Asynchronous Dynamic Proactive Secret Sharing under Honest Majority: Refreshing Without a Consistent View on Shares

Matthieu Rambaud and Antoine Urban
Télécom Paris and Institut Polytechnique de Paris

ABSTRACT

We present the first proactive secret sharing under honest majority which is purely over pairwise asynchronous channels. Moreover:

- it has robust reconstruction. In addition, provided a single broadcast in the sharing phase, then it commits the dealer to a value;
- it operates under a bare PKI and enables dynamic membership;
- the standard version carries over the model known as "receiver-anonymous (Yoso)" in which participants speak only once then erase their memories. Each refresh of a secret takes 2 actual message delays and a total of $O(n^4)$ bits sent by the honest parties;
- it allows an optimization in $O(n^3)$ bits and latency of 5 messages.

1 INTRODUCTION

The goal of threshold cryptography is to process information that should remain secret, and quickly and reliably deliver a result, despite an adversary corrupting any minority of participants. Flagship use-cases are the distributed generation and storage of secret keys, for the purpose of distributed signing of transactions, or of secure storage [BGG+20; Gro21]. The baseline technique is known as *secret sharing*. It enables the owner of a secret to distribute shares of it, to n parties, while ensuring (i) that an adversary controling some threshold number of t-out-of-n parties, will learn no information on the secret, and (ii) robust reconstruction of the secret from any $t+1$ valid shares. To withstand a *mobile* adversary, i.e. that can corrupt possibly all parties within the lifetime of the system, [OY91] introduced the notion of *proactive security*. The lifespan of a protocol is divided into time periods denoted *epochs*, and the adversary is able to corrupt at most t parties per epoch. A *proactive secret sharing* (PSS) scheme *refreshes* the shares held by parties in every new epoch. As a result, parties obtain new shares of the secret that are independent of the old ones, which then can be safely erased. We consider the most general setting, which is known as *dynamic* PSS protocols, i.e., that support changes of participants across epochs. This model considers one separate set of parties per epoch, denoted as a *committee*. Since the model is agnostic of the physical computers on which parties of different committees are running, it captures all settings. It covers the particular case where some party, which would have been corrupt then reinitialized, would re-enter the protocol with an empty state, thus being treated as a new participant. The goal of a dynamic PSS is to:

- enable the secret owner to share its secret to a committee $C^{(1)}$ of parties, which we denote as the committee of the first epoch $e = 1$;
- maintain the correctness and liveness invariants that in each epoch e, the committee $C^{(e)}$ should hold shares of this same secret;
- to this end, there is a protocol, denoted Refresh, between each committee $C^{(e)}$, denoted *exiting*, and $C^{(e+1)}$, denoted *entering*, which should provide $C^{(e+1)}$ with new shares of the same secret, in a way that maintains the privacy invariant that the adversary does not gain any incremental information on the secret, despite having corrupted t parties in every committee so far.

Benefits and Challenges of Asynchronous Protocols. All known PSS under honest majority assume a *synchronous* network, i.e., where the delivery delay of the messages is upper-bounded by a *known* constant. Synchronous PSS protocols either lose privacy, consistency or liveness as soon as the delivery of a single message from a honest party takes more than this constant: see section 1.3. In turn, synchronous PSS protocols instruct to wait for a predetermined delay between each step, which is a worst-case conservative upper bound on the network delay, resulting in artificially long executions. Furthermore, synchronous PSS protocols require broadcast channels, of which the implementation either requires substantial interaction ([KK09] for the state of the art under tight adversary bound and no common coin setup), or, an external public ledger [MZW+19; BGG+20; GKM+22; Gro21]. Asynchronous protocols, by contrast, run at *the actual speed of the network*, which is a property known as *responsiveness*, and still resists to arbitrarily long periods of asynchrony. In an asynchronous network, there is no upper-bound on the latencies of messages. The challenge is thus that it is impossible for a party P_i expecting a message from another party P_j to distinguish between 1) a slow honest party, whose message has not yet arrived due to the network latency and 2) an adversary that did not send anything. Waiting for such a message in an asynchronous protocol could prevent *liveness*, as it can stop P_i from making progress. Thus, whenever we let parties multicast to all the parties in the protocol, in the next step we can at most let an honest party wait for n-t messages. To achieve this, all previous works required some form of Byzantine consensus among the new committee, very roughly, to reach consensus on a subset of new secret shares received obtained from the previous committee. Since asynchronous consensus within a committee is not solvable beyond $t < n/3$ corruptions [DLS88], we need other techniques to break the bound.

This paper's main goal is to answer the following question: Is it possible to construct a proactive and asynchronous secret sharing scheme under honest majority ?

Secondary Challenge: Minimizing the Window of Vulnerability. On the one hand, the dynamic committee model is agnostic of whether two members $P_i^{(e)}$ and $P_i^{(e+1)}$ of adjacent committees, $C^{(e)}$ and $C^{(e+1)}$, would be running on the same machine. On the other hand, it is commonplace to weaken the adversary model by making the reasonable convention that, in such a case, if the one ($P_i^{(e)}$) of the exiting committee is still corrupt and online after $C^{(e+1)}$ has entered the protocol, then, $P_i^{(e+1)}$ counts in the corruption budget of $C^{(e+1)}$. Thus, the longer the time-frame while two adjacent committees must be simultaneously online, the more this reasonable assumption *weakens* the freedom of the adversary to choose

whom to corrupt in each committee. We thus denote this timeframe as *window of vulnerability*. This terminology encompasses the one denoted "future horizon" in [GHK+21], but is orthogonal to the one of [ZSV05]. However, recent optimized PSS protocols [MZW+19; GKM+22; YXD22] take an approach, which consists in generating a fresh sharing of 0 to mask old shares, which incurs a substantial window of vulnerability.

1.1 Results

We answer positively the feasibility question.

Theorem 1. *Consider committees of $n = 2t+1$ parties, in each of which t are maliciously corrupt by a polynomial mobile adversary. Consider a fully asynchronous communication network, with the bare setup of a bulletin board of public keys. Then there exists a protocol, denoted* DP-AVSS*, that implements proactive secret sharing.*

Moreover:

- DP-AVSS has (public) *verifiability*, i.e., that upon termination of the sharing phase, there exists exactly one value, such that only this value can possibly be the one opened. The sharing takes one single terminating broadcast from the secret owner to the first committee, which could furthermore be traded by a simple multicast if we had considered only honest secret owners.
- DP-AVSS has no window of vulnerability: members of the exiting committee need only to send a single batch of messages to the entering committee, then erase their memories;
- DP-AVSS completes a Refresh within the actual network delay of 3 consecutive asynchronous messages deliveries;
- The total bitsize sent by honest parties of a committee is $O(n^4)$, further optimized into $O(n^3)$ (section 5.2) if allowing a window of vulnerability of 2 messages, and a total Refresh delay of 5 messages;
- DP-AVSS straightforwardly adapts to a communication model, known as "You Only Speak Once", where members of committees are known only by their public key and speak only once (§5.3);
- DP-AVSS straightforwardly generalizes to possibly many secret owners and receivers and, by design, enables the opening a linear combination of secrets from possibly several owners.

A by-product of our proof, in section 4.6, is that we show simulatability of publicly verifiable secret sharing (PVSS), thus positively answering the question raized by [SBKN21, p5]. We furthermore achieve it without straight-line extraction, thanks to a gadget denoted "inference of corrupt shares". It thus allows (section 5.4) lighter NIZK proofs, in logarithmic size.

1.2 Our Techniques

Our baseline approach is based on *resharing of secret shares*. It was introduced by [GHY87], then used for proactivity in [AGY95; FGMY97; CKLS02; GG06; DM15; BGG+20; Gro21; GHL22]. Let us outline it very roughly, on the example of [CKLS02]. Initially, each party P_i owns a share s_i of a secret s, under univariate Shamir sharing with threshold t. At the end of an epoch e, parties broadcast new keys and perform the resharing as follows:

- *Non-interactive resharing of each share:* Every party $P_i^{(e)}$ performs asynchronous verifiable secret sharing (AVSS) of its share $s_i^{(e)}$, and *immediately deletes* it.
- Parties perform multi-valued Byzantine consensus to agree on a subset $\mathcal{U}$ of $t+1$ successfully terminated AVSS.
- *Generation of a new share:* Each party $P_j^{(e+1)}$ computes its new share $s_j^{(e+1)}$ from the set of $t+1$ shares received from $\mathcal{U}$, by applying the Lagrange reconstruction formula on them.

However, this approach is *not* applicable in an asynchronous setting under honest majority, as it either requires broadcast [DM15; BGG+20; Gro21], or AVSS and Byzantine consensus [CKLS02] which cannot be achieved for $t \geq n/3$.

To circumvent this difficulty, we propose a new computation model. Parties exchange a structure of data that are n-sized vectors of encrypted shares. We maintain the invariant that all such vectors encrypt shares of the same secret s, furthermore equal to the one initially shared by the secret owner. To guarantee this correctness, we impose them to be signed by a quorum of $t+1$ parties: we denote this structure *verified proactivized sharing* VPS. VPS are created in two steps. First, each member of the exiting committee $C^{(e)}$, for every $\text{VPS}^{(e)}$ which it has, decrypts its share then generates a fresh publicly verifiable (re-)sharing of it, i.e., a vector of encrypted shares of its share. Then it engages into n parallel instances of a sub-protocol, denoted Leader-Based (LB)-refresh, each being coordinated by a party of $C^{(e+1)}$ acting as the leader. The leader aggregates any $t+1$ re-sharings of the same VPS into a new vector of encrypted shares. To this end it applies the same Lagrange reconstruction formula as above, which it evaluates linearly homomorphically on the vectors of shares. Then it submits this new vector of encrypted shares to $C^{(e+1)}$, which verify its correctness then multicast their signatures on it, to form a $\text{VPS}^{(e+1)}$. Existence of at least one (even $t+1$) honest leader maintains the invariant that: all $t+1$ honest parties in $C^{(e+1)}$ ultimately obtain, at least, one $\text{VPS}^{(e+1)}$ in common, even if they do not know which one. Remarkably, there is no guaranteed common view on a unique VPS created in a LB-refresh driven by a malicious leader.

1.3 Further Details on Related Works

The protocol [ZSV05] also runs n instances of a leader-based refresh in parallel. However, to guarantee correctness and uniform termination in each instance, they require quorums of size $2t+1$, in turn limiting their resilience to $n \geq 3t+1$. Moreover, the size of their shares are exponential in n. The protocols [BGG+20; Gro21] and [GHL22, §C] operate by publicly verifiable re-sharings, which are sent over a synchronous broadcast channel. Under our asynchronous setting we do not have such resource, thus the problem of consistency is non-trivial. In the Table 1 we multiply the complexities of [SLL10] by $O(t)$, since in the worst-case of t consecutive corrupt leaders, their refresh could be restarted t times before succeeding. The protocol Exp-CHURP-A [MZW+19] is the one which serves as fallback of their optimistic protocol (not displayed in Table 1). It has same $n\text{BC}(n)$ complexity as [BGG+20] and [GHL22, §C]. In the regime of optimal corruption tolerance $n = 2t+1$, this fallback is activated as soon as one single message is not timely received. Critically, when the broadcast from a honest party P_j is not received in time, then [MZW+19, p. C.1.3] forces honest parties to publicly expose what they sent to P_j, which provides enough

Scheme	Network	Threshold	Dynamic	Comm.	Window of Vulnerability	Setup
Herzberg et al. [HJKY95]	Synch	$t < n/2$	No	$n\|\text{BC}(n)\|$	$3BC$	PKI
Cachin et al. [CKLS02]	Asynch	$t < n/3$	No	$O(n^4)$	0	threshold coin
Zhou et al. [ZSV05]	Asynch	$t < n/3$	Yes	$exp(n)$	$O(\delta)$	secure channels
MPSS [SLL10]	Partial-Synch	$t < n/3$	Yes	$O(tn^4)$	$O(t)$	PKI
Exp-CHURP-A [MZW+19]	Synch	$t < n/2$	Yes	$n\|\text{BC}(n)\|$	5BC	PKI
Goyal et al. [GKM+22]	Synch	$t < n/2$	Yes	$\|\text{BC}(n)\|$	5BC	PKI
Shanrang [YXD22]	Asynch	$t < n/4$	Yes	$O(n^3\log(n))$	MVACS	threshold coin
DP-AVSS(Th 7)	Asynch	$t < n/2$	Yes	$O(n^4)$	0	PKI
Optimization (Section 5.2)	Asynch	$t < n/2$	Yes	$O(n^3)$	2δ	PKI

Table 1: *Window of vulnerability* denotes the worst-case consecutive interactive tasks requiring parties of both the exiting and entering committee to be simultaneously online. Publication delay on the bulletin board of public keys is ignored. 2δ denotes 2 consecutive message deliveries, MVACS denotes multivalued agreement on a common subset. + denotes consecutive tasks, e.g., 5BC denotes 5 consecutive broadcasts. Communication complexity is also measured as the total number of bits sent by honest parties in a committee to realize a task. E.g.: $n|\text{BC}(n)|$ denotes the bitsize of n broadcasts, each with input size $O(n)$ bits.

information to the adversary to reconstruct the stored secret. Exp-CHURP-A also serves as fallback for Shanrang [YXD22], which has asynchronous runs in failure-free executions, and runs in expected $O(\log n)$ rounds for every Refresh.

Likewise in [GKM+22, p8], when a message from some honest sender is not received in time by a honest party, then the sender is forced to publicly expose the content of this message, which provides to the adversary substantial information on the stored secret. The $3BC$ and 5BC of window of vulnerability in [HJKY95] and [GKM+22] account for the possible *accusation-and-response* phase. Their $|\text{BC}(n)|$ complexity is in the amortized regime of sharing more than n secrets

2 MODEL AND DEFINITIONS

The set of non-negative integers is denoted as $\mathbb{N}$, the set of positive ones is denoted as $\mathbb{N}^* = \{1, 2, \dots\}$. We consider integers $t \in \mathbb{N}$ and $n := 2t+1$. For F a finite set, we denote $|F|$ its cardinality, and $f \xleftarrow{\$} F$ the sampling of an element in F uniformly at random. The empty string is denoted as $\perp$. For m an integer, we denote $[m] := \{1, \dots, m\}$. Vectors of size n are denoted in bold: $\boldsymbol{c}$, and their coordinates as c_i, or $\boldsymbol{c}[i]$, for $i \in [n]$.

