Proactive Secret Sharing over Asynchronous Channels under Honest Majority (with Ephemeral Roles): Refreshing Without a Consistent View on Shares

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

***Abstract*—We consider the task of proactive secret sharing (PSS). Expressed in the setting known as *dynamic membership*, the core functionality of a PSS protocol is to enable a committee of n holders of secret-shares, dubbed as "parties" to safely hand-over new shares of the same secret to a new committee. We dub such a sub-protocol as a *Refresh*. We present a PSS protocol, named as "Yoso-Verifiable secret sharing" (Y-VSS), which is the first PSS under honest majority over asynchronous channels. Moreover, Y-VSS matches an even higher security level than dynamic membership, known as *ephemeral roles* or "Yoso". Namely, each party speaks only once then erases its memory.**

In more details: a Refresh in Y-VSS takes only 2 message delays at the *actual speed of the network*, which is a feature known as *responsiveness*, despite the corruption of a tight minority of t-out-of $n := 2t+1$ parties in each committee of share-holders. In each Refresh, each party multicasts one message of $O(n^2)$ bits, which simply means that it sends it to the next committee over public asynchronous channels, e.g., by gossiping. By comparison, all existing protocols for Refresh under honest majority used a terminating broadcast. But this functionality requires consensus protocols under synchrony, incurring substantial latency and communication, and insecurity when synchrony fails. Alternatively, previous works in the "Yoso" model instantiated it from a public ledger, which incurred even more trust assumptions, latency and cost. Our technical contribution is to bypass the paradigm of all previous works, which required parties to reach consensus on a common set of shares, since consensus under honest majority is impossible under asynchrony. Even without providing a common view on a unique set of shares, Y-VSS can be tweaked into allowing the opening of linear combinations (or multi-exponentiations) of several secrets.

We demonstrate efficiency of Y-VSS with an implementation which requires only Elgamal encryption, standard DDH-based proofs of knowledge, and a bare bulletin board of public keys. Of independent interest, we provide the first formalization (and proof) of dynamic asynchronous verifiable secret sharing in the universal composability framework.

Changelog wrt. version 1 of 2022-05-23: Implementation and rewriting. The sketched optimization in $O(n^3)$ bit complexity was removed (a simpler one will appear in a follow-up).

1. Introduction

The goal of threshold cryptography is to process information that should remain secret, and timely deliver a correct result, despite an adversary corrupting any minority of participants. Flagship use-cases are the distributed generation and storage of secret keys [44], either for the purpose of distributed signing of transactions, or for secure storage [10]. The baseline technique is known as secret sharing. It enables any entity, denoted as a *dealer*, $\mathcal{D}$, to distribute shares of a secret, to n parties, while ensuring (i) that an adversary controlling 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, [58] introduced the notion of *proactive security*. The lifetime of the system is divided into time periods denoted *epochs*, and the adversary is able to corrupt at most t parties per epoch. We consider the most general setting, which is known as *dynamic*. 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 hosted, 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. A protocol denoted as a *dynamic proactive secret sharing scheme (PSS)* is one such that:

- it enables the dealer to share its secret to the committee of the first epoch, denoted as $\mathcal{C}^{(1)}$;
- it maintains the correctness and liveness invariants that in each epoch e, the current committee $\mathcal{C}^{(e)}$ holds shares of this same secret;
- to this end, there is a sub-protocol, denoted Refresh, which enables the committee $\mathcal{C}^{(e)}$ of some epoch e, denoted as *exiting*, to provide new shares of the same secret to the next committee $\mathcal{C}^{(e+1)}$, denoted as *entering*. "New shares" are also denoted as *proactivized* or *refreshed* shares. Refresh 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. In all existing PSS protocols, a Refresh requires some form of

Byzantine consensus among the entering committee $\mathcal{C}'$, very roughly, to reach consensus on a set of new secret shares obtained from the exiting committee $\mathcal{C}$. More particularly, all existing Refreshes under honest majority [10, 56, 43, 17, 44, 39] require a protocol for consistent broadcast with, at least, *eventual termination even if the sender is corrupt*, such as in [36]. We dub this primitive as *terminating broadcast*, or BC for short. Implementing BC beyond $t < n/3$ corruptions is impossible ([32]) without the extra assumption, known as *synchrony*, that communication channels deliver messages within a fixed public delay Δ. If one message arrives after Δ, then consistency or termination of the BC is not guaranteed, which in turn ruins the security or liveness of the PSS: see Section 7.5. This is why practical implementations of BC require many rounds and communications of fixed delay Δ equal to a conservative estimate of the worst-case delivery delay (including the gaps between the local clocks of parties). For instance, these issues have very concrete impacts on the security of implementations of the protocol [18], which is the most popular PSS in the context of key-refresh, known under the name "CMP" [35]. First, it is suggested by the authors, following [42], to downgrade BC into a primitive lighter to implement, denoted as "echo". In turn, the whole protocol can non-unanimously abort as soon as one party is corrupt. This suggestion was followed in a recent industrial implementation [26, 2]. Second, in the final version of their PSS [17, Figure 6], they replaced the BC of the first step of [18, Figure 6] with the following cheaper alternative: BC only the hash, then send the actual content by point to point. This replacement has the effect that, when generalizing their PSS to lower thresholds t ([18, p. 1.2.7]), then we do not have anymore the nice guarantee of termination as soon as $t+1$ parties behave honestly. Moving to systems of larger scale, where the communication complexity of BC is untractable (see below Table 6), then state of the art PSS [10, 56, 44, 43] suggest to instantiate BC from a public ledger, which incurs substantial cost and trust assumptions.

