary may bias the new shares if we use Feldman commitments, it cannot bias the key itself.

The simulation proof for the dynamic case is mostly the same as the static case, so we only describe briefly the added components. Specifically, the simulator needs to simulate also the degree-d re-sharing polynomials F_i's from the honest parties. Recall that the simulator in Appendix A.6 generates a Feldman commitment $\hat{F}$ for $\mathbf{F}$, this allows it to compute $S_i\mathbf{F}(i) \cdot G$ for every i. Then it can write $\mathbf{F}_i$ just like it did $\mathbf{F}$, namely $\mathbf{F}_i = \mathbf{F}_{i,\mathsf{Corrupt}} + \mathbf{F}(i) \cdot \mathbf{I}^*$, where it knows $\mathbf{F}_{i,\mathsf{Corrupt}}$ in the clear. The rest of the proof follows, with the simulator setting the re-shared polynomial $\mathbf{F}'$ just like it did the $\mathbf{H}^u$'s (except using the Lagrange coefficients λ_i rather than the matrix entries ψ_i^u).

B Refreshing Packed Secrets

When describing SPRINT, we focused on how to refresh packed secrets in which all the scalars are equal, i.e., a vector of secrets of the form $(s, \ldots, s)$. For our application to high-throughput Schnorr signatures that was enough, since we needed to put the same secret key in all these slots. But other applications may want to maintain shares of more general vectors, of the form $(s_1, s_2, \ldots, s_a)$, where the s_v's can be different from one another. For sake of completeness, we describe here a simple method for refreshing sharing of these more general packed secrets.

Below, let I_u for $u = 1, 2, \ldots, a$ the unique polynomial of degree $a - 1$ satisfying $I_u(1 - u) = 1$ and $I_u(1-v) = 0$ for all $v \in \{1, \ldots, a\}, v \neq u$. Refreshing a packed sharing of $(s_1, s_2, \ldots, s_a)$ roughly consists of sharing a different polynomials, each containing just one of the s_v's, then using the I_u's to combine them into a single packed polynomial. We describe below two variants of this approach: One having each dealer share only a single polynomial, and results in a packed polynomial of the slightly larger degree of $t + 2a - 1$. The other has each dealer share a different polynomials and resulting in a packed polynomial of degree $t + a$ (which is the smallest possible).

B.1 Method One: Sharing One Polynomial

In this method, we maintain the invariant that the vector of secrets $(s_1, \ldots, s_a)$ is shared using a packed degree-$(t + 2a - 2)$ polynomial F with $F(1 - v) = s_v$ for all $v = 1, 2, \ldots, a$. (Note that even though we offer robustness against t corrupted parties and only pack a values, the polynomial that we keep has degree $t + 2a - 2$ and not just $t + a - 1$.)

To refresh, each party re-shares its share via a packed polynomial of degree $t+a-1$. Namely, F_i of degree $t+a-1$ such that $F_i(0)=F_i(-1)=\cdots=F_i(1-a)=\sigma_i$. Each shareholder j gets the sub-share $\sigma_{ij}=F_i(j)$ from each dealer i.

The shareholders then proceed with the usual agreement protocol, agreeing on a set QUAL of qualified dealers with cardinality $|QUAL|=t+a$, and a set HOLD of secret-holding shareholders with cardinality $|\mathsf{HOLD}|\geq 2t+a$, such that all the shareholders in HOLD have valid sub-shares from all the dealers in QUAL.

Next, a shareholder $j\in\mathsf{HOLD}$ first computes a different temporary shares corresponding to a polynomials that hold the a different values. Specifically, for $v=1,2,\ldots,a$ consider the Lagrange coefficients $\{\lambda_{iv}: i\in\mathsf{QUAL}, v=1,\ldots,a\}$, such that for every degree-$(t+a-1)$ polynomial $P(X)$ it holds for all $v=1,\ldots,a$ that $P(1-v)=\sum_{i\in QUAL}\lambda_{i,v}P(i)$. Party j computes for all $v=1,2,\ldots,a$

$$\rho_{j,v}=\sum_{i\in QUAL}\lambda_{i,v}\sigma_{ij}.$$

Let us denote $F'_v=\sum_{i\in QUAL}\lambda_{i,v}F_i$, then clearly all the F'_v's are degree-$(t+a-1)$ polynomials with $\rho_{j,v}=F'_v(j)$, and

$$F'_v(1-v)=\sum_{i\in\mathsf{QUAL}}\lambda_{i,v}F_i(1-v)=\sum_{i\in\mathsf{QUAL}}\lambda_{i,v}F(i)=F(1-v)=s_v.$$

Party P_j computes its final share as $\sigma'_j=\sum_{v=1}^{a}\rho_{j,v}\cdot I_v(j)$. Clearly, this is indeed a share on the polynomial $F'=\sum_{v=1}^{a}I_v\cdot F'_v$ of degree $t+2a-2$, and for all $v=1,2,\ldots,a$ we have

$$F'(1-v)=\sum_{v'=1}^{a}I_{v'}(1-v)\cdot F'_{v'}(1-v)=F'_v(1-v)=s_v,$$

as needed.