Finally, we consider secrets in prime finite fields, as follows. Let p denote any prime number larger than $n+1$, then $\mathbb{F}_p$ denotes the finite field $\mathbb{Z}/p\mathbb{Z}$. $\mathbb{F}_p[X]_t$ denotes the $(t+1)$-vector space of polynomials of degree at most t. We denote $\mathbb{F}_p[X]_t^{(0)} := \left\{\sum_{i=1}^{t} r_i X^i,\ r_i \in \mathbb{F}_p\ \forall i \in [t]\right\} \subset \mathbb{F}_p[X]_t$ the t-subspace of polynomials evaluating to 0 at 0, i.e., with 0^{th} coefficient equal to 0. We leave implicit the security parameter.

2.1 Mobile Corruptions

We consider an arbitrarily long sequence of integers $e \in \mathbb{N}^*$ denoted "epoch numbers". For each e, we consider a set of distinct probabilistic polynomial time (PPT) machines, denoted "committee" $C^{(e)} = (P_1^{(e)}, \dots, P_n^{(e)})$, for simplicity of fixed size $n = 2t+1$. Their members are denoted "parties". The $C^{(e)}$ are disjunct, nonwithstanding some $P_i^{(e)} \in C^{(e)}$ and $P_j^{(e+1)} \in C^{(e+1)}$ could possibly run on the same physical computer. For simplicity parties have fixed and public identities, although the protocol straightforwardly adapts to "receiver-anonymous" models (§5.3).

We consider a PPT machine denoted the Environment $\mathcal{E}$. For each $e \in \mathbb{N}^*$ and honest party $P_i^{(e)} \in C^{(e)}$, $\mathcal{E}$ may send a signal "start" to $P_i^{(e)}$ at some point. From this point we say that $P_i^{(e)}$ is "alive". We also consider two PPT machines denoted "secret owner" SO and "secret-receiver" SR, of which none, one or both can be corrupt. If SO is honest, then $\mathcal{E}$ may send ("share", $s \in \mathbb{F}_p$) to it. Anticipating on §2.3, this instructs SO to share some secret s to $C^{(1)}$. For every e, $\mathcal{E}$ may send: "refresh" to parties in $C^{(e)}$. Anticipating §2.3, this instructs them to start interacting with $C^{(e+1)}$. For any $e_o \in \mathbb{N}^*$, $\mathcal{E}$ may send "open" to some honest parties in $C^{(e_o)}$. Anticipating §2.3, this instructs them to open to SR all the shares of the secret which they have. We then designate e_o as the *opening epoch* and $C^{(e_o)}$ as the "opening committee". For simplicity we consider that $\mathcal{E}$ then does not send "refresh " to parties in $C^{(e_o)}$. $\mathcal{E}$ then learns the value output by SR if it is honest (or, if it is corrupt, all the messages sent to it). Protocols may instruct parties to shut-off themselves, which means that they erase all their memory and quit the protocol.

We consider a PPT machine $\mathcal{A}$ denoted "adversary". Without loss of generality [Can01] we actually consider $\mathcal{A}$ to be fully controlled by $\mathcal{E}$, such $\mathcal{A}$ is known as "dummy". For each $e \in \mathbb{N}^*$, $\mathcal{A}$ may maliciously *corrupt* a set $\mathcal{I}^{(e)}$ of up to t parties in $C^{(e)}$, i.e., it has full control over them. We denote the non-corrupt parties as *honest*. For simplicity we consider *static corruptions* with respect to each committee, i.e., $\mathcal{A}$ cannot corrupt a party which received "start", i.e., which is already alive. DP-AVSS (but not the optimization of §5.2) extends to adaptive corruptions with standard techniques [CDN15, §5.6][GKM+22]. In addition, $\mathcal{A}$ may de-corrupt parties at any time, which also has for effect to shut them off.

2.2 Ideal Functionalities and Delayed Output

Following [Can01] we say that an ideal functionality $\mathcal{F}$ sends a *delayed output* v to R if it engages in the following interaction: instead of simply outputting v to R, $\mathcal{F}$ first sends to the adversary a request, which we denote as network, asking for permission to deliver an output to R. When we make the precision *public* delayed output, then this means that the content of the value v is

furthermore leaked to $\mathcal{A}$ in the request. When the adversary replies [which we denote as activate], $\mathcal{F}$ outputs v to R. By ssid we denote the sub-session numbers of functionalities. We omit the session number since we will consider for simplicity one single instance of secret sharing.

Asynchronous Authenticated Message Transmitting $\mathcal{F}_{\text{AUTH}}$ In Thm 7 all entities are connected by pairwise public authenticated channels, i.e., the content of bitstrings sent on the network is leaked to the adversary. In particular, we do not need to specify private channels from the parties to the secret-receiver SR, even if it is honest, because we are able to perfectly simulate opening shares. This functionality is formalized by [Can01] as $\mathcal{F}_{\text{AUTH}}$. It is parametrized by a sender S and receiver R. On input (input, ssid, v) from S: then $\mathcal{F}_{\text{AUTH}}^{S,R}$ provides R with public delayed output (ssid, v).

2.2.1 Bulletin Board of Public Keys. In Thm 7 we will consider the classical model, denoted as "PKI" in [DMR+21], which can be phrased as enabling parties to publish the public key of their choice upon entering the protocol. We formalize it as the functionality denoted *certification authority* $\mathcal{F}_{CA}$ in [Can04], and provide it in the appendix §A. We will in particular use digital signatures, which are shown in [Can04] to be UC implementable from $\mathcal{F}_{CA}$. Subsequent works [CSV16] also denote $\mathcal{F}_{CA}$ as "bulletin board" PKI or "bare" PKI, to underline that it does not perform any check of knowledge of a secret key. By contrast, protocols known as "distributed key generation" implement a much stronger setup, which securely *assigns* correlated private keys to parties. Our protocol actually withstands a slight downgrading of $\mathcal{F}_{CA}$, into an entity that would deliver certificates tying a public key to the identity of its owner, and would possibly deliver certificates for different keys tied to the same corrupt party.

2.2.2 Terminating Reliable Broadcast. The ideal functionality $\mathcal{F}_{\text{BC}}$, on input a value (possibly $\perp$) from either SO or possibly $\mathcal{A}$ if SO is corrupt, delay-outputs this value to all parties.

2.2.3 NIZKs. We will use *non-interactive zero knowledge arguments of knowledge*, which we dub as *NIZKs*. For simplicity, we capture them by the ideal functionality $\mathcal{F}_{\text{NIZK}}$, defined as in [GO07]. $\mathcal{F}_{\text{NIZK}}$ is parametrized by a NP relation R. Upon request of a prover P exhibiting some public input x and knowledge of some secret witness w, it verifies if $(x, w) \in R$ then deletes w from its memory. If the verification passes, then $\mathcal{F}_{\text{NIZK}}$ *delay-outputs* to P a string π. Upon subsequent input the same string π and x from any verifier, $\mathcal{F}_{\text{NIZK}}$ then confirms to the verifier that P knows some witness for x. We denote Π the space of such strings π, their sizes will be discussed in §5.4. We provide a description of $\mathcal{F}_{\text{NIZK}}$ in Fig. 5. Under our honest majority setting, it can be implemented from $\mathcal{F}_{CA}$ using so-called multi-string NIZKs [GO07].

2.2.4 Signature Scheme. We assume that parties have access to a non-interactive digital signature scheme, as defined in [Bol03, §2]. Recall that it consists in the following algorithms. There is a KeyGen() function that allows any party to generate a key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$. Then, on input its secret key and any message m, a party P_i can generate a signature $\sigma_i \leftarrow \mathsf{Sign}(m, \mathsf{sk}_i)$. There is a public verification function, denoted Sign.Verify, that takes as input a public key pk and checks the validity of any signature σ on any m.

In addition, we require a public aggregation function, that takes as input a set of signatures $\{\sigma_i\}_{i\in\mathcal{U}}$, where $|\mathcal{U}| = t+1$, a fixed list of n public keys $(\mathsf{pk}_i)_{i\in[n]}$ and each $\sigma_i \leftarrow \mathsf{Sign}(m, \mathsf{sk}_i)$, and can be used to produce a signature $\sigma \leftarrow \mathsf{Sign.Combine}(m, \{\sigma_i\}_{i\in\mathcal{U}})$ on m. We require a robustness property, as defined in [Sho00; Bol03] which states that if $\sigma_i \leftarrow \mathsf{Sign}(m, \mathsf{sk}_i)$ with $\mathsf{Sign.Verify}(m, \mathsf{pk}_i, \sigma_i)$ returning True for each $i \in \mathcal{U}$, $|\mathcal{U}| = t+1$, and if $\sigma \leftarrow \mathsf{Sign.Combine}(m, \{\sigma_i\}_{i\in\mathcal{U}})$, then $\mathsf{Sign.Verify}(m, \sigma)$ returns True. We require the unforgeability property that, an adversary being allowed to choose t public keys $(\mathsf{pk}_i)_{i\in\mathcal{I}}$ and given access to oracles $\{\mathsf{Sign}(\cdot, \mathsf{sk}_i)\}_{i\in[n]\setminus\mathcal{I}}$, then it has a negligible probability of producing a signature σ for any message m on which it did not query any of the oracles. Notice that the aggregation function could be implemented as the plain concatenation of the set of $t+1$ signatures, see §5.4 for more efficient implementations under the bare bulletin board PKI setup.

2.3 DP-AVSS Protocols

We formalize the functionality which we aim at implementing, as "Dynamic Proactive Asynchronous Verifiable Secret Sharing" $\mathcal{F}_{\text{DP-AVSS}}$.

$\mathcal{F}_{\text{DP-AVSS}}$

Upon receiving ("share", $s \in \mathbb{F}_p$) {from SO or, possibly, from $\mathcal{A}$ if SO corrupt} for the first time: stores s, sends "stored" to honest parties, and initiates $e := 1$.
Upon receiving "started" from all honest parties in $C^{(e)}$, leak "started" to $\mathcal{A}$.
Upon receiving "refresh" from all honest parties in $C^{(e)}$, leak "refresh" to $\mathcal{A}$ and sets $e := e + 1$.
Upon receiving "open" from all honest parties in $C^{(e)}$, if some $s \in \mathbb{F}_p$ is stored, leaks "open" to $\mathcal{A}$ then delay-outputs s to SR.

Figure 1

Following the model [Can01], we say that a protocol UC-implements $\mathcal{F}_{\text{DP-AVSS}}$ if there exists a PPT machine $\mathcal{S}$ denoted *simulator*, also known as "ideal adversary", such that no $\mathcal{E}$ can tell apart whether it is interacting with:

- either (i) the dummy $\mathcal{A}$, which fully controls t out of the $n = 2t+1$ parties in each of the committees (ii) honest parties performing the actual protocol, (iii) and possibly SO and/or SR if they are honest;
- or with (i) $\mathcal{S}$, which interacts, on the one side, with $\mathcal{F}_{\text{DP-AVSS}}$ on behalf of corrupt parties, and on the other side with $\mathcal{E}$ on behalf of $\mathcal{A}$, (ii) honest parties performing the "dummy protocol", i.e., which simply forward to $\mathcal{F}_{\text{DP-AVSS}}$ the signals "start", "refresh" and "open" every time they receive them from $\mathcal{E}$, (iii) and possibly SO and/or SR if they are honest, in which case: SO performs the dummy protocol, consisting in forwarding ("share", $s \in \mathbb{F}_p$) to $\mathcal{F}_{\text{DP-AVSS}}$ upon receiving it from $\mathcal{E}$, and, for SR, in outputting $s' \in \mathbb{F}_p$ upon receiving it from $\mathcal{F}_{\text{DP-AVSS}}$, if any.

The former is denoted the real execution $REAL_{\mathcal{A}}$ of the protocol, the latter the ideal execution $IDEAL_{\mathcal{F}_{\text{DP-AVSS}},\mathcal{S}}$.

Notice that the trivial protocol, requiring parties to take no action and SR to never output, implements $\mathcal{F}_{\text{DP-AVSS}}$ in the UC sense. To

see this, consider the simulator which does nothing, in particular, upon receiving a request from $\mathcal{F}_{\text{DP-AVSS}}$ to deliver the output to SR, does not respond to this request. Thus we now specify a desirable sub-class of protocols at which we are aiming.

Definition 2. For any integer L, we say that a protocol APSS is a **Dynamic Proactive Asynchronous Verifiable Secret Sharing with Refresh Latency in L Messages** if:

Security: It UC-implements $\mathcal{F}_{\text{DP-AVSS}}$

Structure: It is defined by three sub-protocols:
- APSS.Share (between SO and $C^{(1)}$): Upon receiving start, parties in $C^{(1)}$ perform APSS.Share with SO. Upon receiving "share", SO performs APSS.Share with $C^{(1)}$.
- APSS.Refresh (between any committee $C^{(e)}$, denoted "exiting", and any committee $C^{(e+1)}$, denoted "entering"). For every $e \geq 1$: upon receiving start, parties in $C^{(e+1)}$ perform APSS.Refresh as entering with $C^{(e)}$ as closing. Upon receiving refresh, parties in $C^{(e)}$ perform APSS.Refresh as closing, with $C^{(e+1)}$ as entering.
- APSS.Open (performed by any committee): For every $e \geq 1$, upon receiving "open", parties in $C^{(e)}$ and SR perform APSS.Open.

Liveness: is specified, only when SR is honest, as follows. In any given execution, let us denote as δ the maximum delay taken by the delivery of a message sent by any honest party to a honest party, via $\mathcal{F}_{\text{AUTH}}$. Notice that δ is possibly ∞. Then SR outputs in any execution in which:
- $\mathcal{A}$ allows all delivery requests from $\mathcal{F}_{\text{NIZK}}$ and $\mathcal{F}_{CA}$;
  [In our implementation]: the output (possibly $\perp$ if SO corrupt), of the single terminating reliable broadcast from SO to all parties, is delivered before $\mathcal{E}$ sends a signal, either "Refresh" or "Open", to parties in $C^{(1)}$;
- there is some $e_o \in \mathbb{N}^*$ such that $\mathcal{E}$ sends Open to all honest parties in $C^{(e_o)}$;
- for all $e \in [1, \ldots, e_o - 1]$ (possibly empty), $\{$all parties of $C^{(e)}$ receive refresh and all parties of $C^{(e+1)}$ receive start then publish some string (possibly $\perp$ if corrupt) on $\mathcal{F}_{CA}\}$. Furthemore, from this point, there is at least a delay $L\delta$ before any party of $C^{(e+1)}$ receives "refresh" or "open".
- all messages from honest players in $C^{(e_o)}$ to SR are delivered.

