Threshold Cryptography as a Service (in the Multiserver and YOSO Models)

Fabrice Benhamouda
Shai Halevi
Hugo Krawczyk
fabrice.benhamouda@gmail.com
shaih@alum.mit.edu
hugokraw@gmail.com
Algorand Foundation
New York, USA

Alex Miao
alexmia@sas.upenn.edu

USA

Tal Rabin
talr@seas.upenn.edu
University of Pennsylvania
USA
Algorand Foundation
New York, USA

ABSTRACT

We consider large deployments of threshold cryptographic services that can run in traditional multi-server settings and, at a much larger scale, in blockchain environments. We present a set of techniques that improve performance and meet the requirements of settings with large number of servers and high rate of threshold operations. More fundamentally, our techniques enable threshold cryptographic applications to run in more challenging decentralized permissionless systems, such as contemporary blockchains. In particular, we design and implement a novel threshold solution for the recently introduced YOSO (You Only Speak Once) model. The model builds on ever changing, unpredictable committees that perform ephemeral roles in a way that evades targeting by attackers and enables virtually unlimited scalability in very large networks. Our solution allows for the maintenance of system-wide keys that can be generated, used and proactivized as needed. The specific techniques build on optimized protocols for multi-secret multi-dealer verifiable secret sharing and their adaptation to the YOSO model.

We demonstrate the practicality of our solutions by reporting on an end-to-end implementation of a proactive re-sharing protocol in the YOSO model, showing benchmarks for committees of sizes up to 500 nodes. For traditional multi-server settings, we obtain significant speedups in settings where dealers process many secrets simultaneously (say, to generate or proactivize many keys at the same time), e.g., we show 5× improvements relative to classical Pedersen VSS for 15 servers and 50 secrets, and 48× for 500 servers and 1000 secrets.

CCS CONCEPTS

• **Security and privacy → Cryptography**; *Key management*; *Public key (asymmetric) techniques*; *Digital signatures*; *Public key encryption*;

CCS '22, November 7–11, 2022, Los Angeles, CA, USA
© 2022 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-9450-5/22/11…$15.00
https://doi.org/10.1145/3548606.3559397

KEYWORDS

Threshold cryptography, Verifiable secret sharing, blockchain, YOSO model

ACM Reference Format:
Fabrice Benhamouda, Shai Halevi, Hugo Krawczyk, Alex Miao, and Tal Rabin. 2022. Threshold Cryptography as a Service (in the Multiserver and YOSO Models). In *Proceedings of Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS '22).* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3548606.3559397

1 INTRODUCTION

In threshold cryptography, cryptographic functions are computed by running a protocol among a set of parties, in a way that preserves secrecy and integrity as long as "most parties" are honest. A typical example is a system of n servers that produces signatures in a secure way as long as no more than a threshold t of the servers is corrupted. Threshold cryptography has been studied intensively for more that 30 years. While deployment in the real world has been limited so far, modern trends such as the outsourcing of computation to the cloud and decentralized applications provide new motivation and opportunities to deploy threshold cryptography.

Threshold cryptography as a service. Efficient threshold cryptography systems for small sets of servers are known, but most systems in the literature scale rather poorly to large deployments. In this work we develop techniques that make very large deployment feasible (our techniques show significant benefits in smaller settings too, see Section 5). We envision a large set of servers that provides cryptographic services to applications by computing cryptographic functions with threshold security. We refer to such deployments as *threshold cryptography as a service.*

Traditional examples include threshold encryption services to protect valuable information, signature services to support sensitive applications such as (software) code signing or certification authorities, and many more. A more demanding setting for threshold services arise with decentralized permissionless systems, such as contemporary blockchains. Example include running committees of validators, implementing cross-chain bridges and randomness beacons, using threshold encryption to protect transactions against front-running, protecting custodial services, and many more. These systems are open, allowing anyone to join and run a node, creating the potential for the existence of a large number of nodes. Thus, the protocols that are executed in these systems must perform well even when large number of nodes wish to participate.

One technique for dealing with such large systems rely on smaller committees for doing the work, that are sub-sampled from the entire population of nodes. However, these committees must still be large enough to ensure that they are "mostly honest" with high probability. This results in committees with dozens to hundreds (possibly more) of parties.

YOSO protocols. Recent years saw a new style of cryptographic protocols, where long-term parties are replaced by stateless ephemeral ones. Such protocols are motivated by the general shift towards "serverless computing", as well as by specific blockchain-based architectures that emphasize player replaceability [9].

The latter approach envisions computation carried via a sequence of *randomly-chosen anonymous committees* where a committee receives secret inputs from previous committees, carries computations based on these inputs, and and eventually produces an output. Anonymity of committee members precludes the attacker from targeting these entities. However, the anonymity is broken when the party "speaks" (produces output). Thus, before producing output a committee member must erase all of its secret state. Any information in this state that is needed for future actions of the protocol needs to be shared to a different committee that will complete these actions when required in the future.

This model was recently formalized by Gentry et al. [21] and called *YOSO (You Only Speak Once)*. In this model, long-lived nodes are replaced by ephemeral stateless entities called *roles*.[1] These roles are addressable (i.e. other roles can send them messages), and each role has access to all the messages that were sent to it (or broadcast) in the past. Other than this communication transcript, however, each role begins its life without any state. A role performs some computation, sends (or broadcast) a single batch of messages, and then immediately self destructs (capturing the erasure of state as described above).

While this serverless model is very appealing from a system perspective, realizing secure protocols of this type is very challenging. Some early protocols were described, e.g., in [5, 6, 11], and Gentry et al. provided in [21] a general feasibility result, showing that every function can in principle be computed in this way. But obtaining YOSO protocols that are efficient enough to be useful is still a major challenge. And aiming at large committees makes this challenge a whole lot harder. As we explain later, adapting standard cryptographic protocols to the YOSO model typically entails at least an $n\times$ increase in complexity, where n is the number of parties.

1.1 Our contributions and techniques

In this work we describe optimized threshold protocols that are efficient enough to support large committees, in the traditional multi-server case and even in the YOSO model. Our solutions are all built from a basic primitive of *multi-secret/multi-dealer verifiable secret sharing (MSMD-VSS).* Namely, we show efficient protocols that allow n' dealers, each with m secrets, to share all their secrets among a set of n shareholders.

[1]An example of a role could be "party #3 in committee #2".

First, we show how to amortize the classical Pedersen VSS protocol [30] to the setting of many secrets that are shared by many dealers, in the traditional (non-YOSO) setting. Very roughly, we consider a 3-dimensional matrix with axis corresponding to dealers, secrets, and shareholders, and perform consistency checks across the multiple dimensions to save on work. This protocol is described in Section 3. Our implementation (Section 5) shows 5× improvements relative to classical Pedersen VSS for 15 servers and 50 secrets, and 48× for 500 servers and 1000 secrets.

The main technical challenge of this work is in adapting that basic protocol to the YOSO model, and using it to implement a protocol for "proactive" refreshing of secrets (e.g., as needed in the architecture of Benhamouda et al [5]).

To see some of the issues that are involved, recall that the Pedersen VSS protocol (recalled in Fig. 2) is based on the paradigm of accusations and resolution. Very roughly, it features a dealer that sends shares of its secret to the shareholders, and publishes some associated verification information. The shareholders compare their shares to the verification information, and publish an accusation against the dealer if they don't match. The dealer then responds to accusations by publishing the secret shares that it sent to the accusing shareholders. Finally, every participants can decide whether to accept the published responses or to disqualify the dealer.

Consider now what happens when we try to convert such a protocol to the YOSO model. We need to adapt it so that the dealer only sends a single batch of messages, and shareholders are completely passive and do not send any messages at all. This may seem impossible: How can the shareholders accuse if they cannot speak at all, and how can the dealer respond if it cannot speak a second time? To solve this conundrum, we let the dealer send its messages utilizing two intermediary committees: A *verification committee* that will run checks and broadcast accusations, and a *response committee* that responds to these accusations.

The response committee's (called "future broadcast" in [21]) design is as follows: Whenever a dealer $\mathcal{D}_i$ sends a message to a verifier $\mathcal{V}_j$, it also shares that message among the members of the response committees using Shamir sharing. If $\mathcal{V}_j$ accuses $\mathcal{D}_i$, then the response committee simply publishes their shares for everyone to see. Since it must be the case that either $\mathcal{V}_j$ or $\mathcal{D}_i$ are bad, then there is no security loss in revealing that message.

The verification committee's operations are harder: A dealer $\mathcal{D}_i$ that wants to relay a message M via the verifiers to some shareholder $\mathcal{P}_\ell$, cannot directly send it to any of the verifiers, since any one of them can be bad. Instead, *it has to share that message among all the verifiers*, and have them reconstruct that message to $\mathcal{P}_\ell$. Yet, the sharing must still allow the verifiers to check the shared value and accuse $\mathcal{D}_i$ if it is bad, all by speaking only once.

These ideas can be used to "compile" protocols to the YOSO model, but doing so naively is very expensive. Indeed, it seems to require *two extra levels of secret sharing*: every message $\mathcal{D} \rightarrow \mathcal{P}$ in the original protocol must be shared among the verifiers, and each of these shares must again be shared among the responders. Hence, trying to compile an n-party protocol may increase the complexity by a factor of n^2.

Reducing this overhead is the main technical contribution of this work. On the one hand, we show in Section 4 a collection of techniques that can reduce the expansion factor from $n^2\times$ to $n\times$.

Then we relay on the scaling features of our MSMD-VSS protocol to rein in the complexity: we show that this $n\times$ factor can be expressed as having the dealers share n secrets each (rather than just one), and then use our MSMD-VSS protocol to do so much more efficiently than running n copies of the original protocol.

The combination of all these techniques allows our protocol to remain feasible with many hundreds of parties and even in the YOSO model. This makes our protocol a promising step towards an actual feasible implementation of the architecture from [5].

To demonstrate the practicality of our YOSO resharing protocol, we provide an end to end implementation of it and report performance results. Our code is written in Go and portable C without assembler optimization, using a modified version of libsodium [16] with some new optimizations. With $n = 513$ parties, the protocol takes less than 2 minutes of single-threaded computation per party (when run over an in-memory communication layer). Since the workload per party is highly parallelizable, a multi-core version would scale almost linearly in number of cores and would reduce even more the computation time per party. In addition, each party broadcasts less than 100MB of data.

For the non-YOSO MSMD-VSS protocol, we provide performance results that were obtained from running micro-benchmarks on the various steps. These results show that our amortization techniques provide significant speedup (see above) over a naive implementation of the basic Pedersen protocol.

1.2 Related Work

Optimizing VSS and MPC via batch verification has been examined extensively, especially in information-theoretic settings [1–4, 14, 15, 26]. One limitation of the approach in these works is that it requires a larger honest majority (at least $n \geq 3t + 1$). More critically, these information-theoretic protocols lose much of their benefits when augmented with computational elements to produce *public keys*, corresponding to the secrets or their shares, as needed in many applications, particularly those that motivate our work. Moreover, porting these protocols to the YOSO setting seems very hard. While information-theoretic YOSO protocols are theoretically feasible [21], they typically feature complexity of $O(n^6)$ or more, which is too expensive to be usable in practice.

Some recent works on efficient computational (non-YOSO) VSS include, for example, [7] and [27]. The former describe a very efficient VSS, but *where the shareholders can only recover $s \cdot G$ in the group*, rather than the secret s itself. This may be enough for some applications, but is too restrictive for others (such as threshold signatures or decryption). The latter is more general but it is based on Paillier encryption and significantly less efficient.

A very relevant recent work is the VSS protocol of Gentry et al. [22]. That work is motivated similarly to ours, namely, they aim at feasible threshold protocols for the YOSO model. But it features a different solution approach: Rather than adapt an interactive protocol to the YOSO model, they describe a completely non-interactive VSS protocol. Namely, the dealer(s) just broadcast encrypted shares to the shareholders, and provide NIZK proofs that these shares are valid. They use a combination of lattice-based batched encryption and DL-based proof systems to obtain rather efficient NIZKs. While potentially feasible even for very large committees, their protocol is a lot slower than ours. For example, running their protocol with $n = 512$ parties, verifying *each NIZK proof* reportedly takes 17.4 seconds. This means that each shareholder must work $17.4 \cdot 512 \approx 8900$ seconds (about 2.5 hours) to verify all the proofs. In contrast, our protocol with 513 parties would requires about two minutes to complete the entire verifiable sharing. We also mention a strawman implementation of a non-interactive proactive VSS suitable to the YOSO model presented in [21], based on threshold Paillier, but highly inefficient and not suited for practice.

1.3 Organization

We recall background material in Section 2, and then describe our non-YOSO amortized VSS protocol in Section 3. Our YOSO protocols are described in Section 4, and we show some initial implementation results in Section 5. All proofs are deferred to the full version.