Asynchronous protocols, by contrast, are secure without any assumption on the network, and do not require BC. Better, they run at *the actual speed of the network*, which is a property known as *responsiveness* [59]. The challenge in constructing them is that it is impossible for a party P_i expecting a message from another party P_j, to tell apart if the message from P_j did not yet arrived, or if P_j is corrupt and did not send anything.

Additional challenge: imposing each party to speak only once. We furthermore aim at matching a security requirement even higher than dynamic committees. It was popularized by [24, 10, 40, 47] under the name "stateless ephemeral roles", or "Yoso". It imposes that every party sends only one batch of messages in the protocol, then immediately erases all its memory and quits. Hence, when combined with mechanisms which somehow hide the link between a party and its public key, until the moment when it speaks [56, 40, 38, 21, 14], then this prevents adaptive corruptions. To compensate this requirement, [40] assume the availability of intermediary committees for each elementary step of their protocol.

1.1 Main results. We show feasibility of proactive secret sharing tolerating both honest majority and asynchrony.

Theorem 1. *Consider committees of $n = 2t+1$ parties, in each of which t are maliciously corrupt. Consider a fully asynchronous communication network, with a bare bulletin board of public keys. There exists a proactive secret sharing scheme, denoted as* Y-VSS*, that securely implements verifiable secret sharing (VSS) in the UC sense, and such that each* Refresh *completes within 2δ, where δ is the actual network delivery delay of a message, plus the delay of the publication of their keys by the entering committee.*

Y-VSS moreover enjoys the following:

Ephemeral roles A Refresh between an exiting committee $\mathcal{C}$ and an entering committee $\mathcal{C}'$ (both of n parties), requires no more than the following communications: each party of $\mathcal{C}$ multicasts one message of $O(n^2)$ bits to an intermediary committee of $t+1$ parties, denoted $\mathcal{C}'_{\text{collec}}$ and dubbed as the "collectors". "Multicasts" simply means to send the same message to all parties of $\mathcal{C}'_{\text{collec}}$ over public asynchronous channels, e.g., by gossiping. Finally, each collector K'_k in $\mathcal{C}'_{\text{collec}}$ responsively multicasts one message of $O(n^2)$ bits to $\mathcal{C}'$. Responsively means that a collector multicasts no later than upon receiving messages from all honest parties in $\mathcal{C}$.

Verifiability Provided the additional specification that $\mathcal{D}$ uses a BC to send its (unique) message, thenY-VSS has *verifiability* [25, 6], in the strongest sense of [65], namely: the content of the broadcast of $\mathcal{D}$ *commits* $\mathcal{D}$ to exactly one value s, such that s is the only value that the learner $\mathcal{L}$ can possibly output.

Asynchronous UC security. More generally, we specify in Section 2 then prove, for the first time, asynchronous proactive verifiable secret (re)sharing in the universal composability (UC) framework of [15]. "Asynchronous" means that we do not resort on any functionality which would guarantee some form of synchrony, be it explicit ([54, p. 3.2]) or implicit ([28, §4],[43]).

Minimal trust on intermediaries. Y-VSS remains UC secure even if all collectors are corrupt. Furthermore if every Refresh is launched 2δ after the previous Refresh was launched, and as soon as there is *one* honest party in each committee of collectors, then the protocol terminates. Namely, parties are able to open a secret to any entitled learner (denoted as $\mathcal{L}$): we dub this property as *liveness*.

Linear combinations. Since, in Y-VSS, parties are not instructed to reach consensus on one consistent set of shares, it is not clear yet how they can open the linear combination of several secrets. We sketch in Section 7.4 how this can be achieved, in a slightly non-black-box way. This extends to secrets from multiple dealers, and various algebraic structures. Examples are given in Sections 7 and 7.4, such as linear combinations with polynomial coefficients, and also *multi-exponentiations*, which enables threshold BLS signing.

Simulatability of PVSS, without straight-line extraction. A by-product of our UC proof is that it positively answers in Section 5.1, supported by Appendix B.4, the question raised by Shrestha-Bhat-Kate-Nayak in [64, p5], asking if publicly verifiable secret sharing (PVSS) is simulatable. Of independent interest, we show that, in a strengthening of the model where all keys of honest parties are always published in time (as in [10, 40]), then UC security furthermore holds *without requiring straight-line extractability of witnesses from NIZKs*.

1.2 Baseline: re-sharing of encrypted shares. Let us describe the baseline technique of a Refresh between an exiting committee $\mathcal{C}:=\mathcal{C}^{(e)}$ and an entering committee $\mathcal{C}':=\mathcal{C}^{(e+1)}$. Let us assume that all $t+1$ honest parties in $\mathcal{C}$ have in common a vector of n ciphertexts of Shamir shares of the secret, denoted as $c_{[n]}:=(c_i)_{i\in[n]}$. Informally, this means that the decryption of each c_i under the secret key of the i-th party $P_i\in\mathcal{C}$, is equal to some s_i, such that there exists a polynomial f of degree t, such that $s_i=f(i)$ and $s=f(0)$. In particular, for any $t+1$-sized subset $\mathcal{U}\subset[n]$, there exists public coefficients $(\lambda_i^{\mathcal{U}})_{i\in\mathcal{U}}$, denoted as Lagrange coefficients, such that s can be reconstructed as $s=\sum_{i\in\mathcal{U}}\lambda_i^{\mathcal{U}}s_i$. This will be precised in Definition 3. This starting point is depicted at the bottom of Figure 1. The goal of Refresh is that parties in $\mathcal{C}'$ obtain a vector of ciphertexts $c'_{[n]}$ of new shares of the same secret. It consists of two steps.