3 PUBLICLY VERIFIABLE SECRET SHARING

3.1 Overview. A technique for publicly verifiably sharing a secret s among n parties with threshold t, is suggested in [GMW91, §3.2]. It consists in generating a vector of shares of s under the Shamir secret-sharing scheme, then output (i) the n-sized vector of encryptions: for each $i \in [n]$, of the i-th share under P_i's public key; (ii) appended with a *NIZK proof of plaintext knowledge*, i.e., of: s and of the randomness fully explaining the vector of encrypted shares. Notice that such NIZKs are eased by the use of linearly homomorphic commitments, see section 5.2 for an example. The data structure obtained is known as a Publicly Verifiable Secret Sharing (PVSS), and we denote the algorithm as PVShare. To publicly verifiably open a PVSS, each party P_i outputs (i) a decryption of its share, (ii) appended with, roughly, a NIZK of correctness.

Then, on input any $t+1$ such publicly verifiable opening shares, the public reconstruction function of Shamir secret-sharing, dubbed as FinOpen, returns the secret s. For our use-case of Refresh, we enrich the construction with further ingredients:

(1) an algorithm, which we denote as PVReshare which, on input a ciphertext under one's key, decrypts it then generates (i) a publicly verifiable sharing of the decryption (ii) appended with, roughly, a NIZK of correctness. We denote the data structure output as a *publicly verifiable resharing* (PVR).
(2) parties will need to linearly combine $t+1$ such PVRs. To this end, we instantiate PVSS with any public key encryption scheme, denoted LHE, supporting at least one homomorphic linear combination.
(3) to guarantee robust re-sharing and reconstruction, we specify the LHE to be *committing*, i.e., perfectly correct, in the following sense. We require that after at least one homomorphic linear combination of verifiably encrypted plaintexts under a public key, the holder of the public key cannot possibly exhibit two conflicting verifiable decryptions. This analysis, which is concluded in Prop. 4, may be of independent interest in that it provides detailed specifications for proactive schemes based on PVSS [MZW+19; BGG+20; Gro21].
(4) we provide an elementary gadget, denoted ShSim , that takes as input any t opening shares (of corrupt parties) along with any $s \in \mathbb{F}_p$ (bogus shared secret) and perfectly simulates the $t+1$ remaining opening shares (of honest parties). It is enabled by perfect correctness of Shamir Sharing and of the LHE.
(5) Finally, the function denoted ShInfer perfectly infers the opening shares of corrupt parties from the honest shares. ShInfer may of independent interest, since it is not formalized in classical references on IT-secure MPC, to our knowledge.

In order to define perfect correctness, we specify determinized versions of the algorithms (highlighted in color with a tilde, e.g., $\widetilde{\text{KeyGen}}$ and $\widetilde{\text{Enc}}$, for convenience), i.e., those taking the randomness as input. Their randomness inputs are specified, for simplicity, to vary uniformly in some finite sets denoted R_{key} and R_{enc}, which emulates arbitrary spaces and distributions. This formalism also eases the description of the relations proven in NIZK.

3.2 Linearly Homomorphic Encryption. We now specify requirements on the public key encryption used, which we denote LHE. It has plaintext space $\mathbb{F}_p$, and roughly speaking, it should enable one homomorphic linear combination on $t+1$ ciphertexts, such that the decryption returns the linear combination of plaintexts. Such schemes over $\mathbb{F}_p$ supporting a limited number of linear operations were coined as "Semi-HE" by [BDOZ11], which should not be confused with "Somewhat HE". LHE is defined as the following list of spaces and algorithms.

- $s\mathcal{K}$ and $p\mathcal{K}$ the *spaces of secret and public keys*, R_{key} the *set of key randomness*, R_{enc} the *set of encryption randomness* and C the *ciphertext space*;
- $\widetilde{\textbf{KeyGen}}(\rho_{\text{key}} \in R_{\text{key}}) \to s\mathcal{K} \times p\mathcal{K}$ the *determinized key generation function*;
- $\textbf{KeyGen}() := \widetilde{\text{KeyGen}}(\rho_{\text{key}} \xleftarrow{\$} R_{\text{key}})$ the *key generation algorithm*;
- $\widetilde{\textbf{Enc}}(\text{pk} \in p\mathcal{K}, \rho_{\text{enc}} \in R_{\text{enc}}, x \in \mathbb{F}_p) \to C$ the *determinized encryption function*;

- $\mathbf{Enc}(\mathsf{pk}, x) := \widetilde{\mathsf{Enc}}\big(\mathsf{pk}, \rho_{\mathrm{enc}} \xleftarrow{\$} R_{\mathrm{enc}}, x\big)$ is the *encryption algorithm*
- $\mathbf{Dec}(\mathsf{sk} \in s\mathcal{K}, c \in C) \to \mathbb{F}_p$ the *decryption function*;
- $\boxplus : C \times C \to C$ and $\boxdot : \mathbb{F}_p \times C \to C$ the linearly homomorphic addition and scalar multiplication.

We require IND-CPA, i.e., any PPT $\mathcal{A}$ has negligible advantage in distinguishing whether it is interacting with the "left" oracle O^L which, when queried on a pair $(x^L, x^R) \in \mathbb{F}_p^2$, returns $\mathsf{Enc}(x^L)$ or, with the "right" one, O^R, which returns $\mathsf{Enc}(x^R)$.

Perfect Correctness. For $c \in C$, we say that $x \in \mathbb{F}_p$ is an *explainable plaintext* of c under some $\mathsf{pk} \in p\mathcal{K}$, if there exists $\rho_{\mathrm{enc}} \in R_{\mathrm{enc}}$ such that $c = \widetilde{\mathsf{Enc}}(\mathsf{pk}, \rho_{\mathrm{enc}}, x)$.

We say that $x' \in \mathbb{F}_p$ is an *explainable decryption* of c under some $\mathsf{pk} \in p\mathcal{K}$, if there exists $\rho_{\mathrm{key}} \in R_{\mathrm{key}}$ such that we have $(\mathsf{sk}, \mathsf{pk}) = \widetilde{\mathsf{KeyGen}}(\rho_{\mathrm{key}})$ for some sk, such that $\mathsf{Dec}(\mathsf{sk}, c) = x'$.

Definition 3. *Perfect correctness after one linear combination of size $t+1$* is the guarantee that, for any $\mathsf{pk} \in p\mathcal{K}$, for any up to $t+1$ ciphertexts $(c_i)_{i\in[t+1]} \in C^{t+1}$, for any explainable plaintexts $(x_i)_{i\in[t+1]} \in \mathbb{F}_p^{t+1}$ of them under pk, for any $(\lambda_i)_{i\in[t+1]} \in \mathbb{F}_p^{t+1}$, then, if there exists an explainable decryption y' under pk (not necessarily, if pk is incorrectly generated) of the homomorphic linear combination $\boxplus_{i\in[t+1]}(\lambda_i \boxdot c_i)$, then we must have $y' = \sum_{i=1}^{t+1} \lambda_i m_i$.

Instantiations. The scheme of Paillier supports one linear combination of size n as long as $np^2 < N/2$, where N is the plaintext modulus. The variant of el Gamal, with plaintext in the exponent, supports one linear combination as long as the discrete logarithms up to index np are efficiently computable. A variant with optimized decryption is constructed in [CL15]. Finally, we have the scheme BGV and also, a fortiori, the fully homomorphic ones. Notice that these latter schemes have correctness only if the randomness is sampled with low norm, although the distribution allows large norms with nonzero (but exponentially small) probability. Thus we have to further specify these schemes with randomnesses within explicit bounds, e.g., as required in [GLS15, §4.2].

3.3 Shamir Secret Sharing over $\mathbb{F}_p$. A dealer having a secret s, in order to distribute it to n parties, such that any t-subset of them receives uniform randomness, does as follows. It generates a random polynomial $Q \in \mathbb{F}_p[X]_t^{(0)}$ of degree at most t evaluating to 0 at 0. Then for each $i \in [n]$, it distributes to P_i the evaluation $(Q+s)(i)$, denoted as a *share*. Reconstruction of s from any $t+1$ shares follows from polynomial interpolation.

- $\mathsf{Eval}^{\mathcal{U}} : P \in \mathbb{F}_p[X]_t \to (P(i))_{i\in\mathcal{U}}$, for any subset $\mathcal{U} \subset [0, \ldots, n]$ is the map returning the evaluations of P at the points of $\mathcal{U}$.
  Conversely, any $t+1$ evaluations uniquely determine the polynomial P. More precisely, for every $t+1$-sized $\mathcal{U} \subset [0, \ldots, n]$, the *linear* map defined below and denoted $\mathsf{PolReco}^{\mathcal{U}}$ is an inverse to $\mathsf{Eval}^{\mathcal{U}}$.
- $\mathsf{PolReco}^{\mathcal{U}} : \mathbb{F}_p^{t+1} \longrightarrow \mathbb{F}_p[X]_t$, for any $t+1$-sized $\mathcal{U} \subset [0, \ldots, n]$, denoted *polynomial reconstruction*. It is defined as: for each $i \in \mathcal{U}$, set $\lambda_i^{\mathcal{U}}(X) := \prod_{j\in\mathcal{U}\setminus\{i\}} \frac{X-j}{i-j}$ denoted as the Lagrange polynomial. Then, $\mathsf{PolReco}^{\mathcal{U}}\big((s_i)_{i\in\mathcal{U}}\big) := \sum_{i\in\mathcal{U}} s_i.\lambda_i^{\mathcal{U}}(X)$. The rest follows all at once:
- $\widetilde{\mathbf{Share}}(Q \in \mathbb{F}_p[X]_t^{(0)}, s \in \mathbb{F}_p) := \mathsf{Eval}^{[n]}(s+Q)$, denoted *the determinized Shamir secret sharing*.
  Any n-sized vector of the form $(P(i))_{i\in[n]}$ is denoted as a *vector of shares of $P(0)$*, the latter being denoted the "secret". In particular, $\widetilde{\mathsf{Share}}(s)$ outputs a vector of shares of s. In particular, we have *linearity*: a linear combination coordinate-wise of vectors of shares, is a vector of shares of the linear combination of the secrets.
- $\mathbf{Share}(s \in \mathbb{F}_p) := \widetilde{\mathsf{Share}}\big(Q \xleftarrow{\$} \mathbb{F}_p[X]_t^{(0)}, s \in \mathbb{F}_p\big)$ the *secret-sharing algorithm*.
  By surjectivity of $\mathsf{Eval}^{\mathcal{U}}$ onto any $t+1$-subset $\mathcal{U} \subset [0, \ldots, n]$, we have in particular that, for any fixed $s \in \mathbb{F}_p$: the linear map $\widetilde{\mathsf{Share}}(*, s)$, followed by projection onto any t out of $[n]$ coordinates, is a surjection onto $\mathbb{F}_p^t$. Thus, any t shares produced by $\mathsf{Share}(s)$ vary *uniformly at random* in $\mathbb{F}_p^t$.
- $\mathbf{SReco}^{\mathcal{U}} := \mathsf{Eval}_{\{0\}} \circ \mathsf{PolReco}^{\mathcal{U}}$, for any $t+1$-sized $\mathcal{U} \subset [n]$, denoted the *reconstruction of secret* function. Concretely, on input $(s_i)_{i\in\mathcal{U}}$, it outputs $s := \sum_{i\in\mathcal{U}} \lambda_i^{\mathcal{U}} s_i$, where $\lambda_i^{\mathcal{U}} := \lambda_i^{\mathcal{U}}(0)$ are designated as the *Lagrange coefficients*. s is the unique secret of which the $(s_i)_{i\in\mathcal{U}}$ belong to a vector of shares.
- $\mathbf{ShSim}^{\mathcal{V}} := \mathsf{Eval}_{[n]\setminus\mathcal{V}} \circ \mathsf{PolReco}^{\{0\}\sqcup\mathcal{V}}$ for any t-sized $\mathcal{V} \subset [n]$, denoted the *simulation of shares* function. On input any $s \in \mathbb{F}_p$ and $(s_i)_{i\in\mathcal{V}} \in \mathbb{F}_p^t$, it outputs the unique $(s_i)_{i\in[n]\setminus\mathcal{V}}$ such that the concatenation $(s_i)_{i\in[n]}$ forms a vector of shares of s.
- $\mathbf{ShInfer}^{\mathcal{U}} := \mathsf{Eval}_{[n]\setminus\mathcal{U}} \circ \mathsf{PolReco}^{\mathcal{U}}$, for any $t+1$-sized $\mathcal{U} \subset [n]$, denoted the *inference of shares* function. On input any $(s_i)_{i\in\mathcal{U}} \in \mathbb{F}_p^{t+1}$, it outputs the unique $(s_i)_{i\in[n]\setminus\mathcal{U}}$ such that the concatenation $(s_i)_{i\in[n]}$ forms a vector of shares.

3.4 Publicly Verifiable Secret Sharing (PVSS) and Resharing (PVR). PVShare, on input $s \in \mathbb{F}_p$, returns a PVSS of it, i.e., a vector of Shamir shares encrypted under the list of public keys given as parameter $\mathbf{pk} = (\mathsf{pk}_i)_{i\in[n]} \in p\mathcal{K}^n$. PVReshare, on input $c^{\mathrm{in}} \in C$, decrypts it with sk_{in} then proceeds as in PVShare. PVPartOpen, on input $\boldsymbol{c}^{\mathbf{in}} \in C^n$, with parameter i, returns a decryption, under some secret key $\mathsf{sk}_{\mathrm{in}}$, of the i-th coordinate of $\boldsymbol{c}^{\mathbf{in}}$. All three algorithms also append, to their output, a NIZK explaining it.