We remark that by construction, the polynomial F' cannot reveal more that what's implied by the polynomials $F'_1,\ldots,F'_a$, which in turn only reveal the s_v's.

B.2 Method Two: Sharing a Polynomials

In this method, we maintain the invariant that the vector of secrets $(s_1,\ldots,s_a)$ is shared using a packed degree-$(t+a-1)$ polynomial F with $F(1-v)=s_v$ for all $v=1,2,\ldots,a$.

Each dealer D_i has a share $\sigma_i=F(i)$, and they prepare a degree-t polynomials $F_{i,1},\ldots,F_{i,a}$, random subject to the condition that $F_{i,v}(1-v)=\sigma_i$ for all $v\in[a]$. D_i shares these polynomials, with shareholder P_j receiving $\sigma_{i,j,v}=F_{i,v}(j)$ for all $v\in[a]$.

The shareholders agree on sets HOLD, $\mathsf{QUAL}_1,\ldots,\mathsf{QUAL}_v$ such that for all $v\in[a]$, all the shareholders in HOLD have valid shares of $F_{i,v}$ from all the dealers in QUAL_v, and in addition $|\mathsf{HOLD}|\geq t+a$ and $|\mathsf{QUAL}_v|\geq t+1$ for all $v\in[a]$.

For all $v\in[a]$, let $\{\lambda_{i,v}: i\in\mathsf{QUAL}_v\}$ be the Lagrange coefficients for recovering $F(1-v)$ from $\{F(i): i\in\mathsf{QUAL}_v\}$. Each shareholder $P_j\in\mathsf{HOLD}$ computes a share $\sigma'_{j,v}=\sum_{i\in\mathsf{QUAL}_v}\lambda_{i,v}\sigma_{i,j,v}$. This share is $\sigma'_{j,v}=F'_v(j)$, where F'_v is the degree-t polynomial $F'_v=\sum_{i\in\mathsf{QUAL}_v}\lambda_{i,v}F_{i,v}$. As usual, it is easy to see that we have $F'_v(1-v)=F(1-v)=s_v$ for all $v\in[a]$.

Finally, each shareholder P_j sets $\sigma'_j=\sum_{v=1}^{a}I_v(j)\cdot\sigma'_{j,v}$. Clearly, we have $\sigma'_j=F'(j)$ for the polynomial $F'=\sum_{v=1}^{a}I_v\cdot F'_v$ of degree $t+a-1$, and $F'(1-v)=s_v$ for all $v\in[a]$.

C Faster Multiplication by a Super-Invertible Matrix

Recall that to compute the group elements $R_{u,v} = r_{u,v} \cdot G$ (which is needed in order to compute the Schnorr challenges $e_{u,v} = \mathsf{Hash}(S, R_{u,v}, M_{u,v})$), the signers must perform a matrix-multiplication "in the exponent",

$$\left[R_{u,v}\right]_{u\in[b],v\in[a]} \;=\; \Psi \times \left[\mathbf{H}_i(1-v)\cdot G\right]_{i\in\mathsf{QUAL},v\in[a]}. \tag{3}$$

It is therefore beneficial to make the matrix $\Psi \in \mathbb{Z}_p^{b\times(b+t)}$ as sparse as we can, while ensuring that it remains super-invertible. Namely, any $b \times b$ sub-matrix of Ψ must be invertible. To that end, we use the construction

$$\Psi = (I|H),$$

where I is the $b \times b$ identity matrix and H is a $b \times t$ *hyper-invertible* matrix [2]. Recall that H is hyper-invertible if every square sub-matrix of it is invertible (not just any $b \times b$ sub-matrix). For example, any sub-matrix of a Vandermonde matrix is hyper-invertible.

Lemma 4.2. If H is a hyper-invertible $b \times t$ matrix and I_b is the $b \times b$ identity matrix, then $\Psi = (I_b|H)$ is super-invertible.