2 NOTATIONS AND BACKGROUND

Notation. For an integer n, we denote $[n] = \{1, 2, \dots n\}$ and $[[n]] = \{0, 1, 2, \dots, n\}$. We consider a hard-discrete-logarithm group $\mathbb{G}$ of prime order q, written additively, with a designated generator $G \in \mathbb{G}$. The number of shareholders is usually denoted n, and the bound on the number of faulty parties is t. Secrets are usually denoted by lowercase $s \in \mathbb{Z}_q$, and we use uppercase $S \in \mathbb{G}$ to denote the corresponding group elements, $S = sG$. We often use σ or lowercase z to denote shares and set $Z = zG$. We extend the group additive notations to vectors as follows:

- For a vector $\vec{z} = (z_1, \dots, z_n) \in \mathbb{Z}_q^n$ and an element $X \in \mathbb{G}$, we denote $\vec{z}X = (z_1X, \dots, z_nX) \in \mathbb{G}^n$. Similarly for $z \in \mathbb{Z}_q$ and $\vec{X} = (X_1, \dots, X_n) \in \mathbb{G}^n$, we denote $z\vec{X} = (zX_1, \dots, zX_n) \in \mathbb{G}^n$.
- For same-dimension vectors $\vec{z} = (z_1, \dots, z_n) \in \mathbb{Z}_q^n$ and $\vec{X} = (X_1, \dots, X_n) \in \mathbb{G}^n$ their pointwise product is $\vec{z} \odot \vec{X} = (z_1X_1, \dots, z_nX_n) \in \mathbb{G}^n$, and their inner product is $\vec{z} \bullet \vec{X} = \sum_{i=1}^n z_iX_i \in \mathbb{G}$.

These notations extend also to matrix-vector products.

2.1 Multi-Dealer Verifiable Secret Sharing

We recall informally the Multi-Dealer VSS notion that is central to our protocols and point to some of its applications.

VSS A (t, n) Verifiable Secret Sharing (VSS) [10] is a multi-party protocol with two phases: Dealing and Reconstruction. The dealing phase involves a dealer $\mathcal{D}$ and n shareholders $\mathcal{P}_1, \dots, \mathcal{P}_n$, and the reconstruction phase involves $t + 1$ (or more) shareholders. $\mathcal{D}$'s input for the Dealing phase is a secret s; at the end of the phase, either all honest shareholders disqualify the dealer or otherwise each honest shareholder ends with a share σ_i. In the reconstruction phase, subsequent to a non-disqualified dealing, each shareholder provides its share σ_i as input, and they reconstruct (output) the secret s. Assuming at most $t < n/2$ dishonest parties, this protocol provides the following security guarantees: (a) If the dealer is honest then it is not disqualified; (b) If the dealer is honest then at the end of dealing no set of t shareholders has any information about s; and (c) If the dealer is not disqualified, then at the end of dealing there is a unique value s which is guaranteed to be reconstructed in every execution of the reconstruction protocol with at least $t + 1$

honest shareholders. Our solutions in this work are based on the classic Pedersen VSS protocol, see Section 2.6.

MD-VSS. In a Multi-Dealer VSS (MD-VSS), there are n' dealers $\mathcal{D}_1, \cdots, \mathcal{D}_{n'}$, and n shareholders $\mathcal{P}_1, \ldots, \mathcal{P}_n$. Each $\mathcal{D}_i, i \in [n']$, runs a VSS with the n shareholders on a secret s_i where all n' VSS protocols are run simultaneously.

At the end, each shareholder $\mathcal{P}_j$ outputs a local share which is *a linear combination*[2] of the shares received from non-disqualified dealers. A common example is just the sum of all the shares of the non-disqualified dealers. Additionally, many applications require that the dealers and/or parties generate and verify public commitments to the dealers' secrets and/or to the shares computed by the shareholders, see some examples below.

The linear combination is the same for all shareholders, it is specified by the application running the MD-VSS and depends on the set of non-disqualified dealers. By the properties of VSS, all honest shareholders agree on which dealers are disqualified. In the text we formalize this linear combination by a mapping $L : 2^{[n']} \to \mathbb{Z}_q^{n'}$, that for any non-disqualified set $\mathbb{Q} \subseteq [n']$ yields the linear combination $\vec{\lambda} = (\lambda_1, \ldots, \lambda_{n'}) = L(\mathbb{Q})$ that the shareholders use. We require that the i'th entry of $L(\mathbb{Q})$ is zero for any disqualified dealer, $i \notin \mathbb{Q}$ (namely honest shareholder only use shares of non-disqualified dealers). Thus, the parties compute $z_j = \sum_{i \in [n']} \lambda_i \sigma_{ij}$.

Applications of MD-VSS. A basic use-case for MD-VSS is distributed key generation (DKG) for dlog systems [19]. In this application, each dealer $\mathcal{D}_i$ chooses its secret s_i as a random field element and the linear combination applied by shareholders can be as simple as addition of all shares received from non-disqualified dealers. The resulting shares serve as the shareholders' local private keys and the implicit shared secret serves as the global private key of the system. In addition, all honest parties learn the public keys corresponding to the local and global private keys. Such DKG protocol can also be used for producing ephemeral values in Schnorr or ECDSA signatures (the 'r' element in a (r, s) signature pair).

A more involved application of MD-VSS is for refreshing shares in a proactive security system [23, 28] (that we use in essential ways in our YOSO protocols from Section 4). This setting considers n' shareholders holding shares $s_1, \cdots, s_{n'}$ of a *global secret* s. At some point, these shareholders need to reshare the secret s into fresh shares (without changing s), so that any knowledge of old shares does not help in learning information about the new ones. Namely, an attacker that learns t shares from the old sharing and t shares from the new one, learns nothing about s. In applications where shares act as the parties' private keys (e.g., in a threshold signature scheme), shareholders must also learn each other's public keys as well as the global public key of the system.

Proactive refreshing can be implemented with MD-VSS using the technique from [20]. Roughly, the old shareholders act as dealers in MD-VSS, and the new shareholders use Lagrange interpolation for their linear combination. See more details in Section 4.

2.2 Pedersen Commitments

Pedersen commitments [29] rely on a setup phase in which two random generator $G_0, G_1 \in \mathbb{G}$ are made public, where the discrete logarithm between them is unknown. To commit to $z \in \mathbb{Z}_q$, the committer chooses a uniform $r \in \mathbb{Z}_q$ and outputs the group element $C = rG_0 + zG_1$. To open, the committer reveals z and r.

This is extended to multi-valued commitments to m-dimensional vectors as follows: The setup is extended to use $m+1$ random generators with unknown pairwise discrete logarithm, $G_0, G_1, \ldots, G_m \in \mathbb{G}$. To commit to $(z_1, \ldots, z_m) \in \mathbb{Z}_q^m$, the committer chooses a random scalar $z_0 \in \mathbb{Z}_q$ and outputs the commitment element $C = \sum_{\ell=0}^{m} z_\ell G_\ell$. To open, the committer reveals all the z_ℓ's, $\ell \in [[m]]$. The following are well-known properties of this scheme.

- Perfectly hiding: Given $C \in \mathbb{G}$ and $(z_1, \ldots, z_n) \in \mathbb{Z}_q^n$, there is exactly one value $z_0 \in \mathbb{Z}_q$ for which $C = \sum_{\ell=0}^{m} z_\ell G_\ell$.
- Computationally binding: If the discrete logarithm problem is hard in $\mathbb{G}$ then it is infeasible to find two different vectors $(z_0, z_1, \ldots, z_m), (z'_0, z'_1, \ldots, z'_m) \in \mathbb{Z}_q^m$ such that $\sum_{\ell=0}^{m} z_\ell G_\ell = \sum_{\ell=0}^{m} z'_\ell G_\ell$.
- Linearity: Given vectors $\vec{z}, \vec{z}' \in \mathbb{Z}_q^m$ and their Pedersen commitments C, C' with randomness z_0, z'_0, respectively, the element $C + C' \in \mathbb{G}$ is a valid commitment to $\vec{z} + \vec{z}' \in \mathbb{Z}_q^m$ with randomness $z_0 + z'_0 \in \mathbb{Z}_q$.

2.3 Vector Commitments

Vector commitment [8] is a commitment scheme to a vector that allows the committer to open individual entries of the vector without opening or revealing information on other entries. A trivial vector commitment solution is to commit to each entry individually, but more efficient solutions are known, using Merkle trees or accumulators. Our YOSO protocol from Section 4 uses vector-commitment, but that component has only a negligible effect on the complexity of the overall protocol. Hence in our implementation we used the trivial solution above.

2.4 Linearity Testing

Our protocols need parties to verify that certain vectors of group elements $\vec{C} = (C_1, \ldots, C_n)$ belong to some known linear space "in the exponent". (In particular, we will need to check that $C_j = f(j)G$ for some low-degree polynomial f, namely, where the linear space is defined by a Vandermonde matrix.) This condition can be tested by checking that $[h]\vec{C} = \vec{0}$ where $[h]$ is the parity-check matrix of that linear space over $\mathbb{Z}_q$. But checking this directly is expensive, as it requires as many scalar-point multiplications as there are entries in h. Instead, the verifier can choose a random vector $\vec{e}$ over $\mathbb{Z}_q$, compute $\vec{u} = \vec{e}[h]$, then check the inner product $\vec{u} \bullet \vec{C} = 0$ using a single n-multi-scalar-point multiplications. Note that if $\vec{C}$ does not satisfy the linearity condition, this test succeeds only with probability $1/q$.

If there are multiple verifiers, they can either each choose their own randomness $\vec{e}$ (and the prover will have to answer to each separately) or all use the same randomness (e.g., using a random oracle).

2.5 NIZK-POK for Discrete Logarithm

Schnorr-type proofs of knowledge for discrete logarithm are extremely well-studied and useful primitives for cryptographic protocols, see for example [12, 13] and references within.

[2] It is assumed that secrets and shares are field elements.

Parameters: generators $G_1, \ldots, G_m$, $H_1, \ldots, H_m$
Public input: elements $Z_1, \ldots, Z_m$, Y
Prover input: scalars $z_1, \ldots, z_m$

(1) Prover chooses at random $r_1, \ldots, r_m \in \mathbb{Z}_q$, sends to verifier $R_\ell = r_\ell G_\ell$ and $S_\ell = r_\ell H_\ell$
(2) Verifier sends to prover a random $c \in \mathbb{Z}_q$.
(3) Prover responds with $z'_\ell = r_\ell + c \cdot z_\ell, \ell \in [m]$.
(4) Verifier picks uniformly at random $e_1, \ldots, e_\ell \in \mathbb{Z}_q$. Then it checks that

$$\textstyle\sum_\ell c e_\ell Z_\ell + \sum_\ell e_\ell R_\ell + cY + \sum_\ell S_\ell = \sum_\ell z'_\ell e_\ell G_\ell + \sum_\ell z'_\ell H_\ell.$$

($c, e_1, \ldots, e_\ell$ can be chosen in $[2^\lambda]$ with λ the security parameter rather than uniform in $\mathbb{Z}_q$.)

Figure 1: Σ-protocols to prove knowledge of discrete-logarithms $Z_\ell = z_\ell G_\ell, \ell \in [m]$, in a multi-valued setting. With the text in gray, it also proves that $\sum z_\ell H_\ell = Y$ for the same z_ℓ's. These proofs can be made non-interactive via the Fiat-Shamir transformation.

For our purposes, we will be proving knowledge of (possibly multi-valued) commitments that satisfy some linear relations. Fig. 1 describes two simple cases used in our protocols (in the multi-value setting). First, given $G_1, \ldots, G_m, Z_1, \ldots, Z_m \in \mathbb{G}$, prove knowledge of $z_1, \ldots, z_m \in \mathbb{Z}_q$ such that $Z_\ell = z_\ell G_\ell, \ell \in [m]$. Second, the protocol is augmented to additionally prove that the same z_ℓ's also satisfy $\sum_\ell z_\ell H_\ell = Y$ (for some public $H_1, \ldots, H_m, Y \in \mathbb{G}$). Note that for large enough values of m, these proofs can be made more efficient by using a few large m-multi-scalar-point multiplications rather than similar number of individual scalar-point multiplications.

2.6 Pedersen Verifiable Secret Sharing

In Fig. 2 we present a slight variation of Pedersen VSS [30], that implements the VSS functionality using (t, n)-Shamir sharing, commitments to the shares (instead of the coefficients) and the Linearity Testing (Section 2.4). The protocol uses a broadcast channel and private channels from the dealer to all the shareholders. (The use of index i in $\mathcal{D}_i$ and related shares is for consistency with the notation in multi-dealer settings in subsequent sections.)

Using the commitments for verifying the parties actions, the protocol has an accusation-response round that guarantees that either the dealer is disqualified or otherwise the honest shareholders hold correct shares. Honest dealers are never disqualified and honest shareholders agree on whether the dealer is or is not disqualified. The protocol achieves information-theoretic secrecy (with up to $t < n/2$ corrupt shareholders) thanks to the properties of the Shamir secret sharing and the perfect hiding of Pedersen commitments. Additionally, the commitments also serve as a way to guarantee correct reconstruction of the secret.