- $\widetilde{\mathbf{PVShare}}\Big((\mathsf{pk}_i)_{i\in[n]} \in p\mathcal{K}^n, \boldsymbol{\rho}_{\mathbf{enc}} \in R_{\mathrm{enc}}^n, Q \in \mathbb{F}_p[X]_t^{(0)}, s \in \mathbb{F}_p\Big) \in C^n$ denoted the *determinized public secret sharing function*: set $(s_i)_{i\in[n]} := \widetilde{\mathsf{Share}}(Q, s)$, output $\boldsymbol{c} := \Big(\widetilde{\mathsf{Enc}}(\mathsf{pk}_i, \rho_{\mathrm{enc},i}, s_i)\Big)_{i\in[n]}$.
- $\mathbf{PVShare}\Big(\mathbf{pk} \in p\mathcal{K}^n, s \in \mathbb{F}_p\Big) \in (C^n, \Pi) \in \mathrm{PVSS}$, denoted the *publicly verifiable secret-sharing algorithm*: sample $\boldsymbol{\rho}_{\mathbf{enc}} \xleftarrow{\$} R_{\mathrm{enc}}^n$ and $Q \xleftarrow{\$} \mathbb{F}_p[X]_t^{(0)}$, set $\boldsymbol{c} := \widetilde{\mathsf{PVShare}}\big(\mathbf{pk}, \boldsymbol{\rho}_{\mathbf{enc}}, Q, s\big) \in C^n$. Output $(\boldsymbol{c}, \pi)$ where π is a proof of secret knowledge for $\boldsymbol{c}$, as specified below.
- $\mathbf{PVReshare}\Big(\mathbf{pk} \in p\mathcal{K}^n, \mathsf{sk}_{\mathrm{in}} \in s\mathcal{K}, \boldsymbol{\rho}_{\mathbf{enc}} \in R_{\mathrm{enc}}^n, Q \in \mathbb{F}_p[X]_t^{(0)}, c^{\mathrm{in}} \in C\Big) \in C^n$, denoted the *determinized public re-sharing function*: set $s := \mathsf{Dec}(\mathsf{sk}, c^{\mathrm{in}})$, output $\boldsymbol{c}^{\mathbf{out}} := \widetilde{\mathsf{PVShare}}\big(\mathbf{pk}, \boldsymbol{\rho}_{\mathbf{enc}}, Q, s\big)$.
- $\mathbf{PVReshare}\Big(\mathbf{pk} \in p\mathcal{K}^n, \mathsf{pk}_{\mathrm{in}} \in p\mathcal{K}, \mathsf{sk}_{\mathrm{in}} \in s\mathcal{K}, c^{\mathrm{in}} \in C\Big) \in (C^n, \Pi)$, denoted the *publicly verifiable re-sharing algorithm*: sample $\boldsymbol{\rho}_{\mathbf{enc}} \xleftarrow{\$} R_{\mathrm{enc}}^n$ and $Q \xleftarrow{\$} \mathbb{F}_p[X]_t^{(0)}$, then output $(\boldsymbol{c}^{\mathbf{out}}, \pi)$, where $\boldsymbol{c}^{\mathbf{out}} := \widetilde{\mathsf{PVReshare}}\big(\mathbf{pk}, \mathsf{sk}_{\mathrm{in}}, \boldsymbol{\rho}_{\mathbf{enc}}, Q, c^{\mathrm{in}}\big)$. And π is a NIZK proof

of decryption-then-reshare, specified below [which NIZK proves, in particular, correct generation of $(\mathsf{pk}_{\text{in}}, \mathsf{sk}_{\text{in}})$].

- **PVPartOpen**$\big(i \in [n], \mathsf{pk}_{\text{in}} \in p\mathcal{K}, \mathsf{sk}_{\text{in}} \in s\mathcal{K}, \boldsymbol{c^{\text{in}}} = (c_j^{\text{in}})_{j\in[n]} \in C^n\big)$, denoted *publicly verifiable partial opening of the i-th coordinate of $\boldsymbol{c^{in}}$ under key* pk_{in}. Output (i, s_i, π), where $s_i := \mathsf{Dec}(\mathsf{sk}, c_i^{\text{in}})$ and $\pi \in \Pi$ is a NIZK *of correct decryption of $c_i^{in} \in C$ into $s_i \in \mathbb{F}_p$*.
- **FinOpen**$^{\mathcal{U}} := \mathsf{SReco}^{\mathcal{U}}$, for any $t+1$-subset $\mathcal{U} \subset [n]$, dubbed as *final opening*.

3.5 Definitions of Explainable Secret/Shares/Partial openings, and of the NIZKs. In the following definitions we leave implicit the public parameter of $\mathbf{pk} \in p\mathcal{K}^n$, which will be the keys of the entering committee, in Refresh.

For any $\boldsymbol{c} \in C^n$, we say that $s' \in \mathbb{F}_p$ is an *explainable plaintext secret* of $\boldsymbol{c}$ if there exists such $\boldsymbol{\rho'_{\text{enc}}} \in R_{\text{enc}}^n$ and $Q' \in \mathbb{F}_p[X]_t^{(0)}$ such that $\boldsymbol{c} := \widetilde{\mathsf{PVShare}}\big((\mathsf{pk}_i)_{i\in[n]}, \boldsymbol{\rho'_{\text{enc}}}, Q', s'\big)$. For such Q' and s', we then denote the $\big(s'_i := (Q' + s')(i)\big)_{i\in[n]}$ as *explainable plaintext shares* of $\boldsymbol{c}$.

For any $\boldsymbol{c} \in C^n$, we define as a *NIZK of plaintext secret*, (and thus automatically of plaintext shares), for $\boldsymbol{c}$, as a NIZK of such $s' \in \mathbb{F}_p$, $\boldsymbol{\rho'_{\text{enc}}} \in R_{\text{enc}}^n$ and $Q' \in \mathbb{F}_p[X]_t^{(0)}$.

For $\boldsymbol{c^{\text{out}}} \in C^n$, $\mathsf{pk}_{\text{in}} \in p\mathcal{K}$ and $c^{\text{in}} \in C$, we say that $s' \in \mathbb{F}_p$ is an *explainable decrypted-then-reshared secret* of c^{in} into $\boldsymbol{c^{\text{out}}}$ if there exists some $\Big\{\mathsf{sk}'_{\text{in}} \in s\mathcal{K},\ (\rho_{\text{key}})' \in R_{\text{key}},\ s' \in \mathbb{F}_p,\ (\boldsymbol{\rho_{\text{enc}}})' \in R_{\text{enc}}^n$ and $Q' \in \mathbb{F}_p[X]_t^{(0)}$, such that $(\mathsf{sk}_{\text{in}}, \mathsf{pk}_{in}) = \widetilde{\mathsf{KeyGen}}(\rho_{\text{key}})$, letting $s' := \mathsf{Dec}(\mathsf{sk}'_{\text{in}}, c^{\text{in}})$, then $\boldsymbol{c^{\text{out}}} = \widetilde{\mathsf{PVReshare}}(\mathbf{pk}, \mathsf{sk}_{\text{in}}, \rho'_{\text{enc}}, Q', c^{\text{in}})\Big\}$.

We then denote the $(s'_i)_{i\in[n]} \in \mathbb{F}_p^n$ as *explainable re-sharing shares* of c^{in} into $\boldsymbol{c^{\text{out}}}$.

Notice that such s' is then, in particular, an explainable decryption of c^{in} under pk_{in}.

We define as a *NIZK of decrypted-then-reshared secret of c^{in} into $\boldsymbol{c^{out}}$, under key* pk_{in}, and thus automatically of *plaintext re-sharing shares*, a NIZK of such sk'_{in}, $(\rho_{\text{key}})'$, s', $(\boldsymbol{\rho_{\text{enc}}})'$ and $Q' \in \mathbb{F}_p[X]_t^{(0)}$.

For $\boldsymbol{c^{\text{in}}} = (c_i^{\text{in}})_{i\in[n]} \in C^n$, $s_i \in \mathbb{F}_p$ and $\mathsf{pk}_{\text{in}} \in p\mathcal{K}$, we dub: "$s_i$ is an explainable decryption of c_i under pk_{in}", as: "s_i is an *explainable opening share of the i-th coordinate of $\boldsymbol{c^{in}}$ under* pk_{in}"

3.6 Data Structures. The following are meant to capture the well-formed outputs of PVShare, PVReshare and PVPartOpen.

- **PVSS** is the data structure of pairs $(\boldsymbol{c} \in C^n, \pi \in \Pi)$ where π is a proof of secret knowledge for $\boldsymbol{c}$.
- **PVR**$(\mathsf{pk}_{\text{in}}, c^{\text{in}} \in C)$: *publicly verifiable resharings of c^{in} under key* pk_{in}, is the pairs: $(\boldsymbol{c^{\text{out}}} \in C^n, \pi \in \Pi)$ where π is a NIZK of re-shared secret knowledge for $\boldsymbol{c^{\text{out}}}$, from c^{in}, under key pk_{in}.
- **PVPO**$\big(i \in [n],\ \boldsymbol{c^{\text{in}}} = (c_i^{\text{in}})_{i\in[n]} \in C^n\big)$: *publicly verifiable opening shares of the i-th coordinate of $\boldsymbol{c^{in}}$ under key* pk_{in}, is the triples $(i, s_i \in \mathbb{F}_p, \pi \in \Pi)$ where π is a proof of correct decryption of c_i^{in} into s_i, under key pk_{in}. In particular, s_i is an explainable opening share of the i-th coordinate of $\boldsymbol{c^{\text{in}}}$ under pk_{in}. We deliberately omit the opening key pk_{in} in the notation, since in our use-case it will the public key of the i-th party of the opening committee.

Notice that a PVR proves in particular knowledge of a plaintext secret, thus is a fortiori a PVSS.

3.7 Properties. From black-box linearity of Shamir secret sharing and perfect correctness of LHE after one homomorphic combination, we have:

Property 4 (Homomorphic Combination Commutes with Threshold Opening)**.** For any set of $t+1$ PVR: from ciphertexts $(c^{(i)})_{i\in[t+1]}$, under $\mathbf{pk} \in p\mathcal{K}^n$, into vectors of ciphertexts, of which we denote $(\boldsymbol{c^{(i)}} \in (C^n))_{i\in[t+1]}$ the vectors of ciphertexts. Consider, for every i: $s_i \in \mathbb{F}_p$ an explainable decrypted then re-shared secret of $c^{(i)}$, along with a vector $\boldsymbol{s^{(i)}} \in \mathbb{F}_p^n$ of explainable plaintext shares of it. Then we have that for any coefficients $(\lambda_i)_{i\in[t+1]}$,

(1) the coordinates-wise linear combination $s := \boxplus_{i=1}^{t+1}(\lambda_i \boxdot \boldsymbol{c^{(i)}})$ has a unique explainable decryption $\boldsymbol{s} \in \mathbb{F}_p^n$, which is furthermore a vector of shares, for the secret $\sum_{i=1}^{t+1} s^{(i)}$.
(2) In particular, for any t-sized $\mathcal{V} \subset [n]$, we have $\mathsf{ShSim}(s, (s_j)_{j\in\mathcal{V}}) = (s_j)_{j\in[n]\setminus\mathcal{V}}$.
(3) In particular, for any $t+1$-sized $\mathcal{U} \subset [n]$, we have $\mathsf{ShInfer}((s_j)_{j\in\mathcal{U}}) = (s_j)_{j\in[n]\setminus\mathcal{U}}$.

The following is given a more detailed statement and proof in Appendix B, which might be of independent interest.

Property 5 (IND-CPA)**.** Any PPT machine has negligible advantage in distinguishing between the PVShare of two chosen secrets.

4 PROTOCOL DP-AVSS

In §4.1 we give an overview of DP-AVSS. Then we detail DP-AVSS: sharing §4.2, then Refresh §4.3 and opening §4.5. In §4.6 we state liveness and UC security of DP-AVSS, and prove them.

4.1 Overview

DP-AVSS revolves around the secure generation of objects, in each epoch e, which we denote as Verified Proactivized Sharing: VPS$^{(e)}$ (Def. 6). A VPS$^{(e)}$ simply consists of (i) a n-sized vector of LHE ciphertexts, encrypted under the n public keys of committee $C^{(e)}$, (ii) appended with the signatures of a quorum of $t+1$ parties of $C^{(e)}$. In each committee $C^{(e)}$, each party $P_i^{(e)}$ forms a local list, denoted $V_i^{(e)}$, of the various VPS$^{(e)}$'s it has received or formed so-far. DP-AVSS maintains the **[Correctness]** invariant, that every VPS ever formed in the execution, is equal to encryptions of a consistent set of n Shamir shares of the same secret s, this secret being furthermore the one of the secret owner SO if it is honest. DP-AVSS maintains the **[Liveness]** invariant that, as soon as epochs take at least 3 actual messages delay, then in each committee $C^{(e)}$, the intersection of all local lists $V_i^{(e)}$ of the $t+1$ honest parties is nonempty. Together these two invariants guarantee that s can be reconstructed in any epoch e. Remarkably, parties are not aware of which VPS$^{(e)}$s they have in common, since agreement on a common subset is impossible for $t \geq n/3$. Before we proceed with DP-AVSS step by step, let us finally mention that, unlike a PVSS or a PVR , a VPS is not appended with any NIZK proof. Instead, what will guarantee its correctness, is that in the quorum of $t+1$ signers, at least one honest party verified that the VPS had been correctly formed.

To share its secret, SO generates a PVSS of it, encrypted under the keys of the first committee $C^{(1)}$, broadcasts it to them, then goes offline. At this point, all honest parties in $C^{(1)}$ have the same view on one single n-sized vector of ciphertexts, and furthermore the NIZK appended guarantees that their plaintexts constitute a well-formed set of n Shamir shares of some given secret s. We thus abuse notations and also designate this initial PVSS as (the unique) VPS of epoch $e = 1$.

For each $e \geq 1$, we now describe DP-AVSS-Refresh between exiting committee $C^{(e)}$ and entering committee $C^{(e+1)}$. The goal is that at the end, all honest parties in $C^{(e+1)}$ receive at least one $\mathsf{vps}^{(e+1)} \in \mathsf{VPS}^{(e+1)}$ in common. To this end, parties of $C^{(e+1)}$ engage into n parallel instances of a protocol denoted Leader-Based-Refresh. Each instance is led by a party $P_k^{(e+1)}$ of $C^{(e+1)}$, denoted *leader*, which will drive LB-refresh in just 2 messages. They are also depicted in Figure 2.