Resharing First, each player $P_i\in\mathcal{C}$ decrypts its ciphertext share c_i into s_i, then generates a Shamir secret-sharing of s_i. Concretely, it samples a random polynomial $f_i(X)$ of degree t such that $f_i(0)=s_i$, then for each $j\in[n]=[1,\ldots,n]$: sets $s_{i\to j}:=f_i(j)$. Those shares $(s_{i\to j})_{j\in[n]}$ of s_i are dubbed as *sub-shares*, or also as *re-sharing shares*. Then P_i encrypts them under the public keys of the entering committee $\mathcal{C}'$: $c_{i\to j}:=\mathsf{Enc}_{\mathsf{pk}'_j}(s_{i\to j})$, where pk'_j is the public key of the j-th member of $\mathcal{C}'$. Finally, it appends to the vector $c_{i\to[n]}:=(c_j)_{j\in[n]}$ a non-interactive ZK argument of knowledge (NIZK AoK), denoted π_{pvR}, proving that $c_{i\to[n]}$ is indeed an encrypted resharing of a decryption of c_i. We dub such a pair $(c_{i\to[n]},\pi_{\mathsf{pvR}})$ as a *publicly verifiable resharing* of c_i, shortened as pvR.

Combination Consider any $t+1$ pvRs: $(c_{i\to[n]},\pi_{\mathsf{pvR}})_{i\in\mathcal{U}}$ ≪AU: il faut bien la preuve?≫, which are generated out of the same $c_{[n]}$, where $\mathcal{U}\subset[n]$ is some $t+1$-sized subset. We have that the Lagrange linear combination of their plaintext coordinates: $(s'_i)_{j\in\mathcal{U}}=(\sum_{i\in\mathcal{U}}\lambda_i^{\mathcal{U}}(c_{i\to j}))_{j\in\mathcal{U}}$ is a vector of (new) Shamir shares of the same secret s. This is because, as depicted in Figure 1, the $(s'_i)_{j\in\mathcal{U}}$ are, by construction, evaluations of the new polynomial $f':=\sum_{i\in\mathcal{U}}\lambda_i^{\mathcal{U}}f_i(X)$, whose evaluation at zero is $\sum_{i\in\mathcal{U}}\lambda_i^{\mathcal{U}}s_i=s$. We now add a new ingredient: we make the extra assumption that the public key encryption scheme supports a certain number of homomorphic linear combinations. Hence, any entity, possibly external, can compute homomorphically all-at-once the Lagrange linear combination of the $t+1$ pvRs and obtain a vector of ciphertexts of the new shares $(s'_i)_{j\in\mathcal{U}}$:

(1) $c'_{[n]}:=\mathscr{L}\text{-combine}^{\mathcal{U}}_{\mathsf{PKE}}\big((c_{i\to[n]})_{i\in\mathcal{U}}\big):=\underset{i\in\mathcal{U}}{\boxplus}\lambda_i^{\mathcal{U}}\boxdot c_{i\to[n]}$

The problem is that it is impossible to implement Byzantine broadcast and consensus beyond $t\geq n/3$ corruptions under asynchrony. Hence, it is hopeless that parties reach Byzantine consensus on a common subset of $t+1$ pvRs. In Section 1.3 we sketch our new computation model, which overcomes this. In Appendix E we put the baseline technique in the context of previous works.

1.3 Simpler variant of Y-VSS. We illustrate our techniques on a *simpler variant of* Y-VSS, which takes $\mathbf{3\delta}$ *instead of* $\mathbf{2\delta}$, and where parties speak multiple times, hence which is *not* "yoso".

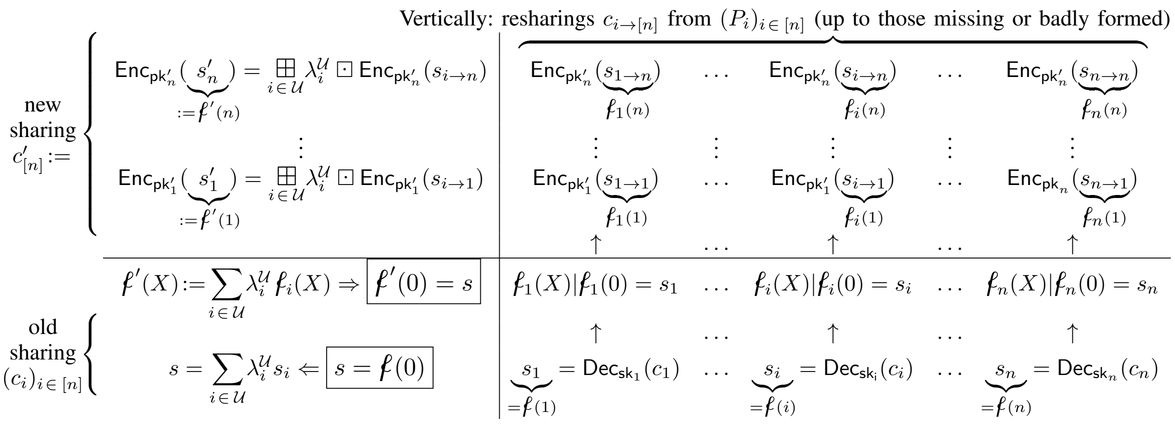
Sharing. The dealer of the secret, denoted $\mathcal{D}$, broadcasts an encrypted sharing $c_{[n]}$ of its secret.