Note, that as the dealer commits to the shares there is no longer a commitment to the secret. There are two possible scenarios: 1. the commitment to the secret already exists from another portion of the protocol. In that case, the linearity test needs to include the commitment to the secret as well. 2. No such commitment exists, though it can be computed from the commitment to the shares. In this case, what will be done in actuality depends on the application.

Parameters: Integers t, n, $n \geq 2t + 1$. Group $\mathbb{G}$ of order q and two generators $G_0, G_1 \in \mathbb{G}$.
Inputs: Dealer $\mathcal{D}_i$ has private input $s_i \in \mathbb{Z}_q$.

1. Share distribution. $\mathcal{D}_i$ chooses two random polynomials $f_i(\cdot), r_i(\cdot)$ of degree t s.t. $f_i(0) = s_i$:

$$f_i(z) = a_{i0} + a_{i1}z + \cdots + a_{it}z^t, \ r_i(z) = b_{i0} + b_{i1}z + \cdots + b_{it}z^t,$$

($a_{i0} = s_i$ and the other a_{ik}, b_{ik} are random in $\mathbb{Z}_q$).

- $\mathcal{D}_i$ computes the shares $\sigma_{ij} = f_i(j), \sigma'_{ij} = r_i(j)$ for $j = 1, \ldots, n$ and sends $\sigma_{ij}, \sigma'_{ij}$ to $\mathcal{P}_j$.
- $\mathcal{D}_i$ broadcasts $C_{ij} = \sigma_{ij}G_1 + \sigma'_{ij}G_0$ for $j \in [n]$.

2. Accusations. Shareholder $\mathcal{P}_j$ does the following:

- Execute the Linearity Testing (Section 2.4) on the points $C_{i\ell}$, for $\ell = 1, \ldots, n$.
- Verify the share from the dealer by checking

$$\sigma_{ij}G_1 + \sigma'_{ij}G_0 = C_{ij} \text{ (over } \mathbb{G}). \qquad (1)$$

If either check fails, $\mathcal{P}_j$ broadcasts an accusation against $\mathcal{D}_i$.

3. Response. For every $\mathcal{P}_j$ that accused $\mathcal{D}_i$, the dealer broadcasts values $\sigma_{ij}, \sigma'_{ij}$ that satisfy Eq. (1).

4. Disqualification. All $\mathcal{P}_j$ *disqualify* dealer $\mathcal{D}_i$ if

- either $\mathcal{D}_i$ received more than t accusations in Step 2,
- or $\mathcal{D}_i$ failed to answer an accusation with values that satisfy Eq. (1).

(Note that thanks to the broadcast channel, all honest shareholders agree on whether $\mathcal{D}_i$ is disqualified or not.)

If the dealer is not disqualified, then each $\mathcal{P}_j$ that sent an accusation replaces the corresponding shares by the values $\sigma_{ij}, \sigma'_{ij}$ that were broadcast in Step 3 (that satisfy Eq. (1)).

Reconstruction. If the dealer was not disqualified, then shareholders can reconstruct the secret s_i by pooling their shares $\sigma_{ij}, \sigma'_{ij}$, which are checked using Eq. (1). Any $t + 1$ shares σ_{ij} that pass verification can be used to interpolate s_i.

Figure 2: A Variant of the Pedersen VSS Protocol.

3 AMORTIZED VSS

An important contribution of our work is the ability to amortize the sharing of many secrets with only a minor increase in complexity. We start by describing the approach for amortizing multiple Pedersen VSS protocols for one dealer, then extend it to handle the multi-dealer case (which is what most applications require).

3.1 Multi-Secret, Single-Dealer Pedersen VSS

Consider a single dealer $\mathcal{D}_i$ (the subscript i is for consistency with later notation) that holds m secrets $s_{i\ell}, \ell \in [m]$, and wants to verifiably share all of them *to the same set of n shareholders*, $\mathcal{P}_1, \ldots, \mathcal{P}_n$.

We show how to accomplish this with better efficiency than running the VSS protocol m times. The idea in a nutshell is to start with multiple Pedersen VSS protocol from Fig. 2, and combine for each shareholder all the shares directed to that shareholder using a multi-Pedersen commitment as in Section 2.2. This can be seen as the batching techniques for secret sharing [4], computed in the exponent.

In more detail, the dealer generates an independent (t, n) Shamir-sharing for each of the secrets $s_{i,\ell}$, and in addition for another random secret s_{i0}. Then consider the share matrix $\mathbb{A}_i \in \mathbb{Z}_q^{(m+1)\times n}$, where $\mathbb{A}_i[\ell, j] = f_{i\ell}(j)$ is the share of party j in the sharing of $s_{i\ell}$:

$$\mathbb{A}_i = \begin{pmatrix} \sigma_{i10} & & \sigma_{in0} \\ \sigma_{i11} & & \sigma_{in1} \\ & \ddots & \\ \sigma_{i1m} & & \sigma_{inm} \end{pmatrix} \quad \begin{matrix} \sigma_{ij0} = \sum_{k=0}^{t} a_{ik0} j^k \ (a_{i00} = s_{i0}) \\ \sigma_{ij1} = \sum_{k=0}^{t} a_{ik1} j^k \ (a_{i01} = s_{i1}) \\ \vdots \\ \sigma_{ijm} = \sum_{k=0}^{t} a_{ikm} j^k \ (a_{i0m} = s_{im}) \end{matrix}$$

The dealer generates and broadcasts a Pedersen multi-value commitment to each column of the matrix, using σ_{ij0} as randomness. Namely, $C_{ij} = \sum_{\ell=0}^{m} \sigma_{ij\ell} G_\ell \in \mathbb{G}$, where the G's are public random generators. The dealer also sends over a private channel to each shareholder $\mathcal{P}_j$ its shares for all the secrets, $\sigma_{ij\ell} = f_{i\ell}(j)$, $\ell = 0, 1, \ldots, m$.

Each shareholder $\mathcal{P}_j$ compares its own shares to the public commitments C_{ij}, by checking that indeed $C_{ij} = \sum_{\ell=0}^{m} \sigma_{ij\ell} G_\ell$. In addition, everyone verifies that the check values $C_{i1}, \ldots, C_{in}$ lie on a degree-t polynomial, using the linearity test from Section 2.4.

If both $\mathcal{D}_i$ and $\mathcal{P}_j$ are honest then this verification will pass. To see that, let $e_\ell = DL_G(G_\ell)$ (for some generator G), then

$$C_j = \sum_{\ell=0}^{m} \sigma_{ij\ell} \cdot G_\ell = \sum_{\ell=0}^{m} f_{i\ell}(j) \cdot e_\ell G = F_i(j) G,$$

where $F_i(\cdot) = \sum_\ell e_\ell \cdot f_{i\ell}(\cdot)$ is a degree-t polynomial.

If verification fails at $\mathcal{P}_j$, $\mathcal{P}_j$ broadcasts a complaint against $\mathcal{D}_i$. This is resolved as usual by $\mathcal{D}_i$ broadcasting all the shares $\sigma_{ij\ell}$ that it sent to $\mathcal{P}_j$, and everyone checking that $C_{ij} = \sum_{\ell=0}^{m} \sigma_{ij\ell} G_\ell$.

The dealer $\mathcal{D}_i$ is disqualified either if it was accused by more than t shareholders, or if it failed to respond to an accusation by broadcasting valid shares. Since disqualifying the dealer is a deterministic function of the content of the broadcast channel, then all honest parties will agree on whether or not $\mathcal{D}_i$ was disqualified.

To reconstruct all the $s_{i\ell}$'s, the participating shareholders pool their shares $\sigma_{ij\ell}$, which are checked against the check values C_{ij}. Any $t + 1$ shares σ_{ij} that pass verification can be used to interpolate $s_{i\ell}$. (Note that share verification relies on the fact that *all the secrets* $s_{i0}, \ldots, s_{im}$ *are reconstructed together* and hence all the $\sigma ij\ell$'s are available.) Details of the protocol appear in Fig. 3.

Soundness of this protocol is established by showing that if the dealer was not disqualified, then the shares of all the honest parties must agree with some set of t-degree polynomials $f_{i\ell}(\cdot)$, $\ell = 0, 1, \ldots, m$. In more detail, we show that finding shares $\{\sigma_{ik\ell}\}_{k,\ell}$ that pass verification in the protocol (i.e., $\sum_{\ell=0}^{m} \sigma_{ij\ell} \cdot G_\ell = C_{ij}$) but do not all lie on the same degree-d polynomials $f_{i\ell}(\cdot)$, implies finding a non-trivial representation of the element $0 \in \mathbb{G}$ in the bases $G_0, G_1, \ldots, G_m$. This only happens with a negligible probability, assuming that finding discrete logarithms in $\mathbb{G}$ is hard. See proof in full version.

Parameters: Integers m, n, t with $n \geq 2t + 1$, group $\mathbb{G}$ of prime order q and generators $G_0, G_1, \ldots, G_m \in \mathbb{G}$.
Inputs: Dealer $\mathcal{D}_i$ has input secrets $s_{i1}, \ldots, s_{im} \in \mathbb{Z}_q$.

Dealing. Dealer $\mathcal{D}_i$:

(1) Chooses a random value $s_{i0} \in \mathbb{Z}_q$.
(2) For all $\ell \in [[m]]$, sets $a_{i0\ell} = s_{i\ell}$, chooses at random $a_{ik\ell} \in \mathbb{Z}_q$ for all $k \in [t]$. (This defines the polynomials $f_{i\ell}(x) = \sum_{k=0}^{t} a_{ik\ell} x^k$ over $\mathbb{Z}_q$.)
(3) For all $j \in [n], \ell \in [[m]]$, computes shares $\sigma_{ij\ell} = f_{i\ell}(j)$ and sends to $\mathcal{P}_j$ over private channels.
(4) For all $j \in [n]$, computes $C_{ij} = \sum_{\ell=0}^{m} \sigma_{ij\ell} G_\ell \in \mathbb{G}$ and broadcasts $C_{i1}, \ldots, C_{in}$.

Verifications and Accusations. Each shareholder $\mathcal{P}_j$:

(1) Verify that $C_{i1}, \ldots, C_{in}$ lie on a degree-t polynomial using the linearity test (Section 2.4)
(2) Check that $C_{ij} = \sum_{\ell=0}^{m} \sigma_{ij\ell} G_\ell$.

If any check fails, $\mathcal{P}_j$ broadcasts an *accusation* against $\mathcal{D}_i$.

Response.

(1) Each $\mathcal{D}_i$ broadcasts the shares $\sigma_{ij\ell}, \ell \in [[m]]$ for every $\mathcal{P}_j$ that accused it.
(2) Each $\mathcal{P}_j$ that accused $\mathcal{D}_i$ replaces its shares $\sigma_{ij\ell}$ with the ones that $\mathcal{D}_i$ broadcasted.

Dealer disqualification. Dealer $\mathcal{D}_i$ is disqualified if

- either it was accused by more than t shareholders,
- or it failed to respond to some accusation with shares $\sigma_{ij\ell}$ such that $C_{ij} = \sum_{\ell=0}^{m} \sigma_{ij\ell} G_\ell$.

Figure 3: Fast Amortized Multi-secret, Single-dealer VSS

3.2 Multi-Secret, Multi-dealer VSS

In Fig. 4 we describe our multi-secret, multi-dealer VSS (MSMD-VSS) protocol; a protocol with n' dealers $\mathcal{D}_1, \ldots, \mathcal{D}_{n'}$, each sharing m secrets $s_{i1}, \cdots, s_{im}$, in parallel, to a set of n shareholders $\mathcal{P}_1, \ldots, \mathcal{P}_n$ (for security claims we will consider t of the shareholders to be corrupted with $t < n/2$).

We can think of this protocol as working on m "slices", $\ell \in [m]$. For each slice ℓ, we denote by $\sigma_{ij\ell}$ the share of $\mathcal{P}_j$ for the secret $s_{i\ell}$ of $\mathcal{D}_i$. $\mathcal{P}_j$ then takes a linear combination of the shares that it received from all the (non-disqualified) dealers in that slice, $z_{j\ell} = \sum_i \lambda_i \sigma_{ij\ell}$ (using the same linear combination in all the slices). The linear combination, that depends on the set of non-disqualified dealers and the application, is formalized using the mapping $L : \mathbb{Q} \mapsto \vec{\lambda}$ as described in Section 2.1.