Contribution Phase. Each party $P_i^{(e)}$ of $C^{(e)}$, for each $\mathsf{vps}^{(e)} \in \mathsf{VPS}^{(e)}$ which it has: decrypts its i-th share of $\mathsf{vps}^{(e)}$ then generates a publicly verifiable re-sharing of it, i.e., a PVR. It sends it to the leader $P_k^{(e+1)}$, wrapped in a "contribution message". This is the only action required from parties in $C^{(e)}$. $P_k^{(e+1)}$ waits until it receives $t+1$ correct contributions from $t+1$ distinct parties, such that they furthermore all originate from the *same* $\mathsf{vps}^{(e)}$ (which is publicly verifiable from the contributions). Then, it computes homomorphically the linear Lagrange reconstruction formula on these $t+1$ vectors of encrypted shares. It thus obtains a n-sized vector of encrypted shares of the same secret s. Notice that this formula, which is the one of SReco, is also the same as in the baseline of the Introduction, except that now it is computed homomorphically on all n shares at once. We denote this task as Reshare.Combine.

Verification Phase. $P_k^{(e+1)}$ then multicasts to $C^{(e+1)}$ the previously obtained combined n-sized vector of encrypted shares. It appends to it the concatenation of the $t+1$ input PVRs. Upon receiving the whole, each party in $C^{(e+1)}$ checks: that all the $t+1$ appended PVRs originate from the *same* $\mathsf{vps}^{(e)}$, and correctness of the homomorphic linear combination of them. It then, signs the n-sized vector of encrypted shares (but not the $t+1$ input PVRs) then multicasts it. Any party, upon collecting any $t+1$ signatures, on the same vector of encrypted shares, thereby obtains a $\mathsf{vps}^{(e+1)} \in \mathsf{VPS}^{(e+1)}$, as desired.

Final Remark: LB-refresh *Does Not Have Consistency.* A corrupt leader could well generate two correctly formed n-sized vectors of encrypted shares from the contributions of two (possibly equal) sets of $t+1$ parties $\mathcal{U}$ and $\mathcal{U}'$ in $C^{(e)}$. Then it would send each vector separately to two honest parties $P_i^{(e+1)}$ and $(P_i^{(e+1)})'$ of $C^{(e+1)}$ to obtain their signatures. Since corrupt parties can sign twice, this is enough to obtain a quorum of $t+1$ signatures on each of these two distinct vectors. Nevertheless, existence of the signature of at least one honest party on each of them, certifies both of them to be correct combinations of $t+1$ encrypted re-sharings.

4.2 DP-AVSS-Share

At the beginning of the protocol, i.e. at epoch e=1, every party $P_i^{(1)} \in C^{(1)}$ generates $(\mathsf{pk}_i^{(1)}, \mathsf{sk}_i^{(1)}) \leftarrow \mathsf{KeyGen}()$ then registers it to $\mathcal{F}_{CA}$. Next, when all n keys have been published, the Secret Owner (SO) retrieves them, denoting them as $\mathbf{pk}^{(1)} = \{\mathsf{pk}_i^{(1)}\}_{i\in[n]}$ (including $\perp$ for non correctly published ones) and generates $\mathsf{pvss} \leftarrow \mathsf{PVShare}(\mathbf{pk}^{(1)}, s)$ from its secret input s. Then, SO broadcasts pvss to $C^{(1)}$ and shuts-off. [This broadcast is the only one in the protocol, it could be actually traded for a simple asynchronous multicast, without downgrading any guarantee of DP-AVSS for a honest SO. However, for a corrupt SO, then we would not have anymore consistency of the reconstructed secret, which is unavoidable since asynchronous consistent broadcast is impossible for $t \geq n/3$.]

Each party $P_i^{(1)} \in C^{(1)}$, upon receiving an output of the broadcast of SO: if it is a $\mathsf{pvss} \in \mathsf{PVSS}$, then it sets its local list of $\mathsf{VPS}_i^{(1)}$ of epoch 1 as $\{\mathsf{pvss}\}$. Else, it sets it as $\{\mathsf{pvss}_0\}$ where pvss_0 is a fixed predefined PVSS of 0.

4.3 Leader-Based-refresh: LB-refresh

Data Structures.

Definition 6 (Verified proactivized sharing)**.** A verified proactivized sharing (VPS) is a triple $(e, \boldsymbol{c}, qvc)$, where $\boldsymbol{c}$ is a vector of ciphertexts. *If* $e \geq 2$, then qvc is a *Quorum verification certificate*, i.e. a set (or a short aggregation) of $t+1$ signatures from members of $C^{(e)}$ on $(e, \boldsymbol{c})$. *Else if* $e = 1$, then $\mathsf{vps}^{(1)}$ is a PVSS from SO (thus $qvc = \perp$).

For any $e \in \mathbb{N}^*$ we denote the subset of VPS with first argument equal to e the "VPS's related to epoch e", denoted $\mathsf{VPS}^{(e)}$. We abuse the notations in that, for $\mathsf{vps}^{(e)} = (e, \boldsymbol{c}, qvc)$, we often designate the $\boldsymbol{c}$ enclosed also as "$\mathsf{vps}^{(e)}$" . Likewise, we say that two $\mathsf{vps}^{(e)} \in \mathsf{VPS}^{(e)}$ are the same as soon as the $\boldsymbol{c}$ enclosed are the same, nonwithstanding they can carry different sets of $t+1$ signatures.

We also formalize the following intermediary data structures:

$\mathsf{contribMsg}_i$ is the data structure of triples $(e, \mathsf{vps}^{(e)}, \mathsf{pvr}^{(i)})$, where: $\mathsf{vps}^{(e)} \in \mathsf{VPS}^{(e)}$ and $\mathsf{pvr}^{(i)} \in \mathsf{PVR}(\mathsf{pk}_i^{(e)}, c_i := \mathsf{vps}^{(e)}[i])$.

combineMsg is the data structure of quadruples $(\boldsymbol{c}, \mathsf{vps}^{(e)}, \{\mathsf{pvr}^{(i)}\}_{i\in\mathcal{U}}, \mathcal{U})$, where: $\boldsymbol{c} \in C^n$, $\mathsf{vps}^{(e)} \in \mathsf{VPS}^{(e)}$, each $\mathsf{pvr}^{(i)} \in \mathsf{PVR}(\mathsf{pk}_i^{(e)}, c_i := \mathsf{vps}[i])$, and $\mathcal{U} \subseteq [n]$ is a subset of $t+1$ indices of parties.

$\mathsf{verifMsg}_i$ is the data structure of triples $(e, \boldsymbol{c}, sign_i)$, where: $e \in \mathbb{N}^*$ denotes the epoch counter, $\boldsymbol{c} \in C^n$, and $sign_i$ is a signature of $P_i^{(e)}$ on $(e, \boldsymbol{c})$.

$\mathsf{openMsg}_i$ related to some $e_o \in \mathbb{N}^*$ are $\mathsf{PVPO}(i, \mathsf{vps}^{(e_o)} \in \mathsf{VPS}^{(e_o)})$ under key $\mathsf{pk}_i^{(e_o)}$.

LB-refresh. Let us define the LB-refresh sub-protocol between an "exiting" committee $C^{(e)}$ and an "entering" committee $C^{(e+1)}$, and a designated leader in $C^{(e+1)}$. Parties will locally compute the following functions:

$\mathsf{Reshare.Contrib}(i, \mathsf{vps}^{(e)}, \mathbf{pk}^{(e+1)}, \mathsf{sk}_i^{(e)})$**:** On input a $\mathsf{vps}^{(e)} = (e, \boldsymbol{c}, qvc)$ and a set of keys $\mathbf{pk}^{(e+1)} = \{\mathsf{pk}_i^{(e+1)}\}_{i\in[n]}$, compute $\mathsf{pvr}^{(i)} = \mathsf{PVReshare}(\mathbf{pk}^{(e+1)}, \mathsf{sk}_i^{(e)}, \boldsymbol{c})$, output $(e, \mathsf{vps}^{(e)}, \mathsf{pvr}^{(i)}) \in \mathsf{contribMsg}_i$.

Reshare.Combine $(\{(e, \mathsf{vps}^{(e)}, \mathsf{pvr}^{(i)} = (\boldsymbol{c^{(i)}}, \pi^{(i)})) \in \mathsf{contribMsg}_i\}_{i\in\mathcal{U}})$: On input a set of any $t+1$ contributions from a $(t+1)$-subset $\mathcal{U} \subset [n]$ of parties, compute coordinate-wise the homomorphic linear combination

$$\boldsymbol{c'} := \boxplus_{i\in\mathcal{U}}\left(\lambda_i^{\mathcal{U}} \boxdot \boldsymbol{c^{(i)}}\right) \tag{1}$$

(which is none other than the one defining SReco in §3.3). Output $(\boldsymbol{c'}, \mathsf{vps}^{(e)}, \{\mathsf{pvr}^{(i)}\}_{i\in\mathcal{U}}, \mathcal{U}) \in \mathsf{combineMsg}$.

Verify.Contrib$((\boldsymbol{c}, \mathsf{vps}^{(e)}, \{\mathsf{pvr}^{(i)}\}_{i\in\mathcal{U}}, \mathcal{U}) \in \mathsf{combineMsg}, \mathsf{sk}_i^{(e+1)})$: Test if the combineMsg is well formed, test if Eq. 1 is true. If one of the test fails, output "reject", else, create a signature $sign_i$ on $(e+1, \boldsymbol{c} \in C^n)$ and output $(e+1, \boldsymbol{c}, sign_i) \in \mathsf{verifMsg}_i$.

Verify.Combine$\left(\{(e+1, \boldsymbol{c} \in C^n, sign_i) \in \mathsf{verifMsg}_i\}_{i\in\mathcal{U}'}\right)$ for any $t+1$-subset $\mathcal{U}' \subset [n]$, for the same $\boldsymbol{c} \in C^n$, output $\big(e+1, \boldsymbol{c}, qvc \leftarrow \mathsf{Sign.Combine}(\{sign_i\}_{i\in\mathcal{U}'})\big) \in \mathsf{VPS}^{(e+1)}$.

The Leader-Based-refresh subprotocol is described in Algorithm 1.

4.4 DP-AVSS-Refresh

We can now present in Figure 3 the overall Refresh phase between an exiting committee $C^{(e)}$ and an entering committee $C^{(e+1)}$.

4.5 DP-AVSS-Open

Each $P_i^{(e_o)} \in C^{(e_o)}$, upon receiving the signal "open", for each for each $\mathsf{vps}^{(e_o)} \in V_i^{(e_o)}$, generates a publicly verifiable opening share $(i, \mathbf{d}_i, \pi_i) \leftarrow \mathsf{PVPartOpen}(i, \mathsf{pk}_i^{(e_o)}, \mathsf{sk}_i^{(e_o)}, \boldsymbol{c})$, and sends a message $(\mathsf{vps}^{(e_o)}, d_i, \pi_i) \in \mathsf{openMsg}_i$ to SR through $\mathcal{F}_{\mathrm{AUTH}}$. Next, SR upon receiving any $t+1$ $\mathsf{openMsg}_i$ on a same $\mathsf{vps}^{(e_o)}$ from a set $\mathcal{U}$ of $t+1$ indices, computes $s = \mathsf{FinOpen}(\{d_i\}_{i\in\mathcal{U}})$ and terminates.

4.6 Analysis

Theorem 7. *Consider committees of $n = 2t+1$ parties, in each of which t are maliciously corrupt by a polynomial mobile adversary, in an asynchronous communication network in the bulletin board PKI model. Then protocol* DP-AVSS *UC implements $\mathcal{F}_{\text{DP-AVSS}}$ with refresh latency in 3 messages.*

4.6.1 *Proof of Liveness.*

Algorithm 1 LB-refresh$\left(C^{(e)}, C^{(e+1)}, P_k^{(e+1)} \in C^{(e+1)}\right)$

The following set of instructions apply to all parties in $C^{(e)}$ and $C^{(e+1)}$. Party $P_k^{(e+1)} \in C^{(e+1)}$, denoted "leader", follows additional instructions.

Inputs: Each $P_i^{(e)} \in C^{(e)}$ has $(\mathsf{pk}_i^{(e)}, \mathsf{sk}_i^{(e)})$ and computed a list $\mathcal{M}_i^{(e)} := \left\{(e, \mathsf{vps}^{(e)}, \mathsf{pvr}^{(i)}) \in \mathsf{contribMsg}_i, \quad \text{for} \quad \mathsf{vps}^{(e)} \in V_i^{(e)}\right\}$. Each $P_i^{(e+1)} \in C^{(e+1)}$ has a pair $(\mathsf{pk}_i^{(e+1)}, \mathsf{sk}_i^{(e+1)})$.

Output: Each $P_i^{(e+1)} \in C^{(e+1)}$ outputs at most one $\mathsf{vps}^{(e+1)}$.

Contribution Phase

1: Each $P_i^{(e)} \in C^{(e)}$ sends each $m_i \in \mathcal{M}_i$ to the leader $P_k^{(e+1)}$.

2: Leader $P_k^{(e+1)}$, upon receiving, for the first time, a set of messages $C_{\mathcal{U}} := \left\{(e, \mathsf{vps}^{(e)}, \mathsf{pvr}^{(i)}) \in \mathsf{contribMsg}_i \text{ for } i \in \mathcal{U}\right\}$ for some $(t+1)$-subset $\mathcal{U} \subset [n]$, all with the same $\mathsf{vps}^{(e)} \in \mathsf{VPS}^{(e)}$:

3: Multicast Reshare.Combine$(C_{\mathcal{U}})$

Verification Phase

4: Each party $P_i^{(e+1)} \in C^{(e+1)}$, upon receiving a combineMsg m from $P_k^{(e+1)}$:

5: Compute Verify.Contrib$(m, \mathsf{sk}_i^{(e+1)})$ and, if the output is not "reject", multicast the output.

6: Each party $P_i^{(e+1)} \in C^{(e+1)}$, upon receiving a set $\mathcal{V}$ of messages of the type $\mathsf{verifMsg}_i$ for the same $(e+1, \boldsymbol{c} \in C^n)$ from a subset $\mathcal{U}' \subset [n]$ of $t+1$ parties in $C^{(e+1)}$:

7: Append Verify.Combine$(\mathcal{V})$ to $V_i^{(e+1)}$.