In a model where $\mathcal{D}$ is always honest, then this can simply be implemented by sending (publicly) $c_{[n]}$ to all. Otherwise, an actual BC is necessary here, since VSS is strictly stronger than BC. We now describe a Refresh between an exiting committee $\mathcal{C}$ and an entering committee $\mathcal{C}'$, as depicted on Figure 2.

Inputs and Outputs. Each party in $\mathcal{C}$ has a local *list* containing at most $t+1$ vector of ciphertext shares. A notable friendly case is the first committee, in which all lists consist of one unique vector, which is the one broadcast by $\mathcal{D}$. To be valid and included in a local list, any such vector $c_{[n]}$ must furthermore come appended with $t+1$ signatures issued by $\mathcal{C}$. We call these signatures as a *quorum verification certificate*, denoted as qvc. Existence of a qvc guarantees that at least one honest signer, in $\mathcal{C}$, validated that $c_{[n]}$ had been correctly formed, as we are going to detail. We call such a pair $(c_{[n]},\mathsf{qvc})$ as a *verified proactivized sharing* (of s), denoted as a VPS.

- The *correctness* invariant maintained is that all VPS ever formed are vectors of ciphertext shares of the same secret as in the broadcast of $\mathcal{D}$.
- The *liveness* invariant maintained is that, if Refreshs are launched at least 3δ one after the other, then all the lists of honest parties in $\mathcal{C}$ contain at least one VPS in common. ***Hence, we depart from all previous works [10, 44], which required parties in $\mathcal{C}$ to start a* Refresh *with one common unique system of shares.*** Assuming that both correctness and liveness invariants hold for parties in $\mathcal{C}$, the goal of a Refresh is to achieve, at the end, the same invariant for parties in $\mathcal{C}'$.

Publicly encrypted re-sharing. We assume existence of an intermediary committee denoted as $\mathcal{C}'_{\mathrm{collec}}$ and called as the *collectors*. The mission of each collector is to deliver to all parties in $\mathcal{C}'$ the same output as in the baseline, namely: a batch of $t+1$ publicly verifiable resharings, originating from the same vector of shares. For every VPS: $\mathsf{vps}=(c_{[n]},\mathsf{qvc})$ in its local list, each party of $\mathcal{C}$ multicasts to $\mathcal{C}'_{\mathrm{collec}}$: a publicly verifiable resharing $\mathsf{pvr}_i=(c_{i\to[n]},\pi_{\mathsf{pvR},i})$ of its encrypted share c_i, as in the baseline, *appended with* vps.



Vertically: resharings $c_{i\to[n]}$ from $(P_i)_{i\in[n]}$ (up to those missing or badly formed)
new sharing $c'_{[n]} :=$
$\mathsf{Enc}_{\mathsf{pk}'_n}(\underbrace{s'_n}_{:=f'(n)}) = \boxplus_{i\in\mathcal{U}} \lambda_i^{\mathcal{U}} \boxdot \mathsf{Enc}_{\mathsf{pk}'_n}(s_{i\to n})$
$\mathsf{Enc}_{\mathsf{pk}'_1}(\underbrace{s'_1}_{:=f'(1)}) = \boxplus_{i\in\mathcal{U}} \lambda_i^{\mathcal{U}} \boxdot \mathsf{Enc}_{\mathsf{pk}'_1}(s_{i\to 1})$
$\mathsf{Enc}_{\mathsf{pk}'_n}(\underbrace{s_{1\to n}}_{f_1(n)}) \dots \mathsf{Enc}_{\mathsf{pk}'_n}(\underbrace{s_{i\to n}}_{f_i(n)}) \dots \mathsf{Enc}_{\mathsf{pk}'_n}(\underbrace{s_{n\to n}}_{f_n(n)})$
$\mathsf{Enc}_{\mathsf{pk}'_1}(\underbrace{s_{1\to 1}}_{f_1(1)}) \dots \mathsf{Enc}_{\mathsf{pk}'_1}(\underbrace{s_{i\to 1}}_{f_i(1)}) \dots \mathsf{Enc}_{\mathsf{pk}_n}(\underbrace{s_{n\to 1}}_{f_n(1)})$
$f'(X) := \sum_{i\in\mathcal{U}} \lambda_i^{\mathcal{U}} f_i(X) \Rightarrow \boxed{f'(0) = s}$
$f_1(X) | f_1(0) = s_1 \dots f_i(X) | f_i(0) = s_i \dots f_n(X) | f_n(0) = s_n$
old sharing $(c_i)_{i\in[n]}$
$s = \sum_{i\in\mathcal{U}} \lambda_i^{\mathcal{U}} s_i \Leftarrow \boxed{s = f(0)}$
$\underbrace{s_1}_{=f(1)} = \mathsf{Dec}_{\mathsf{sk}_1}(c_1) \dots \underbrace{s_i}_{=f(i)} = \mathsf{Dec}_{\mathsf{sk}_i}(c_i) \dots \underbrace{s_n}_{=f(n)} = \mathsf{Dec}_{\mathsf{sk}_n}(c_n)$