Moreover, for later share verification, $\mathcal{P}_j$ also produces public values $Z_{j\ell} = z_{j\ell} G_\ell$, $\ell \in [m]$, to which we often refer as shareholder's public keys. The protocol ensures correctness and verifiability of these public values. Furthermore, a linear combination "in the exponent" of these values generates a public key Z_ℓ for a combined *global secret* of that slice, $z_\ell = \sum_{i \in \mathbb{Q}} \lambda_i s_{i\ell}$. For example, in a DKG application we get a global private key z_ℓ and the corresponding *global public key* Z_ℓ for each slice $\ell \in [m]$.

The protocol uses a broadcast channel and a private communication channel between the dealers and the shareholders: For each

Parameters: Integers m, n, t with $n \geq 2t+1$, group $\mathbb{G}$ of prime order q and generators $G_0, G_1, \ldots, G_m \in \mathbb{G}$. Also a mapping $L : 2^{[n']} \to \mathbb{Z}_q^{n'}$ s.t. for all $\mathbb{Q} \subseteq [n']$ and all $j \notin \mathbb{Q}$, $L(\mathbb{Q})_j = 0$.

Inputs: Each dealer $\mathcal{D}_i$ has input secrets $s_{i1}, \ldots, s_{im} \in \mathbb{Z}_q$.

Sharing and Dealer Disqualification:

(1) Each dealer $\mathcal{D}_i$ ($i \in [n']$) executes the Multi Secret Single Dealer protocol of Fig. 3, resulting in shares $\sigma_{ij\ell}$ and commitment to them C_{ij}.

(2) Let $\mathbb{Q} \subset [n']$ be the set of qualified dealers, i.e. the ones that were not disqualified.

Shareholder public values. Let $\vec{\lambda} = L(\mathbb{Q}) \in \mathbb{Z}_q^{n'}$. Each shareholder $\mathcal{P}_j$ does the following:

(5) Computes $z_{j\ell} = \sum_{i \in \mathbb{Q}} \lambda_i \sigma_{ij\ell}$ for all $\ell \in [[m]]$.

(6) Computes and broadcasts $Z_{j\ell} = z_{j\ell} G_\ell$ for all $\ell \in [[m]]$.

(7) For each $Z_{j\ell}$, broadcast a NIZK-POK of the corresponding $z_{j\ell}$, using the protocol from Fig. 1.

Shareholder disqualifications. Every shareholder $\mathcal{P}_{j'}$ does the following for each shareholder $\mathcal{P}_j$, $j \neq j'$:

(8) Verifies the NIZK-POK of $Z_{j\ell}$, for all $\ell \in [[m]]$, and checks that $\sum_{\ell=0}^{m} Z_{j\ell} = \sum_{i \in \mathbb{Q}} \lambda_i C_{ij}$

(9) If verification fails for them then $\mathcal{P}_j$ is disqualified.

Let $\mathbb{Q}' \subseteq [n]$ be the set of qualified shareholders, namely, those that were not disqualified. (If $|\mathbb{Q}'| < t+1$ then abort.)

Reconstruction and global public key. The shares $z_{j\ell}$ of any subset of $t+1$ shareholders from $\mathbb{Q}'$ (which can be verified against the public values $Z_{j\ell}$) can be used to interpolate the global secrets z_ℓ.

Moreover, the $Z_{j\ell}$'s themselves can be used to compute the global public keys via $Z_\ell = z_\ell G_\ell = \sum_j \gamma_j Z_{j\ell}$, where the γ_j's are the corresponding Lagrange interpolation coefficients.

Figure 4: Fast Amortized Multi-secret, Multi-dealer VSS

dealer, we first run the Multi Secret Single Dealer protocol of Fig. 3. The parties define the set of qualified dealers. Then each shareholder computes its share of the global secrets z_ℓ as $z_{j\ell} = \sum_{i \in \mathbb{Q}} \lambda_i \sigma_{ij\ell}$. Finally, the shareholders engage in a protocol to generate the public values $Z_\ell = z_\ell G_\ell$ as follows: Each shareholder broadcasts the elements $Z_{j\ell} = z_{j\ell} G_\ell$ for all ℓ, along with a NIZK-POK for the corresponding $z_{j\ell}$. Then everyone verifies these NIZK proofs, and also uses the commitments to the columns of the matrices $\mathbb{A}_i$ (cf. Section 3.1) to check that these $Z_{j\ell}$'s lie on degree-t polynomials. If all these checks pass, then everyone can compute the public values Z_ℓ by interpolating "in the exponent" the $Z_{j\ell}$'s from $t+1$ shareholders.

Proofs of soundness and security of the protocol (the latter based on definitions in Section 3.3) are presented in the full version.

3.3 Functionality and Simulation

We codify the security properties that we need from our MSMD-VSS protocol by an "ideal functionality" $\mathcal{F}_{MSMD.VSS}$. See Fig. 5 for a schematics of the functionality and the corresponding simulator.

Parameters. The functionality interacts with n' dealers $\mathcal{D}_i$ and n shareholders $\mathcal{P}_j$, some of which are controlled by the ideal-world adversary. Let $\mathbb{B}$ be the set of adversarially-controlled dealers and $\mathbb{B}'$ are the adversarially-controlled shareholders, and assume that $\|\mathbb{B}'\| \leq t$.

The functionality is parametrized by the same group $\mathbb{G}$ of order q with generators $G_0, \ldots, G_m$ and the same mapping $L : 2^{[n']} \to \mathbb{Z}_q^{n'}$ from the high-level application.

Inputs. The functionality receives m "original secrets" $s_{i1}, \ldots, s_{im} \in \mathbb{Z}_q$ from each honest dealer $\mathcal{D}_i$, and optionally also some of the dishonest dealers.

Qualified dealers. The functionality also receives from the adversary a set $\mathbb{P} \subseteq \mathbb{B}$ of dealers under its control that should be included in the qualified set. All these dealers $i \in \mathbb{P}$ must supply original secrets $s_{i\ell}$. The set of qualified dealers will include $\mathbb{P}$ and the honest dealers, $\mathbb{Q} = \mathbb{P} \cup \overline{\mathbb{B}}$.

Global secrets. The functionality computes the linear combination coefficients $\vec{\lambda} = L(\mathbb{Q})$, and computes the global secrets $z_\ell = \sum_{i \in \mathbb{Q}} \lambda_i s_{i\ell}$ for all $\ell \in [m]$.

Shares and verification values. The simulator also sends to the functionality the alleged shares of the bad shareholders, $\{\sigma_{ij\ell} : i \in [n'], j \in \mathbb{B}', \ell \in [m]\}$. The functionality prepares sharing of all the global secrets, which is consistent with those shares.

Specifically, for each qualified dealer $i \in \mathbb{Q}$ it computes the unique polynomials $f_{i\ell}(\cdot)$ consistent with the bad shareholders' view and the secrets $s_{i\ell}$, then sets $g_\ell = \sum_{i \in \mathbb{Q}} \lambda_i f_{i\ell}$. It then sets $z_{j\ell} = g_\ell(j)$ for all $j \in [n]$ and $\ell \in [m]$, and sends to the adversary the public values $Z_{j\ell} = z_{j\ell} G_\ell$ for the honest shareholders $j \notin \mathbb{B}'$ and all $\ell \in [m]$.

Qualified shareholders and output. The adversary responds with a set $\mathbb{P}' \subseteq \mathbb{B}'$ of shareholders under its control that should be included in the qualified set.

The functionality sets $\mathbb{Q}' = \mathbb{P}' \cup \overline{\mathbb{B}'}$. It returns to each qualified shareholder $j \in \mathbb{Q}'$ the shares $z_{j\ell} = g_\ell(j)$ for all $\ell \in [m]$. It also sends to everyone the verification values $Z_{j\ell} = z_{j\ell} G_\ell$, for all $j \in \mathbb{Q}'$ and $\ell \in [m]$.

Is $\mathcal{F}_{MSMD.VSS}$ meaningful? The functionality above is not very "ideal", in that the adversary $\mathcal{S}$ has a lot of influence over the output shares $\{z_{j\ell}\}$. In what sense, then, does it provide meaningful secrecy guarantees?

In the full version we prove two lemmas. The first says that the view in the sharing phase is independent of the original secrets shared by the honest dealers. The second says that the entire protocol provides information-theoretic security for the original secrets: Any two sets of original secrets that result in the same set of global secrets, will induce the same probability distribution over the adversary's view of the entire protocol. (This means that we could rewrite the functionality above to sample a new set of original secrets that results in the same global secrets, and use that other set to compute the shares $z_{j\ell}$.)

Of course, whether or not this guarantee is enough depends on the application. In our main motivation applications, where the global secrets are random secret keys (or shares thereof), it turns out to be enough.

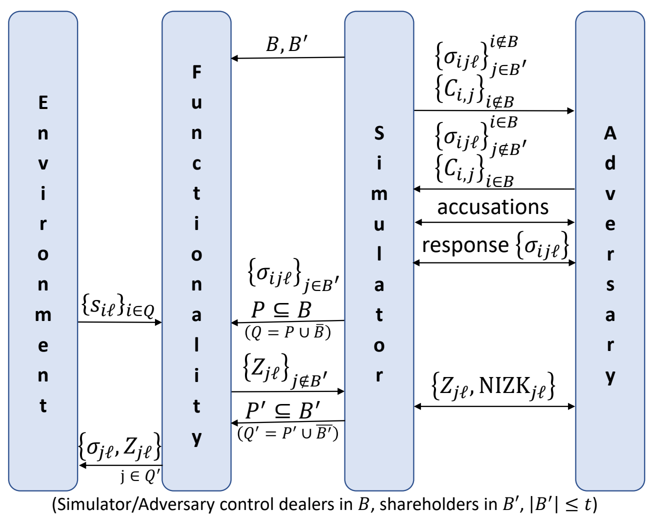

Environment
Functionality
Simulator
Adversary
B, B'
$\{\sigma_{ij\ell}\}^{i\notin B}_{j\in B'}$
$\{C_{i,j}\}_{i\notin B}$
$\{\sigma_{ij\ell}\}^{i\in B}_{j\notin B'}$
$\{C_{i,j}\}_{i\in B}$
accusations
response $\{\sigma_{ij\ell}\}$
$\{s_{i\ell}\}_{i\in Q}$
$\{\sigma_{ij\ell}\}_{j\in B'}$
$P \subseteq B$
$(Q = P \cup \bar{B})$
$\{Z_{j\ell}\}_{j\notin B'}$
$P' \subseteq B'$
$(Q' = P' \cup \overline{B'})$
$\{Z_{j\ell}, \mathrm{NIZK}_{j\ell}\}$
$\{\sigma_{j\ell}, Z_{j\ell}\}$
$j \in Q'$
(Simulator/Adversary control dealers in B, shareholders in B', $|B'| \le t$)

Figure 5: Schematics of the MSMD-VSS functionality $\mathcal{F}_{MSMD.VSS}$ and the simulator

Analysis of the MSMD-VSS protocol from Fig. 4 including the proof that it realize the functionality above, appears in the full version.

3.4 Complexity

We briefly analyze the number of operations carried by the different parties in the protocol as well as the bandwidth requirements which are the main complexity measures for this type of protocols. Computations are composed mostly by m-multi-scalar-point multiplication opertions (namely, expressions of the form $\sum_{\ell=0}^{m} x_\ell G_\ell$ for $x_\ell \in \mathbb{Z}_q$ and $G_\ell \in \mathbb{G}$) plus some individual scalar-point multiplications of the form xG. For simplicity, we will count an m-multi-scalar-point multiplication as m scalar-point multiplications although the former are significantly faster than the latter (and this speed up is central to our practical performance).

Sharing and Creation of $\mathbb{Q}$. **Sharing.** Each dealer needs to compute the n "column commitments" $C_{ij} = \sum_{\ell=0}^{m} \sigma_{ij\ell} G_\ell \in \mathbb{G}$ that requires a single m-multi-scalar-point multiplication. The total number of scalar-point multiplications is thus $n(m+1)$ per dealer.

The dealer broadcasts the n group elements C_{ij} and sends $m+1$ scalars in $\mathbb{Z}_q$ to each shareholder, the total bandwidth over secret channels is $n(m+1)$ scalars for each dealer.

Accusations. Each shareholder $\mathcal{P}_j, j \in [n]$ computes two inner-products to check each dealer: one for the linearity test and the other for testing $C_{ij} = \sum_{\ell=0}^{m} \sigma_{ij\ell} G_\ell \in \mathbb{G}$. The total number of scalar-point multiplications per shareholder in this step is therefore $n'(n+m+1)$. The bandwidth is upto n' accusation broadcasts per shareholder.

Response. Each dealer broadcasts $m+1$ scalars for each accusation. We can have upto t accusations (above that the dealer is automatically disqualified), so this step can consume broadcast bandwidth of upto $t(m+1)$ scalars.