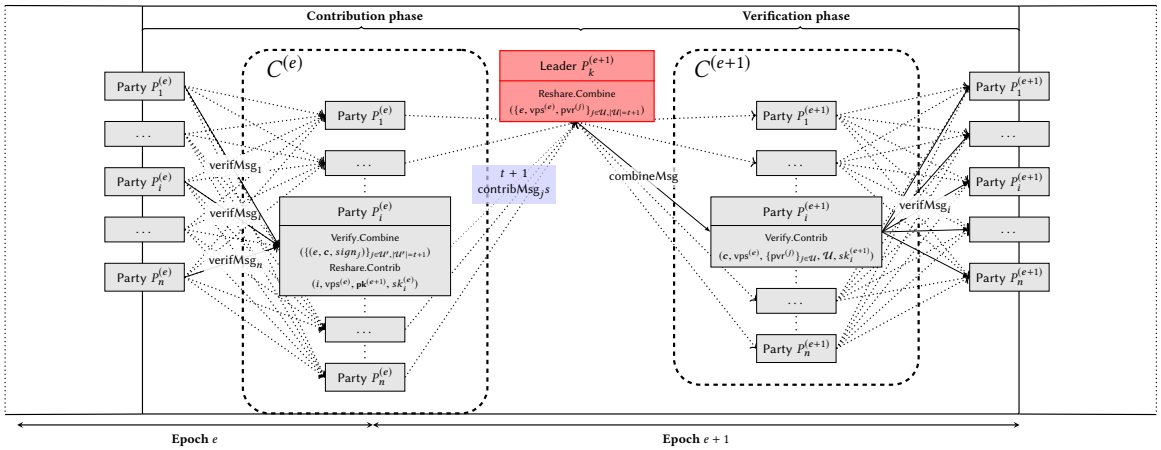

Figure 2: LB-refresh **for a party $P_i^{(e)}$ in a committee $C^{(e)}$. It received a set of $t+1$** verifMsg **from at least $t+1$ parties in $C^{(e)}$. Together they form a verified proactivized sharing** $\mathsf{vps}^{(e)}$ **for epoch e. It then computes a resharing contribution. A leader $P_k^{(e+1)}$ collects $t+1$ such contributions, verifies their correctness and combines them. Finally, a party $P_i^{(e+1)}$ in a new committee $C^{(e+1)}$ verifies the output of the contribution phase, signs it and multicasts the signed output to parties in $C^{(e+1)}$.**

Refresh($C^{(e)}, C^{(e+1)}$)

Inputs: Each party $P_i^{(e)}$ in $C^{(e)}$ has 1) a key pair $(\mathsf{pk}_i^{(e)}, \mathsf{sk}_i^{(e)})$, and 2) a set $V_i^{(e)}$ of at most n $\mathsf{vps}^{(e)} \in \mathrm{VPS}^{(e)}$.
Each party $P_i^{(e+1)} \in C^{(e+1)}$ starts with an empty state.

(PKI) At signal start, each party $P_i^{(e+1)} \in C^{(e+1)}$: generates a key pair $(\mathsf{sk}_i^{(e+1)}, \mathsf{pk}_i^{(e+1)}) \leftarrow$ KeyGen(), publishes $(\mathsf{pk}_i^{(e+1)})$ to $\mathcal{F}_{CA}$, then starts n parallel instances of LB-refresh($C^{(e)}, C^{(e+1)}, k$) for $k \in [n]$. It plays in addition the role of the leader for $k = i$.

(Reshare) At signal refresh, each $P_i^{(e)} \in C^{(e)}$: waits until all keys have been published, then for each $\mathsf{vps}^{(e)} \in V_i^{(e)}$, generates Reshare.Contrib($i, \mathsf{vps}^{(e)}, \mathbf{pk}^{(e+1)}, \mathsf{sk}_i^{(e)}$) $\in \mathsf{contribMsg}_i$, and stores them in a list $\mathcal{M}_i^{(e)}$. Then for each party $P_k^{(e+1)}$ of $C^{(e+1)}$, it executes LB-refresh($C^{(e)}, C^{(e+1)}, k$), i.e. sends to it all $m_i \in \mathcal{M}_i^{(e)}$. Then it shuts-off.

(Output): At signal refresh (for epoch $e+1$), each party $P_i^{(e+1)} \in C^{(e+1)}$ stops all n pending instances of LB-refresh($C^{(e)}, C^{(e+1)}, k$) for $k \in [n]$. Its set $V_i^{(e+1)}$ of $\mathrm{VPS}^{(e+1)}$ is thus the concatenation of the outputs (if any) obtained in each of the instances.

Figure 3: Refresh($C^{(e)}, C^{(e+1)}$)

Lemma 8. *For any $e \geq 1$, if all $t+1$ honest parties in $C^{(e)}$ own a $\mathsf{vps}^{(e)}$ when they receive the signal* refresh*, consider an execution of* Refresh *uninterrupted during at least 3δ, then, it will exist at least one $\mathsf{vps}^{(e+1)} \in \mathrm{VPS}^{(e+1)}$ that all parties $P_i^{(e+1)} \in C^{(e+1)}$ will have in their local lists $V_i^{(e+1)}$.*

Proof. A honest leader $P_k^{(e+1)}$ is guaranteed to receive $t+1$ correct contributions on $\mathsf{vps}^{(e)}$ within δ. The combineMsg that it makes from them, will be received by all $t+1$ honest parties in $C^{(e+1)}$ within δ. All their $t+1$ signatures on it will be received by all $t+1$ honest parties in $C^{(e+1)}$ within δ, enabling them to form the same $\mathrm{VPS}^{(e+1)}$ (possibly with different sets of $t+1$ signatures). □

4.6.2 Proof of Security, i.e., of UC implementation of $\mathcal{F}_{\text{DP-AVSS}}$. We describe a simulator $\mathcal{S}$, whose behavior depends on whether SO is corrupt or honest (and on SR, to a much lesser extend). We note $\mathcal{S}_h$ the behavior of $\mathcal{S}$ in the case of an honest SO, and $\mathcal{S}_c$ in the case of a corrupt SO. We start with a description of $\mathcal{S}_h$. The simulator runs an internal copy of $\mathcal{F}_{\text{BC}}$, $\mathcal{F}_{\text{AUTH}}$, $\mathcal{F}_{\text{NIZK}}$ and $\mathcal{F}_{CA}$. It simulates a correct behavior of them, except that $\mathcal{F}_{\text{NIZK}}$ does not check validity of the witness given by honest parties, nor from SO if it is honest. Importantly, $\mathcal{S}$ erases from its memory the witnesses received from the simulated corrupt parties on behalf of $\mathcal{F}_{\text{NIZK}}$, straight after having verified them. Thus, $\mathcal{S}$ does not perform any extraction.

$\mathcal{S}_h$

Setup $\mathcal{S}$ collects the public keys $(\mathsf{pk}_i^{(1)})_{i \in \mathcal{I}^{(1)}}$ of corrupt parties in $C^{(1)}$ on behalf of the simulated $\mathcal{F}_{CA}$. Non-correctly received keys are set to $\perp$. For every simulated honest party $P_i^{(1)} \in C^{(1)}$, it generates $(\mathsf{sk}_i^{(1)}, \mathsf{pk}_i^{(1)}) \leftarrow$ KeyGen(), which it registers on $\mathcal{F}_{CA}$. It sets $\mathbf{pk} = (\mathsf{pk}_1^{(1)}, \ldots, \mathsf{pk}_n^{(1)})$.

Share $\mathcal{S}_h$ assigns input $\tilde{s} := 0$ to the simulated honest SO, which correctly does the DP-AVSS.Share, i.e., broadcasts a PVSS of 0 encrypted under $\mathbf{pk}$.

Refresh For $e = 1$: if the Refresh signal is received before $\mathcal{A}$ allows the delivery of the broadcast from the simulated honest SO to all simulated honest parties, then $\mathcal{S}_c$ stops. For each epoch e: $\mathcal{S}$ receives a signal start from all honest parties and simulates the **Setup** as above.

Open : PartOpen Upon receiving a receiving a request from $\mathcal{F}_{\text{DP-AVSS}}$ to deliver the delayed output $s \in \mathbb{F}_p$ of "open", while in some epoch e_o, $\mathcal{S}_h$ waits that all simulated honest parties receive a $\widetilde{\mathsf{vps}}^{(e_o)} \in \mathrm{VPS}^{(e_o)}$. $\mathcal{S}_h$ then generates *false* opening shares of honest parties on $\widetilde{\mathsf{vps}}^{(e_o)}$ into s. [We could have specified private channels, in order to avoid simulation for a honest SR, but in any case this simulation is required for a corrupt SR]. This simulation is computed as follows:

(1) First $\mathcal{S}_h$ does an *inference of corrupt shares*: from the $t+1$ opening shares $\widetilde{d}_{i \in \mathcal{H}}$ of $\widetilde{\mathsf{vps}}^{(e_o)}$ held by simulated honest parties, it applies ShInfer to deduce the t corrupt opening shares $\widetilde{d}_{i \in \mathcal{I}}$ of $\widetilde{\mathsf{vps}}^{(e_o)}$.

(2) Second to simulate consistent opening shares from these $\widetilde{d}_{i \in \mathcal{I}}$ and the plaintext s, it applies ShSim to deduce the (unique) corresponding $t+1$ honest opening shares $d_{i \in \mathcal{H}^{e_o}}$ of s. Simulated honest parties $P_i^{(e_o)} \in \mathcal{H}^{(e_o)}$ then query $\mathcal{F}_{\text{NIZK}}$ for proofs of correct decryption for the $(d_i)_{i \in \mathcal{H}^{(e_o)}}$ under their keys $\mathsf{pk}_i^{(e_o)}$ without providing any witness.

Open : Output We distinguish two cases:

(1) If SR is honest, $\mathcal{S}$ sends the opening shares generated in the previous step to the simulated copy of SR through the simulated $\mathcal{F}_{\text{AUTH}}$ on behalf of honest parties. Once the simulated SR has received $t+1$ opening shares from $\mathcal{F}_{\text{AUTH}}$, $\mathcal{S}$ allows $\mathcal{F}_{\text{DP-AVSS}}$ to deliver to SR the output of "open".

(2) If SR is corrupt, $\mathcal{S}$ sends the simulated opening shares generated in the previous step to the simulated corrupt SR through $\mathcal{F}_{\text{AUTH}}$ on behalf of honest parties.

We now define $\mathcal{S}_c$. The simulations of the Setup, Refresh and Output are identical to the ones of $\mathcal{S}_h$, thus we only present the differences with $\mathcal{S}_h$.

$\mathcal{S}_c$

Setup Exactly as for $\mathcal{S}_h$.

Share $\mathcal{S}_c$ waits for the broadcast from SO, via the simulated $\mathcal{F}_{\text{BC}}$, to deliver an output to the simulated honest parties in $C^{(1)}$. If this output is not a PVSS then it sets $\widetilde{s} := 0$. Otherwise, $\mathcal{S}_c$ internally performs an opening of it [using the secret keys of the simulated honest parties in $C^{(1)}$ to generate $t+1$ opening shares] to obtain some $\widetilde{s} \in \mathbb{F}_p$. $\mathcal{S}_c$ sends $(\text{share}, \widetilde{s})$ to $\mathcal{F}_{\text{DP-AVSS}}$.

Refresh Exactly as for $\mathcal{S}_h$ [in the case $e = 1$: the precision "simulated honest SO" is replaced by "simulated corrupt SO".]

Open : PartOpen Simulated honest parties follow the protocol. [Namely: when some simulated honest $P_i^{(e_o)}$ receives "Open " in some epoch e_o, for all $\widetilde{\text{vps}}^{(e_o)}$ in its list $V_i^{(e_o)}$, it generates the opening shares of $\widetilde{\text{vps}}^{(e_o)}$ using its simulated secret key.]

Open : Output Exactly as for $\mathcal{S}_h$.

4.6.3 Indistinguishability with a Real Execution for a Corrupt SO*, i.e., Correctness of* DP-AVSS*.* Since the simulated honest parties follow the protocol, if SR is corrupt, then the view of $\mathcal{E}$ is identical to one of an actual execution. If SR is honest, then there are only two remaining variables based on which $\mathcal{E}$ can possibly distinguish the simulated execution from the real one: the *moment* when the *real dummy* SR outputs, and, in case it outputs, the *value* of this output. Since $\mathcal{S}_c$ sends "open" to $\mathcal{F}_{\text{DP-AVSS}}$ exactly when $t+1$ opening shares are delivered to the simulated SR, we have that the real dummy SR is delivered its output, by $\mathcal{F}_{\text{DP-AVSS}}$, exactly at the same moment. Thus it remains to prove that the value received by the real dummy SR is the same as the one opened in the simulated execution. Thus we are brought back to proving *correctness of* DP-AVSS, i.e., that the (unique, if any) explainable secret of the PVSS of SO, is equal to the opening of any VPS in the opening epoch e_o.

Lemma 9 (Correctness of Refresh)**.** *For any $e \geq 1$, any $\text{vps}^{(e+1)} \in \text{VPS}^{(e+1)}$ has exactly one threshold opening, furthermore there exists a $\text{vps}^{(e)} \in \text{VPS}^{(e)}$ such that both have the same threshold opening.*

Proof. $\text{vps}^{(e+1)} = (e+1, \boldsymbol{c'}, qvc)$ is signed by $t+1$ parties, thus of which at least one is honest. This implies that this party verified that: there exists (i) a $\text{vps}^{(e)} = (e, \boldsymbol{c}, qvc)$; (ii) $t+1$ resharing contributions from parties in a set $\mathcal{U}$ computed from this $\text{vps}^{(e)}$, i.e., all of the form $(e, \text{vps}^{(e)}, \boldsymbol{c^{(i)}}, \pi^{(i)}) \in \text{contribMsg}_i$; (iii) such that Reshare.Combine is correctly evaluated, i.e. $\boldsymbol{c}^{(e+1)} = \boxplus_{i\in\mathcal{U}}(\lambda_i^{\mathcal{U}} \boxdot \boldsymbol{c^{(i)}})$. By Proposition 4 (1), denoting, for each $i \in \mathcal{U}$: s_i the unique explainable decryptions of the i-th coordinate of $\text{vps}^{(e)}$ (of which knowledge is proven in the contribMsg_i), then we have that $\boldsymbol{c}^{(e+1)}$ has threshold opening into $\sum_{i\in\mathcal{U}}(\lambda_i^{\mathcal{U}} s_i)$, which is a threshold opening of $\text{vps}^{(e)}$. □

Lemma 10 (Correctness of DP-AVSS)**.** *For any epoch $e \geq 1$, the threshold opening of any $\text{vps}^{(e)} \in \text{VPS}^{(e)}$ is equal to the (unique) one of the $\text{PVSS}^{(1)}$ broadcast by* SO.