Figure 1: Resharing from $\mathcal{C}$ to $\mathcal{C}'$ of one vector of encrypted shares $c_{[n]} := \big(\mathsf{Enc}_{\mathsf{pk}_i}(s_i := f(i))\big)_{i\in[n]}$ of some secret s, under the keys $(\mathsf{pk}_i)_{i\in[n]}$ of $\mathcal{C}$. Each $P_i \in \mathcal{U}$ holding $c_{[n]}$ must generate a (publicly verifiable) resharing of s_i in the form of a vector of shares $\big(\mathsf{Enc}_{\mathsf{pk}'_j}(s_{i\to j} := f_i(j))\big)_{j\in[n]}$ (displayed vertically) encrypted under the keys $(\mathsf{pk}'_j)_{j\in[n]}$ of $\mathcal{C}'$. From correct resharings issued by a $t+1$-sized sub-set $\mathcal{U} \subset \mathcal{C}$ of parties, anyone can compute homomorphically their Lagrange linear combination, coordinate-wise, to obtain the vector of encrypted new shares $c'_{[n]}$ displayed vertically on the left. The n vertical equalities $=$ on the left-upper-hand state imply that the plaintext coordinates of $c'_{[n]}$ are evaluations of the Lagrange linear combination of the resharing polynomials $(f_i)_{i\in\mathcal{U}}$. Hence, as concluded in the lower-hand left corner (which illustrates Appendix B.1), these plaintexts also form a sharing of s.

Batching of encrypted re-sharings. Each collector K'_k: waits until it receives $t+1$ (valid) messages of the previous form $\big(\mathsf{vps}, (\mathsf{pvr}_i)_{i\in\mathcal{U}}\big)$ from some $(t+1)$-sized subset $\mathcal{U} \subset [n]$, all containing the *same* vps. Then it multicasts to $\mathcal{C}'$ the batch: $\mathsf{pps} := \big(\mathsf{vps}, (\mathsf{pvr}_i)_{i\in\mathcal{U}}\big)$ to $\mathcal{C}'$. Any entity receiving such a batch, *even from a corrupt collector* K'_k, can be convinced from the NIZKs appended to the $(\mathsf{pvr}_i = (c_{i\to[n]}, \pi_{\mathsf{pvR},i}))_{i\in\mathcal{U}}$, that they indeed form a consistent set of correct resharings of this same vps.

Combination and validation. Upon receiving a PPS, any party P'_j in $\mathcal{C}'$ homomorphically applies the Lagrange linear combination, as in Equation (1), to deterministically obtain a single vector of ciphertext new shares $c'_{[n]}$. Hence, by the same reason as in the baseline, $c'_{[n]}$ is a vector of ciphertexts of shares of the same secret as the one of vps. Since $c'_{[n]}$ is fully determined from the batch sent by the leader, we will denote such a batch $\mathsf{pps} := \big(\mathsf{vps}, (\mathsf{pvr}_i)_{i\in\mathcal{U}}\big)$ as a *proven proactivized sharing*, denoted as a PPS. Finally, P'_j multicasts to $\mathcal{C}'$: $c'_{[n]}$, with its (validation) signature σ_j on it.

Output Upon collecting $t+1$ such signatures vouching for the same vector of ciphertexts $c'_{[n]}$, which we denoted as a *quorum verification certificate*: qvc', any party in $\mathcal{C}'$ obtains a VPS: $\mathsf{vps} = (c'_{[n]}, \mathsf{qvc}')$, as desired.

Since there is at least one honest party in the signers of the qvc, this proves that $c'_{[n]}$ was indeed correctly formed out of $t+1$ correct pvRs generated from some same VPS: vps. Hence, by correctness of the baseline method, $c'_{[n]}$ must be a vector of ciphertext shares of the same secret as vps. So by induction on the number of consecutive Refreshes, we have that for every VPS ever formed in the protocol, it is necessarily a vector of ciphertext shares of the same secret as the one initially broadcast by the dealer. This concludes the proof of correctness. Assuming that liveness holds up to $\mathcal{C}$, then it holds for $\mathcal{C}'$ as soon as one collector is honest. Indeed, this collector K'_k will ultimately receive a re-sharing, from each honest party in $\mathcal{C}$, generated from the VPS that all honest parties in $\mathcal{C}$ have in common in their list. So it will be able to multicast a PPS to all $\mathcal{C}'$, they will all receive it and all honest parties among them will multicast their validation signature on the vector of ciphertext shares $c'_{[n]}$ formed out of $\mathcal{C}'$. Hence, every member of $\mathcal{C}'$ will obtain, at least, this same $c'_{[n]}$ with $t+1$ signatures, which constitutes a VPS, within 3δ (along with possibly other VPSs).

Opening of the secret s to any learner $\mathcal{L}$. Parties in a committee send to $\mathcal{L}$ their verifiably decrypted share of all vps which they have.

By the liveness invariant, there exists at least one VPS: vps, of which the $\mathcal{L}$ will receive at least $t+1$ verifiably decrypted shares.