Dealer disqualification. For every accusation $\mathcal{P}_j \to \mathcal{D}_i$, every other shareholder $\mathcal{P}_{j'}$ needs to compute one m-multi-scalar-point multiplication or m scalar-point multiplications. As there can be upto t accusations per dealer, this may require upto $tn'(m+1)$ more scalar-point multiplications for each shareholder. In the worst case, this is by far the most time-consuming step in this protocol. Below we show that when dealers communicate privately with shareholders via public key encryption over the broadcast channel (as is the case in our applications in later sections), the complexity of resolving accusations is no more than $(n'+t)$ m-multi-scalar-point multiplications per shareholder.

Verification values. Each shareholder $\mathcal{P}_j$ performs $m+1$ scalar-point multiplications to compute the $Z_{j\ell}$'s, and $m+1$ more to prepare the NIZK proofs for them. Hence a total of $2m+2$ scalar-point multiplications. In terms of bandwidth, each shareholder broadcasts $2m+2$ group elements ($Z_{j\ell}$'s and $R_{j\ell}$'s). See Fig. 1[3].

Shareholder disqualification. Every shareholder $\mathcal{P}_{j'}$ must verify the values of every other shareholder $\mathcal{P}_j$. This requires verifying the NIZK ($2m+2$ scalar-point multiplications per $\mathcal{P}_j$), and $\sum_{i\in\mathbb{Q}} \lambda_i C_{ij}$ ($|\mathbb{Q}| \le n'$ scalar-point multiplications per $\mathcal{P}_j$). Hence the total number of scalar-point multiplications in this step is upto $n(n'+2m+2)$.

Summing up the above, we get the following total complexity:

Multiplications. Each dealer computes a total of $n(m+1)$ scalar-point multiplications. If there are no accusations, then each shareholder computes

$$n'(n+m+1) + 2m + 2 + n(n' + 2m + 2)$$

scalar-point multiplications. With accusations, in the worst case, each shareholder may need to do up to $tn'(m+1)$ more scalar-point multiplications, but not more than $(n'+t)$ m-multi-scalar-point multiplications with the optimization from xyz.

Broadcast bandwidth. Each dealer broadcasts n group elements if there are no accusations, and upto $t(m+1)$ more scalars if there are accusations. Each shareholder broadcasts upto n' accusations, $2m+2$ group elements and $m+1$ scalars. The total broadcast bandwidth over the entire protocol (i.e., the number of "broadcast units" that each participants must listen to) is $n(n'+3m+3)$ if there are no accusations, and upto $n't(m+1)$ more if there are many accusations (or only $(n'+t)(m+1)$ with the below optimization).

Point-to-point bandwidth. Each dealer sends $n(m+1)$ scalars over these channels, and each shareholder listens to $n'(m+1)$ scalars.

Comparison to naive Pedersen VSS. Using $m \cdot n'$ runs of single-dealer/single-secret Pedersen VSS would require that each shareholder computes roughly $mn't$ scalar-point multiplications, which is roughly a factor of t more than in our protocol. Our protocol also does slightly better in terms of bandwidth, saving roughly a factor of two on both broadcast and point-to-point bandwidth.

3.4.1 *Faster handling of accusations.*

As described, the protocol from Fig. 4 has worst-case complexity $O(n^3)$ if there are many accusations (assuming $m, n', t = \Theta(n)$). This can be improved to $O(n^2)$ by using the broadcast channel (together with PKI) to implement the private channels between dealers and shareholders, Specifically, the dealer can use standard committing encryption over broadcast to send the shares to the $\mathcal{V}_j$'s. For every accusation $\mathcal{V}_j \to \mathcal{D}_i$, the dealer can open the encryption, and everyone can check if the

[3] We use the Fiat-Shamir transformation, so the NIZK challenges c and the e_ℓ's are computed as a hash of the prover's message.

shares that were sent were valid. This means that whenever an accusation is resolved, either the dealer or the verifier will be disqualified. Hence at most $t + n'$ accusations would ever need to be checked, so the overall complexity of handling them all is reduced to $O(n^2)$.

3.5 Further Improvements

We describe optimizations that can further improve the amortized cost of a multi-secret VSS. We also comment on the applicability of our protocol to settings with dishonest majority ($t \geq n/2$).

3.5.1 Deriving more global secrets. While the protocol from Fig. 4 offers a large improvement over naive Pedersen, in many contexts it can be amortized further using known techniques. In the full version we mention two of these techniques based on packed secret sharing [17] and super-invertible matrices [24].

3.5.2 Beyond honest majority. It is easy to see that the protocol above works right out of the box in settings where there is a mix of semi-honest and malicious parties. If there are s semi-honest and t malicious parties, the protocol ensures both privacy and correctness as long as $n > 2t + s$. Note that this is a weaker condition than $n > 2(t + s)$, so in some settings we can get security even when $n - t - s < t + s$. In other words, even if there are more dishonest than honest parties.

4 MULTI-DEALER VSS IN THE YOSO SETTING

We present an MD-VSS protocol in the YOSO model which can serve as a basis for multiple applications of threshold cryptography in this model.

Recall that in the YOSO model, computation is carried via a sequence of *randomly-chosen anonymous committees* where a committee receives secret inputs from previous committees, carries computations based on these inputs and additional public information, and eventually produces an output. Anonymity of committee members precludes the attacker from targeting these entities. However, the anonymity is broken when the party "speaks" (produces output). For this reason, before producing output a committee member must erase all of its secret state (and renew its private-public key pair). Any information in this state that is needed for future actions of the protocol needs to be shared to a committee that will complete these actions when required in the future. The following design is geared to comply with these YOSO rules

4.1 The Basic YOSO MD-VSS Protocol

Consider the VSS functionality (at the core of threshold protocols) and specifically Pedersen's VSS from Fig. 2. The essential verification-response mechanism in the protocol makes it into an interactive protocol where parties may speak more than once. Indeed, a dealer that distributed shares and verification information needs to stay around to respond to accusations. Similarly, shareholders cannot stay silent until they use their shares as they need to first speak in the accusation round hence revealing their identities.

This leads to our re-design of the VSS protocol for the YOSO setting along the lines described in the introduction, namely, creating two intermediate committees between the dealer and shareholders, one for the verification/accusation step, the verifiers, and one for the responses step, the responders. Additionally, the verifying committee transfers information to the target shareholder committee so the latter can compute and verify their final shares.

The introduction of the verification committee basically means that in the YOSO VSS protocol, a dealer will first create a secret sharing $s_1, \ldots, s_n$ of its secret s and then a second-level secret sharing $s_{1\ell}, \ldots, s_{n\ell}$ for each share s_ℓ, $\ell \in [n]$. This second level is shared with the verifiers who check that shares lie on a t-degree polynomial and accuse the dealer if the test fails. Interestingly, if we consider the shares $s_1, \ldots, s_n$ that are reshared by the dealer as n secrets, we can see this as a multi-secret VSS protocol for which we can apply the amortization techniques from Section 3. Furthermore, in the multi-dealer case in the YOSO model, we can reuse the techniques and analysis of the MSMD-VSS protocol from Fig. 4.

We further remark that the sharing of n^2 shares $s_{1\ell}, \ldots, s_{n\ell}$, $\ell \in [n]$, to the responder committee also benefits from the amortization presented in Fig. 4, yet it is not sufficient. The sharing leads to $O(n^3)$ bandwidth per dealer or $O(n^4)$ for n dealers, making the protocol impractical for large committees. We manage to reduce this to a practical size via the "secret sharing made short" technique [25]. Specifically, to send a (long) message from $\mathcal{D}_i$ to $\mathcal{V}_j$, the dealer chooses a short symmetric key ϵ_{ij}, uses it to encrypt the message, and broadcasts the ciphertext. Then it privately sends ϵ_{ij} to $\mathcal{V}_j$, and also shares it among the responders committee members. This way we only pay this extra $n\times$ factor for the short ϵ_{ij}, while broadcasting the long message only once (see Fig. 6).

In our presentation, we split the YOSO MD-VSS protocol in two parts in Fig. 6 and Fig. 8 (Section 4.3). The first includes the sharing mechanisms described above while the second extends the protocol to the case where the secret dealt by the dealer has a known public commitment, and correctness needs to include a proof that the shares obtained by shareholders are consistent with this commitment. Moreover, this part of the protocol also creates public commitments for the shareholder shares. In typical threshold cryptography applications (e.g., threshold signatures), the shareholder shares correspond to their private keys and the commitments act as public keys.

We note that the YOSO MD-VSS protocol shown here has the additional property of being proactive, namely, any information learned about t or less shares becomes obsolete for attacking the fresh sharing. This is a fundamental functionality within the YOSO model where information learned from corrupt parties in one committee should be useless when corrupting parties in any future committee.

In the full version we present a simplification of the protocol that utilizes the existence of public keys for all parties. In essence, the simplification is to collapse the response committee onto the verification committee. This saves a round and eliminates the need for an additional level of sharing. What is required to enable this increased efficiency is for the dealers to encrypt their messages under an encryption scheme where the recipient can expose the encrypted message. This can be done either with an encryption scheme that recovers the randomness, or by adding a ZK proof that the exposed message is in fact the one sent under the encryption.

Parameters: $n \geq 2t + 1$, group $\mathbb{G}$, generators $\vec{G} = (G_0, \dots, G_n)$.
Parties: Dealers $\mathcal{D}_i$, verifiers $\mathcal{V}_j$, responders $\mathcal{R}_k$, shareholders $\mathcal{P}_\ell$.
Inputs: Each $\mathcal{D}_i$, $i \in [n]$, has input $s_i \in \mathbb{Z}_q$.

Dealing. Each dealer $\mathcal{D}_i$, $i \in [n]$, does the following:

(1) Secret share input s_i: Choose a random degree-t polynomial $f_i(\cdot)$ over $\mathbb{Z}_q$ s.t. $f_i(0) = s_i$. Set $s_{ij} = f_i(j)$ $j \in [n]$.
(2) Choose a random $s_{i0} \in \mathbb{Z}_q$.
(3) Secret share each s_{ij}: $\forall \ell \in [[n]]$, choose a random degree-t polynomial $f_{i\ell}(\cdot)$ over $\mathbb{Z}_q$ s.t. $f_{i\ell}(0) = s_{i\ell}$. Set $\sigma_{ij\ell} = f_{i\ell}(j)$ for all $j \in [n]$.
(4) $\forall j \in [n]$, denote $\vec{\sigma}_{ij} = (\sigma_{ij0}, \dots, \sigma_{ijn})$ and then do:
  (a) Compute column-check values $C_{ij} = \vec{\sigma}_{ij} \bullet \vec{G} = \sum_{\ell=0}^{n} \sigma_{ij\ell} G_\ell$.
  (b) Choose a random degree-t polynomial $g_{ij}(\cdot)$, and set $\epsilon_{ij} = g_{ij}(0)$ and $\epsilon_{ijk} = g_{ij}(k)$ for all $k \in [n]$.
  (c) Encrypt ϵ_{ij} with $\mathcal{V}_j$'s public key, ciphertext is E'_{ij}.
  (d) Encrypt $\vec{\sigma}_{ij}$ with symmetric key (derived from) ϵ_{ij}, ctxt is E_{ij}.
(5) $\forall k \in [n]$, denote $\vec{\epsilon}_{ik} = (\epsilon_{i1k}, \dots, \epsilon_{ink})$.
  (a) Compute a vector commitment γ_{ik} to $\vec{\epsilon}_{ik}$, and let ρ_{ik} be the randomness needed to open it.
  (b) Encrypt $(\vec{\epsilon}_{ik}, \rho_{ik})$ with $\mathcal{R}_k$'s public key, ctxt is F_{ik}.
(6) Broadcast $\{C_{ij}, E_{ij}, E'_{ij}\}_{j\in[n]}$ and $\{\gamma_{ik}, F_{ik}\}_{k\in[n]}$.

Verification and accusations. Each verifier $\mathcal{V}_j$, $j \in [n]$, does the following:

(1) For all $i \in [n]$:
  (a) Decrypt E'_{ij} and then E_{ij} to recover the shares $\vec{\sigma}_{ij}$.
  (b) If $C_{ij} \neq \vec{\sigma}_{ij} \bullet \vec{G}$, broadcast an *accusation* against $\mathcal{D}_i$.
(2) Define the set $\mathbb{NA}_j = \{i : \mathcal{D}_i$ was not accused by $\mathcal{V}_j\}$.
(3) $\forall \ell \in [[n]]$, denote $\vec{\xi}_{j\ell} = (\sigma_{ij\ell} : i \in \mathbb{NA}_j)$.
(4) For $\ell \in [n]$, let $h_{j\ell}$ be a commitment to $\vec{\xi}_{j\ell}$ and $\tau_{j\ell}$ is the opening. Denote $\vec{h}_j = (h_{j1}, \dots, h_{jn})$.
(5) Set $\vec{e}_j = (e_{ij} : i \in \mathbb{NA}_j) \in \mathbb{Z}_q^{|\mathbb{NA}_j|}$ as $\vec{e}_j \leftarrow O(j, \vec{h}_j, \mathbb{NA}_j)$, where O is modeled as a random oracle.
(6) $\forall \ell \in [[n]]$ set $\rho_{j\ell} = \sum_{i\in\mathbb{NA}_j} \sigma_{ij\ell} e_{ij} \in \mathbb{Z}_q$ and row-check values $R_{j\ell} = \rho_{j\ell} G_\ell \in \mathbb{G}$.
(7) Compute a NIZK-POK for $\rho_{j\ell} = DL_{G_\ell}(R_{j\ell})$ $\forall \ell \in [[n]]$.
(8) For $\ell \in [n]$, send $(\vec{\xi}_{j\ell}, \tau_{j\ell})$ privately to $\mathcal{P}_\ell$.
(9) For $\ell \in [[n]]$, broadcast $(h_{j\ell}, R_{j\ell}, NIZK_{j\ell})$.