Proof. By Lemma 9, the unique threshold opening of any $\text{vps}^{(e)} \in \text{VPS}^{(e)}$ is equal to the one of some $\text{vps}^{(e-1)} \in \text{VPS}^{(e-1)}$. We conclude by induction down to $e = 1$, where the only $\text{vps}^{(1)} \in \text{VPS}^{(1)}$ is the $\text{PVSS}^{(1)}$ broadcast by SO. Furthermore it has one single explainable opening, by perfect correctness of LHE. □

4.6.4 Proof of indistinguishability with a real execution for an honest SO*.* We go through a series of hybrid games that will be used to prove the indistinguishability of the real and ideal worlds. The output of each game is the output of the environment.

The Game $\text{REAL}_{\mathcal{A}}$. This is the actual execution of the protocol DP-AVSSwith environment $\mathcal{E}$ and adversary $\mathcal{A}$ (and ideal functionalities ($\mathcal{F}_{CA}$, $\mathcal{F}_{NIZK}$)). We make the change that $\mathcal{F}_{\text{NIZK}}$ does not check validity of witnesses (if any) received from honest parties nor from SO. This does not change its outputs, since honest participants always provide correct witnesses when querying $\mathcal{F}_{\text{NIZK}}$ in the actual protocol.

The Game $\text{Hyb}^{\text{ShSim}}$ *.* This game differs from the *REAL* protocol, in that the opening shares of every $\text{vps}^{(e_o)} \in \text{VPS}^{(e_o)}$, of honest parties in the opening committee e_o, which are sent to SR, are now simulated as $\text{ShSim}(s, (s_i)_{i\in\mathcal{I}^{(e_o)}})$, of which the arguments are computed as follows. s is the input of SO. The opening shares of corrupt parties: $(s_i)_{i\in\mathcal{I}^{(e_o)}}$, are obtained by tracing back the $t+1$ contributions $(\text{pvr}_i \in \text{PVR})_{i\in\mathcal{U}}$ originating from $C^{(e_o-1)}$, from which $\text{vps}^{(e_o)}$ is formed. We compute their plaintext resharings shares $(s_i^{(i)})_{i\in\mathcal{U}}$ as follows, then apply the Lagrange reconstruction formula (equation (1)) on their coordinates in $\mathcal{I}^{(e_o)}$. The ones $(s_i^{(i)})_{i\in\mathcal{U}\cap\mathcal{H}^{(e_o-1)}}$ from honest parties are those generated by them. The ones: $(s_i^{(i)})_{i\in\mathcal{U}\cap\mathcal{I}^{(e_o-1)}}$ from corrupt parties, are those given by them to $\mathcal{F}_{\text{NIZK}}$, when generating their $(\text{pvr}_i)_{i\in\mathcal{U}\cap\mathcal{I}^{(e_o-1)}}$.

Claim 10.1. $\text{REAL}_{\mathcal{A}} \equiv \text{Hyb}^{\text{ShSim}}$

Proof. By correctness of the protocol (Lemma 10), s is equal to the (unique) explainable threshold opening of any $\text{vps}^{(e_o)}$. By perfect correctness of LHE, the $(s_j)_{j\in\mathcal{I}^{(e_o)}}$ are the unique explainable decryptions of the coordinates $j \in \mathcal{I}^{(e_o)}$ of the $\text{vps}^{(e_o)}$. The conclusion follows from Proposition 4 (2). □

The Game $\text{Hyb}^{\text{0Refresh}}$*.* During the Refresh between $C^{(e)}$ and $C^{(e+1)}$, the only place where the secret keys of honest parties of $C^{(e)}$ are used is when they decrypt their coordinate of a $\text{vps}^{(e)}$ into their plaintext share. We make the modification that they do not decrypt but instead set to 0 (or any arbitrary value) this plaintext coordinate. Notice that $\mathcal{F}_{\text{NIZK}}$ still issues proofs of correct resharing, since it does not check any witness from honest parties.

Claim 10.2. $\text{Hyb}^{\text{ShSim}} \equiv \text{Hyb}^{\text{0Refresh}}$

Proof. We make a backward induction with a cascade a subgames $\text{Hyb}^{\text{0Refresh}}[e, i]$ for each $e \in \{1, \ldots, e_o-1\}$ and $i \in [0, \ldots, n]$, in which the modification has been made from committees $e+1$ up to e_o, and for honest parties up to i (so no modification is done yet in $C^{(e)}$ if $i = 0$). In particular for any index $[e, i]$ in this cascade, we have the invariant that the decryption keys of parties in later committees $C^{(\geq e+1)}$ are not used anymore. This invariant holds for

the starting point of the induction, i.e., $e := e_o - 1$, since parties in $C^{(e_o)}$ do not use their decryption keys since $\mathsf{Hyb}^{\mathsf{ShSim}}$. We are thus brought back to showing indistinguishability of $\mathsf{Hyb}^{\mathsf{0Refresh}}[e, i]$ with $\mathsf{Hyb}^{\mathsf{0Refresh}}[e, i+1]$ when $P_{i+1}^{(e)}$ is honest. The difference between the two is that, in the former, the PVR is generated by $P_{i+1}^{(e)}$ with plaintext equal to its actual decrypted share, while in the latter, it is a PVR of 0. Since the PVR's are encrypted under the keys of parties of $C^{(e+1)}$, of which the secret keys are not used anymore, we conclude by IND-CPA of PVSS (Prop 5). [To further set ideas, this latter conclusion would not change even if $\mathcal{E}$ had been given the secret key of $P_{i+1}^{(e)}$]. □

The Game $\mathsf{Hyb}^{\mathsf{0Share}}$. This game is just like an execution of $\mathsf{Hyb}^{\mathsf{ShSim}}$ except in the Share stage, where SO plays the protocol as if it had input 0, even though it still sends s to $\mathcal{F}_{\text{DP-AVSS}}$. Notice that $\mathcal{F}_{\text{NIZK}}$ still issues proofs of correct sharing, since it does not check any witness from SO.

Claim 10.3. $\mathsf{Hyb}^{\mathsf{0Refresh}} \equiv \mathsf{Hyb}^{\mathsf{0Share}}$

Proof. Since the private keys of honest parties of $C^{(1)}$ are not used anymore since $\mathsf{Hyb}^{\mathsf{0Refresh}}$, we can apply IND-CPA (Prop 5) to the PVSS of SO, which is encrypted under their public keys. □

The Game $\mathsf{Hyb}^{\mathsf{DecRefresh}}$. We make the reverse operation of $\mathsf{Hyb}^{\mathsf{0Refresh}}$: honest parties use again their secret keys and correctly compute the Refresh. The same induction, this time in forward order up to $e_o - 1$, gives:

Claim 10.4. $\mathsf{Hyb}^{\mathsf{0Refresh}} \equiv \mathsf{Hyb}^{\mathsf{DecRefresh}}$

Note that up to this point, in all the previous games since $\mathsf{Hyb}^{\mathsf{ShSim}}$, we still have that the corrupt opening shares, which are fed into ShSim, are obtained by extracting the NIZKs of plaintext resharing shares of the PVRs generated by corrupt parties of $C^{(e_o-1)}$.

The Game $\mathsf{Hyb}^{\mathsf{ShInfer}}_{\mathsf{Open}}$. We now change the method to infer the opening shares of corrupt parties fed into ShSim . We now use the secret keys of parties in $C^{(e_o)}$ to correctly compute their opening shares, but these are not the ones that we send to SR (otherwise this would lead to an opening of 0, instead of s). Instead, we use them as input of ShInfer to infer the opening shares of corrupt parties.

Claim 10.5. $\mathsf{Hyb}^{\mathsf{DecRefresh}} \equiv \mathsf{Hyb}^{\mathsf{ShInfer}}_{\mathsf{Open}}$

Proof. Both methods to infer corrupt shares produce the same result, by Proposition 4 (3). □

Claim 10.6. $\mathsf{Hyb}^{\mathsf{ShInfer}}_{\mathsf{Open}} = \mathit{IDEAL}_{\mathcal{F}_{\text{DP-AVSS}},\mathcal{S}}$

Proof. This follows since the behavior of the ideal functionalities ($\mathcal{F}_{NIZK}, \mathcal{F}_{CA}, \mathcal{F}_{\text{AUTH}}, \mathcal{F}_{\text{BC}}$) and of the honest parties in $\mathsf{Hyb}^{\mathsf{ShInfer}}_{\mathsf{Open}}$, are identical to the simulation done by $\mathcal{S}$. □

5 OPTIMIZATIONS AND EXTENSIONS

5.1 Refresh in 2 Messages Instead of 3

Each $P_k^{(e+1)}$sends the reshare combineMsg directly to $C^{(e+2)}$. Parties in $C^{(e+2)}$ then verify the combined n-sized ciphertext and, if the verification passes: (i) generate a signature on it, (ii) and generate a PVR of their share, then send both (i) and (ii) to leaders in $C^{(e+2)}$. Each $P_k^{(e+2)}$ forms both: a $\mathsf{vps}^{(e+1)}$ out of $t+1$ signatures, and a combined n-sized vector of ciphertexts out of the $t+1$ PVRs.

5.2 Communication in $O(n^3)$ per Refresh

In each of the 3 steps of DP-AVSS.Refresh, honest parties send a worst-case total number of $O(n^4)$ bits. Indeed, each party sends messages to all n parties, each consisting in a concatenation of a worst-case number of n messages (each related to a $\mathsf{vps}^{(e)}$), each containing a n-sized vector of ciphertext. We can bring down the complexity of all three steps down to $O(n^3)$, as follows.

First, we pre-pend to each LB-refresh the following round-trip: [**pre-LB-refresh**] Each $P_i^{(e)} \in C^{(e)}$ chooses *one* of its $\mathsf{vps}^{(e)} \in V_i^{(e)}$ and sends it to the leader $P_k^{(e+1)} \in C^{(e+1)}$. Then $P_k^{(e+1)}$, upon receiving a $\mathsf{vps}^{(e)} \in \mathsf{VPS}^{(e)}$ for the first time, multicasts it to $C^{(e)}$. Each $P_i^{(e)} \in C^{(e)}$, upon receiving for the first time from $P_k^{(e+1)}$ some $\mathsf{vps}^{(e)} \in \mathsf{VPS}^{(e)}$, then executes LB-refresh$(C^{(e)}, C^{(e+1)}, k)$ by sending to $P_k^{(e+1)}$ a *single* resharing contribution, generated from this specific $\mathsf{vps}^{(e)}$, instead .

Then, we have that the combineMsgs are appended with $t+1$ PVRs, in order to enable public verification that the combined ciphertext is a linear combination taking into account contributions from at least $t+1$ distinct parties. We reduce their size down to $O(n)$, by the following easy adaptation of the technique of [GJM+21, p 13]. Each party appends, to every PVR that it generates (i) a n-sized vector of additively homomorphic commitment to the shares (ii) and to the polynomial evaluation at 0, i.e., the plaintext re-shared share, (iii) its signature on (ii). The leader then exhibits (i) the combined n-sized vectors of commitments, obtained by Lagrange homomorphic combination of the $t+1$ received vectors of commitments, along with (ii) (iii) the $t+1$ signed commitments to the evaluations at 0. Parties then verify that the Lagrange homomorphic combination of (ii) are equal to the commitment to the secret of the n-sized combined vector (i).

Finally, we can bring down the complexity of the last step by having each player send its signature, on a combined vector of ciphertexts, *only to* the leader $P_k^{(e+1)} \in C^{(e+1)}$ from which it received it. Indeed, $P_k^{(e+1)}$ will be able to form a $\mathsf{vps}^{(e+1)}$ from the signatures received. Thus, it will be able to hand-off this $\mathsf{vps}^{(e+1)}$ to all leaders in $C^{(e+2)}$ when performing pre-LB-refresh(or to all parties of $C^{(e+1)}$, in case this is the opening committee).

5.3 Adapting DP-AVSS to the Yoso Model

The computation model denoted "you only speak once" ("yoso") [GHK+21] was first introduced by [BGG+20] for proactive secret sharing. As in our model, they consider a possibly infinite sequence of committees, of which the members are uniquely determined

by their public key, nonwithstanding several of them could run on the same physical machine. Their corruption model is that the adversary corrupts parties at random [GHM+21], and thus that the choices of corruptions can be made by the simulator. This model is very close to our static corruption model, although our methods carry over adaptive corruptions.

The main requirement of "yoso" is that each party only speaks once then erases its memory. On the face of it, DP-AVSS imposes parties in $C^{(e+1)}$ to speak three times in a row, thus is not "yoso". But DP-AVSScan be easily compiled into "yoso" (although not the improvement in $O(n^3)$ of §5.2), since the roles in each step are *stateless*. In detail, we replace the entering committee by 3 committees: $C^{(e+1)}$ the "secret holders", $C^{(e+1),k}$ the "leaders" and $C^{(e+1),v}$ the "verifiers". At each Refresh, the exiting secret holders $C^{(e)}$ generate PVRs of the shares of the $\mathsf{vps}^{(e)}$s that they have, under the list of public keys of $C^{(e+1)}$, i.e. of the entering "secret holders". They send them to each leader of $C^{(e+1),k}$. Leaders combine then send their combineMsg to the verifiers. The latter sign and send their signatures to the secret holders $C^{(e+1)}$, enabling them to form lists of $\mathsf{vps}^{(e+1)} \in \mathsf{VPS}^{(e+1)}$.

5.4 Other Optimizations and Generalizations

NIZKs. UC NIZKs have size linear in the circuit proven. But actually, the simulator in our proof in Section 4.6.2, does not perform any straight-line extraction of witnesses. Then, for the the hybrid games, we need only simulation-sound extractability, not in straight-line. The consequence is that, e.g., Bulletproofs [BBB+18], which were recently shown to satisfy this property [GOP+22] under the algebraic group assumption, are thus applicable in DP-AVSSin place of $\mathcal{F}_{\mathrm{NIZK}}$. Bulletproofs have logarithmic size in the circuit proven, require no more setup than a random oracle, and their usage in PVSS is optimized in [GHL22].

Efficient aggregation of k signatures out of n signers. In our bare bulletin board PKI model: [BDN18] aggregate k signatures into one group element, appended with the list of the k signing public keys. [ACR21] prove (zero)-knowledge of k out of n signatures on m, in size $O(\log(n))$.

Changes of Sizes of Committees, of Threshold. To accommodate an entering committee of size $n' = 2t' + 1$, change accordingly the parameters of PVReshare. Also, setting $n > 2t+1$ enables not to wait for honest parties taking a very long time to respond, at the cost of lowering the privacy threshold: this trade-off is denoted "churn-tolerance" in [MZW+19].