Let us wrap-up the novelties of the new construction. The main one is that it does not require nor imply a consensus on one VPS in common. Actually, the same malicious collector could send one different PPS to each of the $t+1$ honest parties in $\mathcal{C}'$. Nothing prevents each of these PPS to, subsequently, receive $t+1$ signatures (one from the honest receiver, t from corrupt members of $\mathcal{C}'$). So nothing prevents that every party of $\mathcal{C}'$ ends up with $t+1$ VPS at the end of the Refresh, of which t of them are not in common with any other party of $\mathcal{C}'$. Overall, Y-VSS is the first proactive secret sharing scheme in which parties do not know which consistent system of shares they have in common. The second novelty is the validation mechanism, involving the quorums of signatures. Without this mechanism, the parties would have to incorporate a whole PPS in their resharing messages, instead of the small vps. Worse, this extra-size would add up at every new Refresh: a PPS formed out of resharing messages, contains in particular the PPS out of which these resharing messages

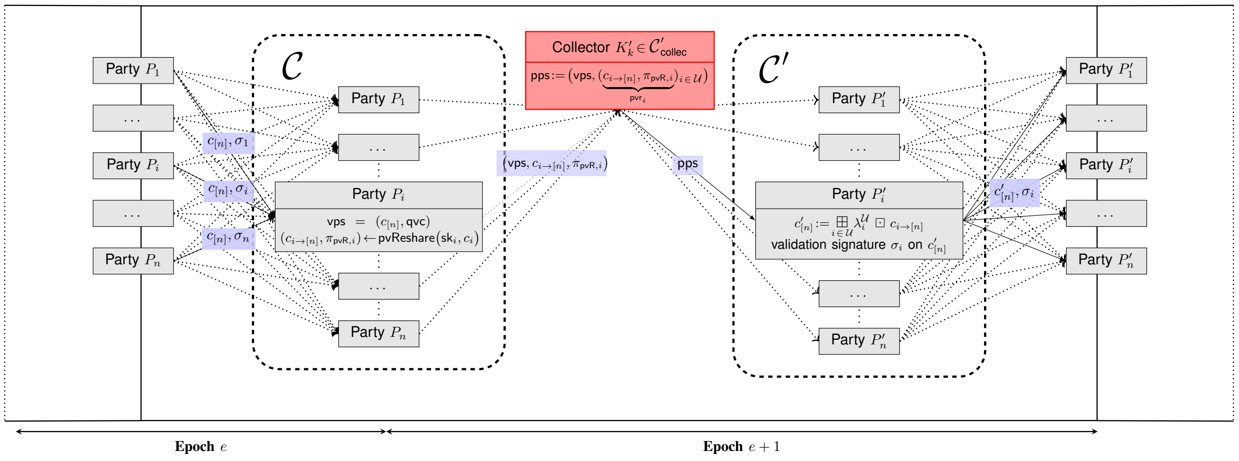

Party P_1
...
Party P_i
...
Party P_n
$c_{[n]}, \sigma_1$
$c_{[n]}, \sigma_i$
$c_{[n]}, \sigma_n$
$\mathcal{C}$
Party P_1
...
Party P_i
vps $=$ $(c_{[n]}, \mathsf{qvc})$
$(c_{i\to[n]}, \pi_{\mathsf{pvR},i}) \leftarrow \mathsf{pvReshare}(\mathsf{sk}_i, c_i)$
...
Party P_n
$(\mathsf{vps}, c_{i\to[n]}, \pi_{\mathsf{pvR},i})$
Collector $K'_k \in \mathcal{C}'_{\mathsf{collec}}$
$\mathsf{pps} := (\mathsf{vps}, \underbrace{(c_{i\to[n]}, \pi_{\mathsf{pvR},i})}_{\mathsf{pvr}_i}{}_{i\in\mathcal{U}})$
pps
$\mathcal{C}'$
Party P'_1
...
Party P'_i
$c'_{[n]} := \boxplus_{i\in\mathcal{U}} \lambda_i^{\mathcal{U}} \boxdot c_{i\to[n]}$
validation signature σ_i on $c'_{[n]}$
...
Party P'_n
$c'_{[n]}, \sigma_i$
Party P'_1
...
Party P'_i
...
Party P'_n
Epoch e
Epoch $e+1$

Figure 2: $\mathsf{Refresh}(\mathcal{C}, \mathcal{C}')$ in the simpler variant of Y-VSS in 3δ. Each party P_i of $\mathcal{C}$ has a list of verified proactivized sharings, consisting in tuples $\mathsf{vps} = (c_{[n]}, \mathsf{qvc})$ where $c_{[n]} = (c_i)_{i\in[n]}$ is a vector of ciphertexts, and qvc a set of $t+1$ signatures from $\mathcal{C}$ on $c_{[n]}$ received at the end of the previous Refresh. For each such vps in its list (**only one is depicted**), P_i computes a publicly verifiable resharing of its coordinate c_i: $(c_{i\to[n]}, \pi_{\mathsf{pvR},i}) \leftarrow \mathsf{pvReshare}(\mathsf{sk}_i, c_i)$ which it sends to the collectors (**only one is depicted**) appended with vps. Each collector waits until it collects a batch of such encrypted resharings, out of a subset $\mathcal{U} \subset [n]$ of $t+1$ parties, all verifiably generated out of the same vps, then multicasts the batch to $\mathcal{C}'$ in the form $\mathsf{pps} := (\mathsf{vps}, (\mathsf{pvr}_i)_{i\in\mathcal{U}})$ Every party $P'_j \in \mathcal{C}'$, for every (valid) pps which it receives (**only one is depicted**), forms $c'_{[n]}$ by Lagrange linear combination, and multicasts $c'_{[n]}$ with its validation signature σ_j, to $\mathcal{C}'$. Upon receiving such a $c'_{[n]}$ with $t+1$ signatures, a party in $\mathcal{C}'$ assembles them into a VPS, which it adds to its list.

were generated, etc. This means that at every new Refresh, the size of the messages would be augmented by $t+1$ vectors of ciphertexts, compared to the previous Refresh. Instead, our mechanism enables that a VPS can be safely re-used in future refreshes, it needs not containing any other justification data than the qvc.