Verifiers that broadcast more than t accusations are ignored. Dealers that received more than t accusations are disqualified.

Response. Responder $\mathcal{R}_k$ does the following for every remaining accusation $\mathcal{V}_j \rightarrow \mathcal{D}_i$:

(1) Decrypt F_{ik} to recover $(\vec{\epsilon}_{ik}, \rho_{ik})$.
(2) Use ρ_{ik} to open the commitment to ϵ_{ijk} inside of $\vec{\epsilon}_{ik}$
(3) Broadcast ϵ_{ijk} and its opening.

Disqualifications. Every shareholder $\mathcal{P}_\ell$ does the following:

(1) For each non-disqualified dealer $\mathcal{D}_i$, check that $C_{i1}, \dots, C_{in}$ lie in the linear subspace of evaluations of degree-t polynomials. Disqualify $\mathcal{D}_i$ if the test fails.
(2) As long as there remain accusations $\mathcal{V}_j \rightarrow \mathcal{D}_i$:
  (a) Check all the commitments γ_{ik} against the opening to ϵ_{ijk} that the $\mathcal{R}_k$'s broadcasted. Disqualify $\mathcal{D}_i$ if less than $n - t$ are valid, or if not all the valid ϵ_{ijk}'s lie on a degree-t polynomial.
  (b) Otherwise use the shares ϵ_{ijk}'s to recover ϵ_{ij}, and use ϵ_{ij} to decrypt E_{ik} and recover $\vec{\sigma}_{ij}$.
  (c) If $C_{ij} \neq \vec{\sigma}_{ij} \bullet \vec{G}$ then disqualify $\mathcal{D}_i$. Otherwise ignore $\mathcal{V}_j$ and all of its accusations and shares.
(3) For each remaining verifier $\mathcal{V}_j$ with received shares $\sigma'_{1j\ell}, \dots, \sigma'_{nj\ell}$ and commitment opening $\tau_{j\ell}$ check the following (and ignore $\mathcal{V}_j$ if any check fails):
  (a) Check $\vec{\xi}_{j\ell}, \tau_{j\ell}$ against the commitment $h_{j\ell}$.
  (b) Verify all the proofs $\{NIZK_{jr}\}_{r\in[[n]]}$.
  (c) Set $\vec{e} \leftarrow O(j, \vec{h}_j, \mathbb{NA}_j)$ and $\rho'_{j\ell} = \sum_{i\in\mathbb{NA}_j} e_{ij}\sigma'_{ij\ell}$. Check that $R_{j\ell} = \rho'_{j\ell} G_\ell$.
  (d) Check that $\sum_{r=0}^{n} R_{jr} = \sum_{i\in\mathbb{NA}_j} e_{ij} C_{ij}$.

Shares. $\mathcal{P}_\ell$ aborts if it has less than $t + 1$ remaining verifiers. Let $J = \{j_0, \dots, j_t\}$ be its first $t + 1$ remaining verifiers and $\mathbb{Q}$ be the qualified dealers, and denote by $\{\sigma'_{ij\ell} : i \in \mathbb{Q}, j \in J\}$ their shares as received by $\mathcal{P}_\ell$. Let $(\lambda_j : j \in J)$ be the Lagrange interpolation coefficients for J.
For every qualified dealer $\mathcal{D}_i$, $i \in \mathbb{Q}$, $\mathcal{P}_\ell$ computes its share of the secret s_i as $s_{i\ell} = \sum_{j\in J} \lambda_j \sigma'_{ij\ell}$.

Figure 6: The basic YOSO MD-VSS protocol

4.2 Analysis of the YOSO MD-VSS protocol

We present our YOSO MD-VSS in Fig. 6 and its analysis in this section. In terms of secrecy, we point out that as long as there are at most t bad parties in each of the committees, the adversary's view in this protocol is essentially the same as in the non-YOSO MSMD-VSS protocol from Fig. 4. There are some commitments and ciphertexts that are sent and never opened, and the ones that are opened correspond to accusations, which means that either sender or receiver must be bad. Finally, there are the shares dealt to honest parties by dishonest dealers, and these are distributed with uniform distribution since there are at most t of them in each committee. A formal simulation proof (using the UC-formulation from [21]) is deferred to later work.

Note the following technical point. Just as in the non-YOSO protocol from Section 3, the honest shareholders agree on the set $\mathbb{Q}$ of qualified dealers. However, in the YOSO protocol we introduce the verifiers and parties may ignore some of the verifiers, and there is no need to have consensus on who are the ignored verifiers.

For soundness, we prove Lemma 4.1 that formalizes the intuitive argument that the only way for $\mathcal{V}_j$ to fool $\mathcal{P}_\ell$ is to have committed (via the $h_{j\ell}$'s) to different shares that happen by chance to agree with the real shares on a random linear combination. Namely, $\mathcal{V}_j$ must commit to values $(\sigma'_{1j\ell}, \dots, \sigma'_{nj\ell})$ that satisfy $\sum_{i=1}^{n} e_{ij}(\sigma'_{ij\ell} - \sigma_{ij\ell}) = 0$, which is enforced by the checks 3c and 3d performed by $\mathcal{P}_\ell$. If the shares are not all the same, then the probability of choosing e_{ij}'s that satisfy this equality is $1/q$.

In the proof below, we ignore the Fiat-Shamir aspect of the protocol and analyze it as if it was an interactive protocol. That is, we consider instead a protocol in which $\mathcal{V}_j$ first sends to each $\mathcal{P}_\ell$ the shares $\sigma'_{ij\ell}$, then the coefficients e_{ij} are chosen at random, and then $\mathcal{V}_j$ uses them to compute the $\rho_{j\ell}$'s and $R_{j\ell}$'s. (In the actual protocol, the e_{ij}'s are computed instead by applying a random oracle to all the $h_{j\ell} = Commit(\vec{\xi}_{j\ell})$, where each $\mathcal{P}_\ell$ verifies the commitment to its own $h_{j\ell}$ and takes the other h_{jr}'s from the broadcast channel "on faith".)

For every honest shareholder $\mathcal{P}_\ell$, and every (not necessarily honest) dealer and verifier $\mathcal{D}_i, \mathcal{V}_j$, let $\sigma'_{ij\ell}$ be the value (alleged share) received by $\mathcal{P}_\ell$ from $\mathcal{D}_i$ via $\mathcal{V}_j$. This is either the value sent from $\mathcal{V}_j$ if there was no accusation, or the value that was reconstructed by the $\mathcal{R}_k$'s if $\mathcal{V}_j$ accused $\mathcal{D}_i$, or 0 is there was an accusation but no value was reconstructed.

For each $i \in \mathbb{Q}$ we also let $\sigma_{ij\ell}$ denote "the correct share" that $\mathcal{D}_i$ was supposed to send, which is defined as follows:

- If $\mathcal{D}_i$ is honest then this is just the value of $f_{ij}(\ell)$ that was sent to $\mathcal{V}_j$ (and shared among the $\mathcal{R}_k$'s).
- If $\mathcal{D}_i$ is dishonest and $\mathcal{V}_j$ is honest, then $\sigma_{ij\ell}$ is either the value that $\mathcal{V}_j$ received (if $\mathcal{V}_j$ did not accuse $\mathcal{D}_i$), or the value that was reconstructed from the $\mathcal{R}_k$'s (if $\mathcal{V}_j$ accused $\mathcal{D}_i$). [4]
- If neither $\mathcal{D}_i$ nor $\mathcal{V}_j$ are honest, then $\sigma_{ij\ell}$ is defined as the interpolation of the "real shares" $\sigma_{ij'\ell}$ for all the honest verifiers $\mathcal{V}_{j'}$. [5]

We note that the values $\sigma_{ij\ell}, \sigma'_{ij\ell}$ are all well defined by the time that $\mathcal{P}_\ell$ runs the disqualification procedure, and can be recovered from the view of all the honest parties. Note also that by definition, $\sigma'_{ij\ell} = \sigma_{ij\ell}$ if $\mathcal{V}_j$ accused $\mathcal{D}_i$.

LEMMA 4.1. *Fix an honest shareholder $\mathcal{P}_\ell$ and a (not necessarily honest) verifier $\mathcal{V}_j$. Assume that the coefficients $(e_{ij})_{i \in \mathbb{NA}_j}$ are chosen at random after all the $\sigma'_{ij\ell}$. Let $\mathbb{Q} \subseteq [n]$ denote the set of indexes of qualified dealers $\mathcal{D}_i$ at the end of the protocol.*

Then, the probability that there exists $i \in \mathbb{Q}$ such that $\sigma'_{ij\ell} \neq \sigma_{ij\ell}$, and yet $\mathcal{V}_j$ is not ignored by $\mathcal{P}_\ell$, is negligible.

The proof is presented in the full version.

4.3 Proving consistent sharing and correct public keys/commitments

We now describe how to extend the basic MSMD-VSS from Fig. 6 to get a proactive re-sharing protocol with public verification keys. Namely, each dealer $\mathcal{D}_i$ has an input share $s_i = f(i)$ for the global secret $s = f(0)$, and it shares its s_i among the shareholders $\mathcal{P}_\ell$. Each shareholder $\mathcal{P}_\ell$ then uses the shares-of-shares $s_{i\ell}$ from all the dealers to reconstruct a new share s'_ℓ of the same global secret. To that end, we use the MSMD-VSS protocol for distributing these $s_{i\ell}$'s, and in addition each dealer $\mathcal{D}_i$ prove that these $s_{i\ell}$'s are "the right ones". We also add a procedure for computing public verification values for the shares.

In more detail, at the beginning of the protocol each dealer $\mathcal{D}_i$ has a single secret input s_i, and we assume that the s_i's lie on a degree-t polynomial f, representing the global secret $s = f(0)$. In addition all the values $S_i = s_i G$ are publicly known. Our goal is to maintain that invariant, so that the same will hold also at the end of the protocol, for new shares s'_ℓ that are held by the shareholders (but the same global secret).

Before we begin, we note that *if the dealers share the "right secrets"* (i.e. valid shares of a global secret), then the shareholders in the protocol from Fig. 6 can indeed compute new shares of the same global secret. To see that, let the original secrets lie on a degree-t polynomial, $s_i = f(i)$, and let $s = f(0)$. Assume further than each dealer $\mathcal{D}_i$ really chooses a degree-t polynomial f_i such that $f_i(0) = s_i = f(i)$, that more than t dealers remain qualified at the conclusion of the protocol, and that each honest shareholder $\mathcal{P}_\ell$ holds the shares $\sigma_{i\ell} = f_i(\ell)$ for every qualified dealer $i \in \mathbb{Q}$.

Let $I = \{i_0, \ldots, i_t\} \subseteq \mathbb{Q}$ be the first $t+1$ qualified dealers. Let $\mu_{i_0}, \ldots, \mu_{i_t}$ be the Lagrange interpolation coefficients for the set I. Namely $p(0) = \sum_{i \in I} \mu_i p(i)$ for every degree-t polynomial p. Consider then the degree-t polynomial $f' = \sum_{i \in I} \mu_i f_i$. On the one hand, each shareholder $\mathcal{P}_\ell$ knows $f_i(\ell)$ for all $i \in I$, so it can compute its share $s'_\ell = f'(\ell) = \sum_{i \in I} \mu_i f_i(\ell)$. On the other hand, we have

$$f'(0) = \sum_{i \in I} \mu_i f_i(0) = \sum_{i \in I} \mu_i f(i) = f(0).$$

Hence the scalars s'_ℓ are indeed shares of the same secret as the original s_i's. We now turn to the tasks at hand, having the dealers prove that they shared the right secrets, and computing the public verification elements.

4.3.1 A first attempt. A natural approach for proving correct resharing goes as follows: Each dealer $\mathcal{D}_i$, after choosing the polynomial f_i and computing a value $\sigma_{i\ell} = f_i(\ell)$ that it wants to communicate to $\mathcal{P}_\ell$'s (via the $\mathcal{V}_j$'s), would broadcast all the corresponding values $Z_{i\ell} = \sigma_{i\ell} G$ (along with NIZK proofs of correctness). Once the set of qualified dealers $\mathbb{Q}$ is established (and hence the Lagrange interpolation coefficients μ_i are known), this will let everyone compute the public value $S'_\ell = \sum_i \mu_i Z_{i\ell}$. The same derivation as above implies that this is the correct public value. Namely $S'_\ell = s'_\ell G$ for the share s'_ℓ that $\mathcal{P}_\ell$ recovered.