REFERENCES

[ACR21] Thomas Attema, Ronald Cramer, and Matthieu Rambaud. "Compressed Σ-Protocols for Bilinear Group Arithmetic Circuits and Application to Logarithmic Transparent Threshold Signatures". In: *ASIACRYPT*. 2021.

[AGY95] Noga Alon, Zvi Galil, and Moti Yung. "Efficient dynamic-resharing "verifiable secret sharing" against mobile adversary". In: *European Symposium on Algorithms*. Springer. 1995, pp. 523–537.

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. "Bulletproofs: Short proofs for confidential transactions and more". In: *IEEE S&P*. 2018.

[BDN18] Dan Boneh, Manu Drijvers, and Gregory Neven. "Compact Multi-signatures for Smaller Blockchains". In: *ASIACRYPT*. 2018.

[BDOZ11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. "Semi-homomorphic Encryption and Multiparty Computation". In: *EUROCRYPT*. 2011.

[BGG+20] Fabrice Benhamouda, Craig Gentry, Sergey Gorbunov, Shai Halevi, Hugo Krawczyk, Chengyu Lin, Tal Rabin, and Leonid Reyzin. "Can a Public Blockchain Keep a Secret?" In: *TCC*. 2020.

[Bol03] Alexandra Boldyreva. "Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme". In: *PKC*. 2003.

[BS20] Dan Boneh and Victor Shoup. *A Graduate Course in Applied Cryptography*. Version 0.5 Jan 2020. 2020.

[Can01] Ran Canetti. "Universally composable security: A new paradigm for cryptographic protocols". In: *FOCS*. 2001.

[Can04] Ran Canetti. "Universally composable signature, certification, and authentication". In: *Proceedings. 17th IEEE Computer Security Foundations Workshop, 2004.* IEEE. 2004, pp. 219–233.

[CDN15] Ronald Cramer, Ivan Bjerre Damgård, and Jesper Buus Nielsen. *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, 2015.

[CKLS02] Christian Cachin, Klaus Kursawe, Anna Lysyanskaya, and Reto Strobl. "Asynchronous Verifiable Secret Sharing and Proactive Cryptosystems". In: *ACM CCS*. 2002.

[CL15] Guilhem Castagnos and Fabien Laguillaumie. "Linearly Homomorphic Encryption from DDH". In: *IACR Cryptol. ePrint Arch.* 2015 (2015), p. 47.

[CSV16] Ran Canetti, Daniel Shahaf, and Margarita Vald. "Universally composable authentication and key-exchange with global PKI". In: *PKC*. 2016.

[DLS88] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. "Consensus in the Presence of Partial Synchrony". In: *J. ACM* (1988).

[DM15] Yvo Desmedt and Kirill Morozov. "Parity Check based redistribution of secret shares". In: *ISIT*. 2015.

[DMR+21] Ivan Damgård, Bernardo Magri, Divya Ravi, Luisa Siniscalchi, and Sophia Yakoubov. "Broadcast-optimal two round MPC with an honest majority". In: *CRYPTO*. 2021.

[FGMY97] Yair Frankel, Peter Gemmell, Philip D MacKenzie, and Moti Yung. "Optimal-resilience proactive public-key cryptosystems". In: *FOCS*. 1997.

[GG06] V.H. Gupta and K. Gopinath. "An extended verifiable secret redistribution protocol for archival systems". In: *First International Conference on Availability, Reliability and Security (ARES'06)*. 2006.

[GHK+21] Craig Gentry, Shai Halevi, Hugo Krawczyk, Bernardo Magri, Jesper Buus Nielsen, Tal Rabin, and Sophia

Yakoubov. "YOSO: You Only Speak Once: Secure MPC with Stateless Ephemeral Roles". In: *CRYPTO*. 2021.

[GHL22] Craig Gentry, Shai Halevi, and Vadim Lyubashevsky. "Practical Non-interactive Publicly Verifiable Secret Sharing with Thousands of Parties". In: *EUROCRYPT*. 2022.

[GHM+21] Craig Gentry, Shai Halevi, Bernardo Magri, Jesper Buus Nielsen, and Sophia Yakoubov. "Random-Index PIR and Applications". In: *TCC*. 2021.

[GHY87] Zvi Galil, Stuart Haber, and Moti Yung. "Cryptographic computation: Secure fault-tolerant protocols and the public-key model". In: *CRYPTO*. 1987.

[GJM+21] Kobi Gurkan, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. "Aggregatable Distributed Key Generation". In: *EUROCRYPT*. 2021.

[GKM+22] Vipul Goyal, Abhiram Kothapalli, Elisaweta Masserova, Bryan Parno, and Yifan Song. "Storing and Retrieving Secrets on a Blockchain". In: 2022.

[GLS15] S. Dov Gordon, Feng-Hao Liu, and Elaine Shi. "Constant-Round MPC with Fairness and Guarantee of Output Delivery". In: *CRYPTO*. 2015.

[GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. "Proofs That Yield Nothing but Their Validity or All Languages in NP Have Zero-Knowledge Proof Systems". In: *J. ACM* (1991).

[GO07] Jens Groth and Rafail Ostrovsky. "Cryptography in the Multi-string Model". In: *CRYPTO*. 2007.

[GOP+22] Chaya Ganesh, Claudio Orlandi, Mahak Pancholi, Akira Takahashi, and Daniel Tschudi. "Fiat-Shamir Bulletproofs are Non-Malleable (in the Algebraic Group Model)". In: *EUROCRYPT* (2022).

[Gro21] Jens Groth. *Non-interactive distributed key generation and key resharing*. Iacr eprint. https://ia.cr/2021/339. 2021.

[HJKY95] Amir Herzberg, Stanislaw Jarecki, Hugo Krawczyk, and Moti Yung. "Proactive Secret Sharing Or: How to Cope With Perpetual Leakage". In: *CRYPTO*. 1995.

[KK09] Jonathan Katz and Chiu-Yuen Koo. "On Expected Constant-Round Protocols for Byzantine Agreement". In: (2009).

[MZW+19] Sai Krishna Deepak Maram, Fan Zhang, Lun Wang, Andrew Low, Yupeng Zhang, Ari Juels, and Dawn Song. "Churp: Dynamic-committee proactive secret sharing". In: *ACM CCS*. 2019.

[OY91] Rafail Ostrovsky and Moti Yung. "How to Withstand Mobile Virus Attacks (Extended Abstract)". In: *PODC*. 1991.

[SBKN21] Nibesh Shrestha, Adithya Bhat, Aniket Kate, and Kartik Nayak. "Synchronous Distributed Key Generation without Broadcasts". In: *IACR ePrint* (2021).

[Sho00] Victor Shoup. "Practical threshold signatures". In: *CRYPTO*. 2000.

[SLL10] David Schultz, Barbara Liskov, and Moses Liskov. "MPSS: Mobile Proactive Secret Sharing". In: *ACM Trans. Inf. Syst. Secur.* (2010).

[YXD22] Yunzhou Yan, Yu Xia, and Srinivas Devadas. *Shanrang: Fully Asynchronous Proactive Secret Sharing with Dynamic Committees*. Cryptology ePrint Archive, Report 2022/164. https://ia.cr/2022/164. 2022.

[ZSV05] Lidong Zhou, Fred B Schneider, and Robbert Van Renesse. "APSS: Proactive secret sharing in asynchronous systems". In: *ACM transactions on information and system security* (2005).

A COMPLEMENTS ON THE MODEL

We present in Fig. 4 the ideal functionality of a Bulletin Board of Public Keys introduced in §2.2.1 as $\mathcal{F}_{CA}$.

$\mathcal{F}_{CA}$

1. Upon receiving the first message (Register, v) from party P, send (Registered, P, v) to $\mathcal{A}$; upon receiving ok from $\mathcal{A}$, and if this is the first request from P , then record the pair (P, v).
2. Upon receiving a message (Retrieve, P) from party Q , send (Retrieve, P, Q) to $\mathcal{A}$, and wait for an ok from it. Then, if there is a recorded pair (P, v) output (Retrieve, P, v) to Q. Else output (Retrieve, P, $\perp$) to Q.

Figure 4: The certification authority functionality, $\mathcal{F}_{CA}$.

We present in Fig. 5 the ideal functionality of Non-interactive Zero-Knowledge proofs of knowledge, introduced in §2.2.3 as $\mathcal{F}_{NIZK}$.

$\mathcal{F}_{NIZK}$

The functionality is parameterized with an NP relation R of an NP language L and a prover P.

Proof: On input $(prove, \text{sid}, \text{ssid}, x, w)$ from P, ignore if $(x, w) \notin R$. Request $(\text{network}, proof, x)$ to $\mathcal{A}$. Upon receiving $(\text{activate}, \pi)$ from $\mathcal{A}$, store (x, π) and send $(proof, \text{sid}, \text{ssid}, \pi)$ to P.

Verification: On input $(verify, sid, ssid, x, \pi)$ from any party V, check whether (x, π) is stored. If not, send $(\text{network}, verify, x, \pi)$ to $\mathcal{A}$ and wait for an answer $(\text{activate}, witness, w)$. Upon receiving of the answer, check whether $(x, w) \in R$ and in that case, store (x, π). If (x, π) is stored, return (verification, sid, ssid, 1) to V, else return (verification, sid, ssid, 0).

Figure 5: Non-Interactive Zero-knowledge functionality

B IND-CPA OF PVSS

Property 11 (IND-CPA of PVSS)**.** Any PPT adversary $\mathcal{A}$ has negligible advantage in the following game with an oracle O, for any threshold $1 \leq t$. $\mathcal{A}$ selects t indices $I \subset [n]$, and gives to O a set of t public keys $(\text{pk}_i)_{i \in I}$ of its choice. Then O correctly generates $n - t$ public keys for the remaining indices $(\text{pk}_i)_{i \in [n] \setminus I}$ which it shows to $\mathcal{A}$. Then O tosses a bit b and subsequently responds to equests from $\mathcal{A}$ as follows. Either $(b = 1)$ then O returns to $\mathcal{A}$, upon each query $s \in \mathbb{F}_p$, a PVShare(s), or, if $(b = 0)$, a sample of the following distribution over C^n, which we denote $\mathcal{D}$:

- the coordinates in $\mathcal{I}$ are LHE encryptions of uniform independent values in $\mathbb{F}_p$;
- the remaining entries are LHE encryptions of 0.

Proof. We are going to bound the advantage of any adversary $\mathcal{A}$, by the maximum advantage of an adversary $\mathcal{A}_{\mathsf{LHE}}$ against oracle O_{LHE} of the following $(t+1)$-keys variant indistinguishability game for LHE. The latter is upper-bounded by $n-t$ times the advantage for one-message indistinguishability, see e.g. [BS20, Thm 5.1]. O_{LHE} samples $(n-t)$ LHE public keys $(\mathsf{pk}_h)_{h\in\mathcal{H}}$ which it gives to $\mathcal{A}_{\mathsf{LHE}}$. O_{LHE} tosses a bit $b \in \{0,1\}$ and subsequently has the following behavior: When $\mathcal{A}_{\mathsf{LHE}}$ submits $(n-t)$ chosen plaintexts $(s^h)_{h\in\mathcal{H}}$ to O_{LHE} either $(b=0)$ then O_{LHE} returns $(n-t)$ encryptions of 0: $(\mathsf{Enc}(\mathsf{pk}_h, 0))_{h\in\mathcal{H}}$, or: $(b=1)$ then O_E returns actual encryptions of the plaintexts $(\mathsf{Enc}(\mathsf{pk}_h, s^h))_{h\in\mathcal{H}}$.

The reduction is as follows. Upon receiving a set of keys $(\mathsf{pk}_h)_{h\in\mathcal{H}}$ from O_{LHE}, then $\mathcal{A}_E$ samples itself t key pairs $(\mathsf{sk}_i, \mathsf{pk}_i)_{i\in\mathcal{I}}$, initiates $\mathcal{A}$, reorganizes the indices so that the indices chosen by $\mathcal{A}$ correspond to $\mathcal{I}$, receives from $\mathcal{A}$ the t corrupt keys and gives to $\mathcal{A}$ the $n-t$ ones $(\mathsf{pk}_h)_{h\in\mathcal{H}}$ received from O_{LHE}.

Upon receiving one challenge plaintext s from $\mathcal{A}$, $\mathcal{A}_{\mathsf{LHE}}$ computes the first step of PVShare on it, namely: $(s_1, ..., s_n) \leftarrow \mathsf{Share}(s)$. It then sends the challenge $(n-t)$ plaintexts: $(s_h)_{h\in\mathcal{H}}$ to O_{LHE}. Upon receiving the response ciphertexts $(c_h)_{h\in\mathcal{H}}$ from O_{LHE}, it then computes the n-sized vector $\boldsymbol{c} \in C^n$ consisting of:

- The entries in $\mathcal{I}$ equal to correct encryptions $\left\{\mathsf{Enc}(\mathsf{pk}_i, s_i)_{i\in\mathcal{I}}\right\}$ that $\mathcal{A}_{\mathsf{LHE}}$ generates itself.
- The remaining entries are set to the $\{c_h\}_{h\in\mathcal{H}}$ received from O_E.

And sends $\boldsymbol{c}$ to $\mathcal{A}$ as response to its challenge. Upon answer a bit b from $\mathcal{A}_{\mathsf{LHE}}$, then $\mathcal{A}_E$ outputs the same bit b to O_E. Analysis:

- in the case where the ciphertexts $\{c_h\}_{h\in\mathcal{H}}$ are encryptions of the actual $n-t$ shares $\{s_h\}_{h\in\mathcal{H}}$, then $\mathcal{A}$ receives from $\mathcal{A}_{\mathsf{LHE}}$ a correctly generated PVSS of s;
- in the case where the ciphertexts $\{c_h\}_{h\in\mathcal{H}}$ are encryptions of 0, then, by uniform independence of the t plaintext shares $\left((s_i)_{i\in\mathcal{I}}\right.$, we have that what $\mathcal{A}$ receives from $\mathcal{A}_E$ is indistinguishable from a sample in the distribution $\mathcal{D}$;

Thus in both cases $b \in \{0,1\}$, $\mathcal{A}$ is faced with the same distribution as would have been generated by oracle O for the same b, thus the distinguishing advantage of $\mathcal{A}_{\mathsf{LHE}}$ is the same as the one of $\mathcal{A}$.

□