Proof. Consider any $b \times b$ sub-matrix $\Psi' = (I'|H')$ of Ψ, it consists of some number k of columns from I and $b-k$ columns from H. By swapping rows we can move all the single 1's from I' to the top k rows, and then by column reduction we can zero out all the other entries in these k top rows without affecting any of the $b-k$ bottom rows. This does not change the rank, but result in a matrix of the form

$$\Psi'' = \left[\begin{array}{c|c} I_k & 0 \\ \hline 0 & H'' \end{array}\right],$$

where I_k is the $k \times k$ identity matrix and H'' is some $(b-k) \times (b-k)$ sub-matrix of H. Since H is hyper-invertible then H'' is invertible, and therefore so is Ψ'' and therefore Ψ'. □

We remark that this matrix Ψ is "as sparse as possible", in that no row of a super-invertible matrix can have more than $b-1$ zeros.

Using this construction, we can reduce the number of scalar-by-element product when computing Eq. (3) to only $a \cdot b \cdot t$ rather than $a \cdot b \cdot (t+b)$, even when using the naive matrix-multiplication algorithm. More saving are possible by using Strassen's algorithm or FFT-based techniques.

D The Parameter-Finding Utility

```
#!/usr/bin/env python3
"""
Computation of committee sizes for secrets-on-blockchain
Modify the parameters at the bottom of the file
"""

import math
from typing import Callable, Tuple
from scipy.stats import binom

# ==============
# Core functions
# ==============

def binary_search(f: Callable[[int], float], max_val: float, start: int, end: int) -> int:
    """
    Find the max integer i between start and end so that f(i) <= max_val
    Assumes that f is non-decreasing
    start/end included in search
    """
    while start != end:
        mid = int((start + end + 1) / 2)
        if f(mid) <= max_val:
            start = mid
        else:
            end = mid - 1
    return start

def liveness_error(n: int, t: int, alpha: float, a: int) -> float:
    """
    Return the liveness error for alpha total fraction corrupted
    and packing parameter a
    """
    # Pr[number honest parties selected < 2*t + 2*a - 1]
    # = Pr[number hones parties selected <= 2*t + 2*a - 2]
    return binom(n, 1 - alpha).cdf(2 * t + 2 * a - 2)

def safety_error(n: int, t: int, alpha: float) -> float:
    """
    Return the safety error for alpha total fraction corrupted
    """
    # Pr[number of corrupted parties selected > t]
    return binom(n, alpha).sf(t)

def threshold_and_safety_error_for_liveness(n: int, max_liveness_error: float,
                                            alpha_safety: float, alpha_liveness: float, a: int) \
        -> Tuple[int, float]:
    """
    Find the optimal threshold t and associated safety error
    to achieve the given liveness error
    """
    # find t for liveness to hold
    t = binary_search(lambda t: liveness_error(n, t, alpha_liveness, a), max_liveness_error, 1, n)
    # return the associated safety error
    return t, float(safety_error(n, t, alpha_safety))

def find_nt(max_n: int, max_liveness_error: float, max_safety_error: float,
            alpha_liveness: float, alpha_safety: float, a: int) -> Tuple[int, int]:
    """
    Find the optimal number of parties n and threshold t
    to achieve the given max_liveness_error and max_safety_error
    assuming a fraction alpha_safety of malicious parties for safety
    and a fraction alpha_liveness of malicious parties for liveness

    The packing parameter is a. a=1 is no packing
    """
    # we first search for the minimum n such that the safety error is below max_safety_error
    # note that we use negative n here because we want the min and binary_search looks for the max
    n = -1 * binary_search(
        lambda n: threshold_and_safety_error_for_liveness(
            -n, max_liveness_error, alpha_safety=alpha_safety, alpha_liveness=alpha_liveness, a=a
        )[1],
        max_safety_error, -max_n, -1
    )

    t, _ = threshold_and_safety_error_for_liveness(
        n, max_liveness_error, alpha_safety=alpha_safety, alpha_liveness=alpha_liveness, a=a
    )

    return n, t

# ============
def print_setting(max_n: int, max_liveness_error: float, max_safety_error: float,
                  alpha_liveness: float, alpha_safety: float, a: int) -> None:
    """
    See function find_nt
    """
    n, t = find_nt(max_n, max_liveness_error, max_safety_error,
                   alpha_safety=alpha_safety, alpha_liveness=alpha_liveness, a=a)
    print(f"number of parties:         n = {n}")
    print(f"threshold for safety:      t = {t}")
    print(f"packing parameter:         a = {a}")
    print(f"corrupt fraction liveness: alpha = {round(alpha_liveness * 100, 2)} %")
    print(f"corrupt fraction safety:   alpha = {round(alpha_safety * 100, 2)} %")
    err = float(liveness_error(n, t, alpha_liveness, a))
    print(f"liveness error:            {err:.2e} = 2^{round(math.log(err, 2), 2)}")
    err = float(safety_error(n, t, alpha_safety))
    print(f"safety error:              {err:.2e} = 2^{round(math.log(err, 2), 2)}")

    # find percentage of corrupt people so safety error goes down to max_safety_error2
    # max_safety_error2 = 2**-40
    print("safety errors for higher alpha:")
    max_safety_error2 = 2 ** -10

    alpha2 = 1e-4 * binary_search(
        lambda alpha: safety_error(n, t, alpha / 1e4),
        max_safety_error2,
        0, 1e4
    )
    delta = (alpha_safety - alpha2) / 8
    for i in range(8):
        alpha = alpha_safety - (i + 1) * delta
        err = float(safety_error(n, t, alpha))
        print(f"  for alpha = {round(alpha * 100, 2):02.2f} %:    "
              f" {err:.2e} = 2^{round(math.log(err, 2), 1)}")

# ===========================
def main():
    # ==========
    # Parameters
    # ==========

    a1 = 64  # packing parameter for optimistic setting
    a2 = 40  # packing parameter for pessimistic setting
    max_n = 2 ** 12  # upper bound for the search for n

    alpha_safety = 0.2  # total fraction of corrupted parties, for safety - denoted f in the paper

    alpha_liveness1 = 0.05  # total fraction of corrupted parties, for liveness (optimistic)
    alpha_liveness2 = 0.2  # total fraction of corrupted parties, for liveness (pessimistic)

    max_liveness_error1 = 0.005  # 99.5%, optimistic setting
    max_liveness_error2 = 2 ** -11  # 99.95%, pessimistic setting

    max_safety_error = 2 ** -80

    # =======
    # Display
    # =======

    print("Optimistic Setting:")
    print_setting(max_n, max_liveness_error1, max_safety_error, alpha_liveness1, alpha_safety, a1)
    print("=========================================================")
    print()
    print("Pessimistic Setting:")
    print_setting(max_n, max_liveness_error2, max_safety_error, alpha_liveness2, alpha_safety, a2)
    print("=========================================================")
    print()

if __name__ == "__main__":
    main()
```