Unfortunately, this approach suffers from the randomness-biasing drawback that was pointed out by Gennaro et al. [19]: After seeing the public values $Z_{i\ell}$, the adversary can influence the set of qualified dealers (by making bad verifiers accuse/not accuse some dealers). It can therefore bias the public values S'_ℓ (e.g., ensuring that S'_i begins with a '0'), by trying different combinations for $\mathbb{Q}$ until it finds one that yields the desired result.

4.3.2 Preventing biasing attacks. To overcome this drawback, we need to hide from the adversary the eventual S'_ℓ elements until after the set $\mathbb{Q}$ is determined. The approach that we take here is to *encrypt the values $Z_{i\ell}$ under $\mathcal{P}_\ell$'s public key*, while proving that the encrypted value is what it should be. This will enable $\mathcal{P}_\ell$ to later decrypt, broadcast, and prove that decryption was done correctly.

To improve efficiency, we use Elgamal encryption with keys that "play nice" with the protocol itself. Specifically, each shareholder $\mathcal{P}_\ell$ has an Elgamal public key $(G_\ell, H_\ell = k_\ell G_\ell)$, where the G_ℓ's are the same generators that are used for the protocol, and k_ℓ is the secret key of $\mathcal{P}_\ell$.

[4] Since $i \in \mathbb{Q}$ then we know that some value was reconstructed.
[5] Since $i \in \mathbb{Q}$ then we know that the $\sigma_{ij'\ell}$'s must lie on a degree-t polynomial.

We use $Z_{i\ell} = \sigma_{i\ell}G_\ell$ (rather than $\sigma_{i\ell}G$) as the public key, since it is easier to prove.[6] The dealer $\mathcal{D}_i$ broadcasts an Elgamal encryption of $Z_{i\ell}$, namely the pair $(r_{i\ell}G_\ell, r_{i\ell}H_\ell + Z_{i\ell})$ for a random $r_{i\ell} \in \mathbb{Z}_q$. As we show below, this form allows proving that the encrypted value $Z_{i\ell}$ is "the right one".

After setting $s_{i\ell} = f_i(\ell)$ for a random degree-t polynomial f_i with $f_i(0) = s_i$, sharing will be done exactly as before, letting $\mathcal{P}_\ell$ learn the value $s_{i\ell}$. At the same time, $\mathcal{P}_\ell$ can also decrypt the ciphertext to obtain $Z_{i\ell}$, check that it indeed matches its share, $Z_{i\ell} = s_{i\ell}G_\ell$, and then broadcast Z_ℓ together with a NIZK proof of correct decryption.

Consider the column vector consisting of the second element from each Elgamal ciphertext, namely $(r_{i1}H_1 + \sigma_{i1}G_1, \ldots, r_{in}H_n + \sigma_{in}G_n)^T$. This can be thought of as column-0 of the matrix $\mathbb{A}_i$ (corresponding to the values $\sigma_{i\ell 0} = f_{i\ell}(0), \ell = 1, \ldots n$), except it is shifted by $r_{i\ell}H_\ell$. (The dealer will also publish an encryption for row-0 of this matrix, i.e. $(r_{i0}G_0, r_{i0}H_0 + f_{i0}(0)G_0)$ for some random generator H_0.) Everyone can then compute a column check value for this shifted column, namely add the 2nd elements from all these ciphertexts to get

$$C'_{i0} = \sum_{\ell=0}^{n}(r_{i\ell}H_\ell + f_{i\ell}(0)G_\ell).$$

Recall that all the other C_{ij}'s ($j \in [n]$) must lie on a degree-t polynomial, and that polynomial will be consistent with the $f_{i\ell}$'s if the dealer is honest. Everyone can therefore also use interpolation to compute the (alleged) non-shifted value

$$C_{i0} = \sum_{\ell=0}^{n} f_{i\ell}(0)G_\ell,$$

and then compute the difference $\Delta_i = C'_{i0} - C_{i0}$. The dealer $\mathcal{D}_i$ will broadcast a NIZK proof of knowledge of the representation $(r_{i0}, \ldots, r_{in})$ s.t. $\sum_{\ell=1}^{n} r_{i\ell}H_\ell = \Delta_i$, and moreover the same $r_{i\ell}$'s were used in the first coordinates of the ElGamal ciphertexts $r_{i\ell}G_\ell$. (That is exactly the second protocol from Fig. 1 in Section 2.) This proves that the Elgamal ciphertexts indeed encrypt the secrets $s_{i\ell} = f_{i\ell}(0)$ that were shared and will be reconstructed by the $\mathcal{P}_\ell$'s.

For a share-refresh protocol, it is left to show that these $s_{i\ell}$'s are indeed a valid sharing of s_i (which is the share of the global secret held by $\mathcal{D}_i$). For that, recall that $\mathcal{D}_i$ also publishes $S_i = s_iG_i$ together with proofs of correct decryption from the previous round (where it was a shareholder). $\mathcal{D}_i$ will also provide a NIZK proof of knowledge for $s_i = DL_{G_i}(S_i)$ as well as a representation $(s_{i0}, s_{i1}, \ldots, s_{in})$ of C_{i0} in the bases $G_0, G_1, \ldots, G_n$, such that $(s_i, s_{i1}, \ldots, s_{in})$ lie on a degree-t polynomial. A simple Σ-protocol for this is described in Fig. 7, which can be made non-interactive via the Fiat-Shamir transformation.

4.3.3 The protocol. The invariant that we maintain is that at the beginning of each step, each honest dealer $\mathcal{D}_i$ knows s_i, and everyone knows the corresponding public value $Z_i = s_iG_i$. Moreover, the s_i's of the honest dealers lie on a degree-t polynomial. The modifications to the protocol from Fig. 6 are described in Fig. 8.

[6]If the application needs to use $\sigma_{i\ell}G$, then the shareholder $\mathcal{P}_\ell$ can later publish that value together with a NIZK proofs of DL equality. We note that group elements G_ℓ can be derived deterministically from parties' identities via a random oracle into the group $\mathbb{G}$.

Parameters: generators $G_1, \ldots, G_n$, index $i \in [n]$.
Prover input: degree-t polynomial $f(\cdot)$ and scalar $s_0 \in \mathbb{Z}_q$.
Public input: elements $S, T \in \mathbb{G}$.
Prover claims that $S = f(0)G_i$ and $T = s_0G_0 + \sum_{\ell=1}^{n} f(\ell)G_\ell$.

(1) Prover chooses at random a degree-t polynomial $g(\cdot)$ and $r_0 \in \mathbb{Z}_q$. Sends to verifier $R = g(0)G_i$, $R_0 = r_0G_0$, and $R_\ell = g(\ell)G_\ell$ for $\ell \in [n]$.
(2) Verifier sends to prover a random challenge $c \in \mathbb{Z}_q$.
(3) Prover send $h(\cdot) = g(\cdot) + c \cdot f(\cdot)$ and $z_0 = r_0 + cs_0$.
(4) Verifier checks that h has degree t, that $R + cS = h(0)G_i$, and that $\sum_{\ell=0}^{n} R_\ell + cT = z_0G_0 + \sum_{\ell=1}^{n} h(\ell)G_\ell$.

Figure 7: ZKPOK for proving re-sharing of a secret.

Inputs. Each dealer $\mathcal{D}_i$ has an input share s_i, and everyone knows the public value $S_i = s_iG_i$ (which was proven correct in the previous round).
Each shareholder $\mathcal{P}_\ell$ has a secret key k_ℓ, and everyone knows the corresponding public key $H_\ell = k_\ell G_\ell$.
For $\ell = 0$ there is also a public random "public key" H_0 for which no one knows the secret key.

Dealing. $\mathcal{D}_i$ chooses a random degree-t polynomial f_i s.t. $f_i(0) = s_i$, and set $s_{i\ell} = f_i(\ell)$ for all $\ell \in [n]$. Then it executes the dealing phase and adds the following steps:

(6) For all $\ell \in [[n]]$, choose a random $r_{i\ell} \in \mathbb{Z}_q$ and broadcast $(X_{i\ell} = r_{i\ell}G_\ell, Y_{i\ell} = r_{i\ell}H_\ell + s_{i\ell}G_\ell)$.
(7) Set $C_{i0} = \sum_{\ell=0}^{m} s_{i\ell}G_\ell$, $C'_{i0} = \sum_{\ell=0}^{m} Y_{i\ell}$, and $\Delta = C' - C$.
(8) Compute a NIZK-POK of values $r_{i\ell}$ s.t. $\Delta = \sum_\ell r_{i\ell}H_\ell$ and $X_{i\ell} = r_{i\ell}G_i$ for all $\ell \in [[n]]$ (cf. Fig. 1).
(9) Compute a NIZK-POK of a degree-t polynomial $f(\cdot)$ and a scalar s_{i0}, s.t. $S_i = f(0)G_i$ and $C = s_{i0}G_0 + \sum_{\ell=1}^{n} f(\ell)G_\ell$ (cf. Fig. 7).
(10) Broadcast C_{i0} and the NIZK proofs.

Verification and Accusations. After linearity test, everyone interpolates C_{i0} from $C_{i1}, \ldots, C_{i,t+1}$ and computes $C'_{i0} = \sum_{\ell=0}^{m} Y_{i\ell}$ and $\Delta = C_{i0} - C'_{i0}$. Next everyone checks the NIZK proofs, disqualifying $\mathcal{D}_i$ if they do not verify.

Public values. With $\mathbb{Q}$ the set of qualified dealers, if $|\mathbb{Q}| \le t$ then all shareholders abort. Otherwise let $I \subseteq \mathbb{Q}$ be the first $t+1$ dealers in $\mathbb{Q}$. Let $\{\tau_i\}_{i \in I}$ be the Lagrange interpolation coefficients for the set I.
After computing the shares, each shareholder P_ℓ computes the aggregate ciphertext $(X_\ell = \sum_{i \in I} \tau_i X_{i\ell}, Y_\ell = \sum_{i \in I} \tau_i Y_{i\ell})$. $\mathcal{P}_\ell$ uses its secret key k_ℓ to decrypt, getting $S'_\ell = Y_\ell - k_\ell X_\ell$.
$\mathcal{P}_\ell$ also computes its share of the global secret as $s'_\ell = \sum_{i \in I} \tau_i s_{i\ell}$. If $S'_\ell = s'_\ell G_\ell$ then $\mathcal{P}_\ell$ broadcasts S'_ℓ along with a NIZK proof of correct decryption. Otherwise it aborts.

Figure 8: Additions to the MSMD-VSS from Fig. 6 to obtain a share-refresh protocol.

5 IMPLEMENTATION

As a proof of concept, we implemented (an older version of) the protocol in Section 4.3. That version is about twice as expensive as the protocol in Section 4.3, and it uses an information-theoretic protection of the $Z_{i\ell}$ values rather than ElGamal encryption. Hence the performence numbers below should be cut roughly in half if the new protocol is implemented.

The implementation is available off of https://github.com/shaih/go-yosovss. We used Go (version 1.17.6) for most of the code, and C/C++ for the elliptic curve and field operations. We use a modified version of libsodium [16] (version 1.0.18) for the curve operations and most of the field operations. Modifications include use of non-compressed representation of elliptic curve points (to avoid overhead from decompression), support for multi-scalar-point multiplication (both a constant-time version for operations with secret scalars, and a variable-time version for operations with non-secret scalars), fixed-based scalar-point multiplication (and variant with two bases for Pedersen commitments), matrix multiplications over the scalar field, scalar polynomial evaluation, and other small additional functions. The code is pure portable C without any assembly optimizations. We also use NTL [31] for the generation of the parity-check matrix for the Shamir secret sharing (to do the linear tests). This operation needs to only be performed one time in the whole duration of the system (for a given number of parties n and threshold t).

The communication layer (i.e., the broadcast channels) is simulated using Go channels. (Channels are assumed to be authenticated.) Real networking communication layer can be built and used as a drop-in replacement of these simulated channels. Serialization is fully implemented and uses a canonical version of msgpack [18].

Benchmarking was done on a cloud VM with 50 cores and 128GB of RAM (CPU: AMD EPYC™ 7601, 2.2GHz). The OS was Ubuntu 20.04. For efficiency reasons, only the steps of the first party in each committee are fully executed, the other parties are partially simulated (to provide the required inputs for the first party of the other committees).