1.4 Efficiency. In Section 6 we report on our implementation, using Elgamal encryption. In Section 7 we discuss various other instantiations of the encryption scheme and corresponding NIZKs of resharing, all from existing and implemented ingredients. In particular, the schemes of Paillier-$\mathrm{mod}\,p$ and Elgamal in-the-exponent are enabled thanks to a relaxed specification, which we make in Section 3.2, denoted as *perfect correctness after* one *homomorphic linear combination*.

2. Model and preliminaries

In Section 2.2 we give the security requirement which we aim at. Then in Sections 2.3 to 2.11 we specify the resources at hand: dynamic committees of parties, ideal functionalities for asynchronous communication, publication of keys etc. and the constraints in presence: corruptions, and a scheduler instructing to immediately speak-then-erase one's memory. In Appendix A.2 we give the full details of the formalization of dynamic proactive asynchronous secret sharing in the universal composability (UC) framework of [15]. In Appendix A.3 we make various comments on the model and on related ones.

2.1 General notations. The set of integers is denoted as $\mathbb{Z}$, the set of non-negative ones as $\mathbb{N}$, of which the positive ones as $\mathbb{N}^* = \{1, 2, \dots\}$. We consider integers $t \in \mathbb{N}$ and $n := 2t+1$. Let p denote any prime number larger than $n+1$, then $\mathbb{F}_p$ denotes the finite field $\mathbb{Z}/p\mathbb{Z}$. We will often we abuse notations by identifying $\mathbb{F}_p$ with the set of representatives $[0, \dots, p-1]$ in $\mathbb{Z}$. Recall that no secret sharing scheme is both private and robust beyond the threshold $n = 2t+1$. 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 with a subscript $[n]$, e.g., $c_{[n]}$, and their coordinates as c_i, or $c_{[n]}[i]$, for $i \in [n]$. The random inputs of algorithms are written last, at the right of the $|$. When algorithms are called without the random inputs, then this means that the random inputs are sampled uniformly in specified sets. Uniform sampling does not actually restrict the generality. We leave implicit the security parameter. $\mathbb{F}_p[X]_{\leq t}$ denotes the $(t+1)$-vector space of polynomials of degree at most t.

2.2 Dealer, learner, adversary, and security requirement. For simplicity we consider one PPT machine called as *dealer* and denoted as $\mathcal{D}$, which has an input $s \in \mathbb{F}_p$ denoted as its *secret*; and one PPT machine denoted *learner* $\mathcal{L}$, which may output some value in $\mathbb{F}_p$ at some point. We consider a PPT machine called as the *adversary* and denoted $\mathcal{A}$. We are aiming at a protocol which *securely implements, in the UC sense of [15]*, the ideal functionality of *verifiable secret sharing*, denoted as $\mathcal{F}_{\mathrm{VSS}}$ and formalized in Algorithm 3. Very informally, a protocol which UC-implements $\mathcal{F}_{\mathrm{VSS}}$, guarantees that, after $\mathcal{D}$ completed its task, there is one unique value $\widetilde{s}$ which can possibly be released to $\mathcal{L}$, and such that furthermore $\widetilde{s} = s$ if $\mathcal{D}$ is "honest" (see below). It also guarantees that $\mathcal{A}$ is leaked no information on s if $\mathcal{L}$ is "honest". See below Theorem 6 for more details, then Appendix A.2 for the rigorous general definition.

$\mathcal{F}_{\text{VSS}}$

Upon receiving ("share", s) from $\mathcal{D}$ for the first time: stores s, sends "stored" to $\mathcal{D}$ and $\mathcal{L}$.
Upon receiving "open" from $\mathcal{A}$: if some s is stored, send s to the $\mathcal{L}$.

Algorithm 3

2.3 Committees, epochs, malicious scheduler and static 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 polynomial time (PPT) machines $\mathcal{C}^{(e)} = (P_1^{(e)}, \ldots, P_n^{(e)})$, denoted as a *committee of shareholders*, or simply *committee*, for simplicity of fixed size $n = 2t+1$. For each e, we also consider a set of $t+1$ PPT machines $\mathcal{C}_{\text{collec}}^{(e)}$, denoted as a *committee of collectors*. Members of committees as denoted as *parties*. All the committees$\big(\mathcal{C}^{(e)}, \mathcal{C}_{\text{collec}}^{(e)}\big)_{e \in \mathbb{N}^*}$ are disjunct, nonwithstanding any two parties could possibly be hosted on the same physical computer. Parties have initially no internal state.