D.1 Example Parameters

```
$ python math/committee_sizes.py
Optimistic Setting:
number of parties:         n = 676
threshold for safety:      t = 250
packing parameter:         a = 64
corrupt fraction liveness: alpha = 5.0 %
corrupt fraction safety:   alpha = 20.0 %
liveness error:            4.35e-03 = 2^-7.85
safety error:              5.89e-25 = 2^-80.49
safety errors for higher alpha:
  for alpha = 21.43 %:     1.00e-20 = 2^-66.4
  for alpha = 22.86 %:     4.88e-17 = 2^-54.2
  for alpha = 24.29 %:     7.71e-14 = 2^-43.6
  for alpha = 25.72 %:     4.42e-11 = 2^-34.4
  for alpha = 27.16 %:     1.01e-08 = 2^-26.6
  for alpha = 28.59 %:     9.86e-07 = 2^-20.0
  for alpha = 30.02 %:     4.43e-05 = 2^-14.5
  for alpha = 31.45 %:     9.71e-04 = 2^-10.0
=========================================================

Pessimistic Setting:
number of parties:         n = 992
threshold for safety:      t = 336
packing parameter:         a = 40
corrupt fraction liveness: alpha = 20.0 %
corrupt fraction safety:   alpha = 20.0 %
liveness error:            4.12e-04 = 2^-11.24
safety error:              5.95e-25 = 2^-80.48
safety errors for higher alpha:
  for alpha = 21.17 %:     8.78e-21 = 2^-66.6
  for alpha = 22.35 %:     4.01e-17 = 2^-54.5
  for alpha = 23.52 %:     6.29e-14 = 2^-43.9
  for alpha = 24.69 %:     3.70e-11 = 2^-34.7
  for alpha = 25.86 %:     8.79e-09 = 2^-26.8
  for alpha = 27.03 %:     9.01e-07 = 2^-20.1
  for alpha = 28.21 %:     4.24e-05 = 2^-14.5
  for alpha = 29.38 %:     9.63e-04 = 2^-10.0
=========================================================
```