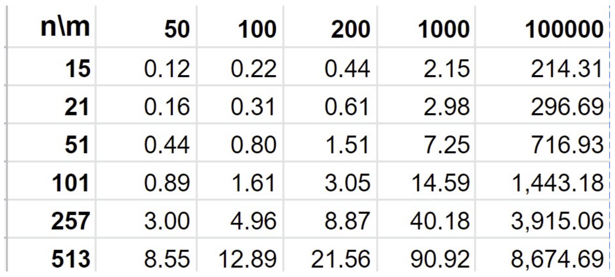
We used a VM with 50 cores and 128GB of RAM just to be able to simulate all the other parties. A single party workload use a very small amount of RAM (less than a few GB) and *a single thread*. The workload of a single party is embarrassingly parallel and the use of multiple threads should give almost linear scaling.

Computational and communication complexity are reported in Table 1. Since all verifiers are honest, there are no accusations and the work of the responders is negligible, hence not reported. The "disqualifications & shares" step include verifying the NIZK made by the dealers and computing the new commitments $S'_i = s'_i G + r'_i H$. From the performance numbers in the table, we can see that all the steps appear to have a quadratic computation and communication complexity (except for the computation complexity of dealing): every time t doubles, the computational and communication complexity is multiplied by 4. Dealing becomes of cubic complexity for high t and n because the scalar operations (to actually generate the shares) become dominant (compared to the scalar-point multiplication that are quadratic and dominant for small t and n).

Micro-benchmarking for the Non-YOSO MultiVSS. We provide in Figs. 9 and 10 some performance results for the non-YOSO MSMD-VSS protocol from Fig. 4. These measurements were obtained by running micro-benchmarking in the above YOSO implementation. As we can see, our protocol provides significant speedup as compared to a basic implementation of the classical Pedersen VSS protocol.

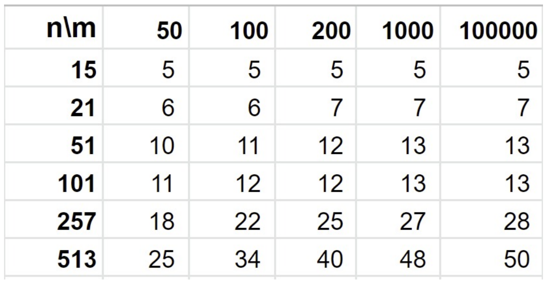
Table 1: Performance of the proactive YOSO resharing protocol in Section 4.3 (computation time and size of the broadcasted message of the first party in each committee)

		Dealing		Accusations		Disqualifications & Shares
t	n	Time	Size	Time	Size	Time
64	129	1.5 s	3.5 MB	1.0s	1.2 MB	3.2 s
128	257	7.2 s	14 MB	4.1s	4.6 MB	12.8 s
256	513	42 s	54 MB	17s	18 MB	52 s



n\m	50	100	200	1000	100000
15	0.12	0.22	0.44	2.15	214.31
21	0.16	0.31	0.61	2.98	296.69
51	0.44	0.80	1.51	7.25	716.93
101	0.89	1.61	3.05	14.59	1,443.18
257	3.00	4.96	8.87	40.18	3,915.06
513	8.55	12.89	21.56	90.92	8,674.69

Figure 9: Single-threaded computation per party (seconds) for the MSMD-VSS protocol, with n dealers and m secrets per dealer.



n\m	50	100	200	1000	100000
15	5	5	5	5	5
21	6	6	7	7	7
51	10	11	12	13	13
101	11	12	12	13	13
257	18	22	25	27	28
513	25	34	40	48	50

Figure 10: The speedup factor of the MSMD-VSS protocol against naive Pedersen, with n dealers and m secrets per dealer.

REFERENCES

[1] Joshua Baron, Karim El Defrawy, Joshua Lampkins, and Rafail Ostrovsky. 2014. How to withstand mobile virus attacks, revisited. In *ACM Symposium on Principles of Distributed Computing, PODC '14, Paris, France, July 15-18, 2014*, Magnús M. Halldórsson and Shlomi Dolev (Eds.). ACM, 293–302. https://doi.org/10.1145/2611462.2611474

[2] Joshua Baron, Karim El Defrawy, Joshua Lampkins, and Rafail Ostrovsky. 2015. Communication-Optimal Proactive Secret Sharing for Dynamic Groups. In *Applied Cryptography and Network Security - 13th International Conference, ACNS 2015, New York, NY, USA, June 2-5, 2015, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 9092)*, Tal Malkin, Vladimir Kolesnikov, Allison Bishop Lewko, and Michalis Polychronakis (Eds.). Springer, 23–41. https://doi.org/10.1007/978-3-319-28166-7_2

[3] Mihir Bellare, Juan A. Garay, and Tal Rabin. 1996. Distributed Pseudo-Random Bit Generators - A New Way to Speed-Up Shared Coin Tossing. In *Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing, Philadelphia, Pennsylvania, USA, May 23-26, 1996*, James E. Burns and Yoram Moses (Eds.). ACM, 191–200. https://doi.org/10.1145/248052.248090

[4] Mihir Bellare, Juan A. Garay, and Tal Rabin. 1998. Fast Batch Verification for Modular Exponentiation and Digital Signatures. In *EUROCRYPT'98 (LNCS, Vol. 1403)*, Kaisa Nyberg (Ed.). Springer, Heidelberg, 236–250. https://doi.org/10.1007/BFb0054130

[5] Fabrice Benhamouda, Craig Gentry, Sergey Gorbunov, Shai Halevi, Hugo Krawczyk, Chengyu Lin, Tal Rabin, and Leonid Reyzin. 2020. Can a Public Blockchain Keep a Secret?. In *Theory of Cryptography - 18th International Conference, TCC 2020, Durham, NC, USA, November 16-19, 2020, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 12550)*, Rafael Pass and Krzysztof Pietrzak (Eds.). Springer, 260–290. https://doi.org/10.1007/978-3-030-64375-1_10

[6] Erica Blum, Jonathan Katz, Chen-Da Liu-Zhang, and Julian Loss. 2020. Asynchronous Byzantine Agreement with Subquadratic Communication. In *TCC 2020, Part I (LNCS, Vol. 12550)*, Rafael Pass and Krzysztof Pietrzak (Eds.). Springer, Heidelberg, 353–380. https://doi.org/10.1007/978-3-030-64375-1_13

[7] Ignacio Cascudo and Bernardo David. 2017. SCRAPE: Scalable Randomness Attested by Public Entities. In *ACNS 17 (LNCS, Vol. 10355)*, Dieter Gollmann, Atsuko Miyaji, and Hiroaki Kikuchi (Eds.). Springer, Heidelberg, 537–556. https://doi.org/10.1007/978-3-319-61204-1_27

[8] Dario Catalano and Dario Fiore. 2013. Vector Commitments and Their Applications. In *PKC 2013 (LNCS, Vol. 7778)*, Kaoru Kurosawa and Goichiro Hanaoka (Eds.). Springer, Heidelberg, 55–72. https://doi.org/10.1007/978-3-642-36362-7_5

[9] Jing Chen and Silvio Micali. 2019. Algorand: A secure and efficient distributed ledger. *Theor. Comput. Sci.* 777 (2019), 155–183. https://doi.org/10.1016/j.tcs.2019.02.001

[10] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. 1985. Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults (Extended Abstract). In *26th FOCS*. IEEE Computer Society Press, 383–395. https://doi.org/10.1109/SFCS.1985.64

[11] Arka Rai Choudhuri, Aarushi Goel, Matthew Green, Abhishek Jain, and Gabriel Kaptchuk. 2021. Fluid MPC: Secure Multiparty Computation with Dynamic Participants. In *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 12826)*, Tal Malkin and Chris Peikert (Eds.). Springer, 94–123. https://doi.org/10.1007/978-3-030-84245-1_4

[12] Ronald Cramer. 1996. *Modular Design of Secure yet Practical Cryptographic Protocols*. Ph. D. Dissertation. CWI and University of Amsterdam.

[13] Ivan Damgård. 2010. On Σ Protocols. https://cs.au.dk/%7Eivan/Sigma.pdf.

[14] Ivan Damgård and Yuval Ishai. 2006. Scalable Secure Multiparty Computation. In *CRYPTO 2006 (LNCS, Vol. 4117)*, Cynthia Dwork (Ed.). Springer, Heidelberg, 501–520. https://doi.org/10.1007/11818175_30

[15] Ivan Damgård, Yuval Ishai, Mikkel Krøigaard, Jesper Buus Nielsen, and Adam Smith. 2008. Scalable Multiparty Computation with Nearly Optimal Work and Resilience. In *CRYPTO 2008 (LNCS, Vol. 5157)*, David Wagner (Ed.). Springer, Heidelberg, 241–261. https://doi.org/10.1007/978-3-540-85174-5_14

[16] Frank Denis. 2022. The Sodium cryptography library. https://download.libsodium.org/doc/

[17] Matthew K. Franklin and Moti Yung. 1992. Communication Complexity of Secure Computation (Extended Abstract). In *24th ACM STOC*. ACM Press, 699–710. https://doi.org/10.1145/129712.129780

[18] Sadayuki Furuhashi. 2013. MessagePack Serialization Format. https://msgpack.org/

[19] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. 2007. Secure Distributed Key Generation for Discrete-Log Based Cryptosystems. *Journal of Cryptology* 20, 1 (Jan. 2007), 51–83. https://doi.org/10.1007/s00145-006-0347-3

[20] Rosario Gennaro, Michael O. Rabin, and Tal Rabin. 1998. Simplified VSS and Fast-Track Multiparty Computations with Applications to Threshold Cryptography. In *17th ACM PODC*, Brian A. Coan and Yehuda Afek (Eds.). ACM, 101–111. https://doi.org/10.1145/277697.277716

[21] Craig Gentry, Shai Halevi, Hugo Krawczyk, Bernardo Magri, Jesper Buus Nielsen, Tal Rabin, and Sophia Yakoubov. 2021. YOSO: You Only Speak Once / Secure MPC with Stateless Ephemeral Roles. In *CRYPTO 2021, to appear.* https://ia.cr/2021/210.

[22] Craig Gentry, Shai Halevi, and Vadim Lyubashevsky. 2021. Practical Non-interactive Publicly Verifiable Secret Sharing with Thousands of Parties. *IACR Cryptol. ePrint Arch.* (2021), 1397. https://eprint.iacr.org/2021/1397

[23] Amir Herzberg, Stanislaw Jarecki, Hugo Krawczyk, and Moti Yung. 1995. Proactive Secret Sharing Or: How to Cope With Perpetual Leakage. In *CRYPTO'95 (LNCS, Vol. 963)*, Don Coppersmith (Ed.). Springer, Heidelberg, 339–352. https://doi.org/10.1007/3-540-44750-4_27

[24] Martin Hirt and Jesper Buus Nielsen. 2006. Robust Multiparty Computation with Linear Communication Complexity. In *CRYPTO 2006 (LNCS, Vol. 4117)*, Cynthia Dwork (Ed.). Springer, Heidelberg, 463–482. https://doi.org/10.1007/11818175_28

[25] Hugo Krawczyk. 1994. Secret Sharing Made Short. In *CRYPTO'93 (LNCS, Vol. 773)*, Douglas R. Stinson (Ed.). Springer, Heidelberg, 136–146. https://doi.org/10.1007/3-540-48329-2_12

[26] Stephan Krenn, Thomas Lorünser, and Christoph Striecks. 2017. Batch-verifiable Secret Sharing with Unconditional Privacy. In *Proceedings of the 3rd International Conference on Information Systems Security and Privacy, ICISSP 2017, Porto, Portugal, February 19-21, 2017*, Paolo Mori, Steven Furnell, and Olivier Camp (Eds.). SciTePress, 303–311. https://doi.org/10.5220/0006133003030311

[27] Yehuda Lindell and Ariel Nof. 2018. Fast Secure Multiparty ECDSA with Practical Distributed Key Generation and Applications to Cryptocurrency Custody. In *ACM CCS 2018*, David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang (Eds.). ACM Press, 1837–1854. https://doi.org/10.1145/3243734.3243788

[28] Rafail Ostrovsky and Moti Yung. 1991. How to Withstand Mobile Virus Attacks (Extended Abstract). In *10th ACM PODC*, Luigi Logrippo (Ed.). ACM, 51–59. https://doi.org/10.1145/112600.112605

[29] Torben P. Pedersen. 1991. A Threshold Cryptosystem without a Trusted Party (Extended Abstract) (Rump Session). In *EUROCRYPT'91 (LNCS, Vol. 547)*, Donald W. Davies (Ed.). Springer, Heidelberg, 522–526. https://doi.org/10.1007/3-540-46416-6_47

[30] Torben P. Pedersen. 1992. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *CRYPTO'91 (LNCS, Vol. 576)*, Joan Feigenbaum (Ed.). Springer, Heidelberg, 129–140. https://doi.org/10.1007/3-540-46766-1_9

[31] Victor Shoup. 2022. NTL: A Library for doing Number Theory. https://libntl.org/