We consider an ideal functionality, denoted as the *(malicious) scheduler* $\mathcal{F}_{\text{Sch}}$ and which is actually fully controlled by $\mathcal{A}$. $\mathcal{F}_{\text{Sch}}$ can send a signal $\mathsf{start\,sig}$ to any party $P_i^{(e)}$ (for any $i \in [n]$ and $e \in \mathbb{N}^*$) at any moment. From this point, $P_i^{(e)}$ can take actions, we say that it is *alive*. On their side, $\mathcal{D}$ and $\mathcal{L}$ start the protocol alive. $\mathcal{F}_{\text{Sch}}$ can also send a signal, denoted as $\mathsf{share\,sig}$, to $\mathcal{D}$. Informally, it instructs $\mathcal{D}$ to share its secret to $\mathcal{C}^{(1)}$. Before $\mathcal{D}$ receives $\mathsf{share\,sig}$, $\mathcal{A}$ has the possibly to *(maliciously) corrupt* it, in the sense below. Likewise, $\mathcal{A}$ can maliciously corrupt any player $P_i^{(e)}$ before it receives $\mathsf{start\,sig}$, up to t parties per committee. $\mathcal{A}$ may also corrupt $\mathcal{L}$ at any point. Concretely, a corrupt participant ($\mathcal{D}$ or $\mathcal{L}$ or a party) is forever fully controlled by $\mathcal{A}$ and forwards to $\mathcal{A}$ all its incoming messages or signals. We denoted the subset of corrupt parties in each committee $\mathcal{C}^{(e)}$ by $\mathcal{I}^{(e)} \subset [n]$. The non-corrupt participants as denoted as *honest*. The honest parties are indexed as $\mathscr{H}^{(e)} = \mathcal{C}^{(e)} \backslash \mathcal{I}^{(e)}$.

2.4 Asynchrony formalized by ideal functionalities with *delayed output*. Following [15] 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 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, $\mathcal{F}$ outputs v to R.

2.5 Public authenticated asynchronous channels $\mathcal{F}_{\text{AT}}$ and multicast. All parties in some committee $\mathcal{C}^{(e)}$ are connected with all parties in $\mathcal{C}_{\text{collec}}^{(e+1)}$, themselves connected with the next committee $\mathcal{C}^{(e+1)}$, by public *authenticated message transmitting* with delayed output. It is formalized as the functionality $\mathcal{F}_{\text{AT}}$ (denoted by $\mathcal{F}_{\text{AUTH}}$ in [15]). It is parametrized by a sender S and a receiver R. On input $(\mathsf{input}, \mathsf{ssid}, v)$ from S: then $\mathcal{F}_{\text{AT}}^{S,R}$ provides R with public delayed output (ssid, v). To *multicast* a message to a committee simply means to send it over $\mathcal{F}_{\text{AT}}$ to all its members, e.g., by gossipping [27].

2.6 Private channels to the learner $\mathcal{F}_{\text{ST}}$. All parties are furthermore connected to $\mathcal{L}$ by *secure message transmitting*, formalized by the ideal functionality $\mathcal{F}_{\text{ST}}$ (as in [28]). It is like $\mathcal{F}_{\text{AT}}$, except that the *content* of the messages sent are not leaked anymore to $\mathcal{A}$. Compared to the "$\mathcal{F}_{\text{SMT}}$" in [15], we do not allow to adaptively corrupt the receiver $\mathcal{L}$ of $\mathcal{F}_{\text{ST}}$ and subsequently learn the content of the message. Indeed, corruptions in our model are static.

2.7 Asynchronous proactivity formalized as forced Refreshes. We now introduce a formalization of what is an *asynchronous proactive* secret sharing protocol, compared to any protocol which would UC-implement VSS. It is the first one to our knowledge, so may be of independent interest. We capture it by giving the power to the malicious scheduler $\mathcal{F}_{\text{Sch}}$, to send at any point a signal $\mathsf{refresh\,sig}$ to any shareholder P. In the model of ephemeral roles ("yoso" [40]) which we consider, this signal imposes P to send at most one batch of messages, to the next collector committee, then $\mathsf{shut\text{-}off}$ itself. This means that P erases all its memory and quits the protocol. Likewise, when some P receives $\mathsf{open\,sig}$, it must send at most one message, to $\mathcal{L}$ over $\mathcal{F}_{\text{ST}}$, then $\mathsf{shut\text{-}off}$ itself.

2.8 Bulletin board PKI: $\mathcal{F}_{\text{CA}}$. We consider the classical model of an entity, denoted as *certification authority* $\mathcal{F}_{\text{CA}}$ in [16]. Each party P_i in any committee $\mathcal{C}^{(e)}$ can give one public key *of its choice* to $\mathcal{F}_{\text{CA}}$. Then after $\mathcal{A}$ allows it, $\mathcal{F}_{\text{CA}}$ publishes the key. Hence, since $\mathcal{F}_{\text{CA}}$ *does not perform any check of knowledge of a secret key*, it as also known as "bulletin board" PKI or "bare" PKI [19]. In our simple model, $\mathcal{F}_{\text{CA}}$ also publishes the identity of the issuer of the key. But Y-VSS seamlessly carries over the model [10] where $\mathcal{F}_{\text{CA}}$ would simply publish that the key belongs to the "i-th role of $\mathcal{C}^{(e)}$ ".

2.9 Terminating reliable broadcast from $\mathcal{D}$ to $\mathcal{C}^{(1)}$. We allow a single call to the following ideal broadcast functionality $\mathcal{F}_{\text{BC}}^{\mathcal{D},\mathcal{C}^{(1)}}$, dubbed as $\mathcal{F}_{\text{BC}}$. If $\mathcal{D}$ is honest, $\mathcal{F}_{\text{BC}}$ waits to receive some value x from $\mathcal{D}$, then delay-outputs x to every party in $\mathcal{C}^{(1)}$. If $\mathcal{D}$ is corrupt, $\mathcal{F}_{\text{BC}}$ requests from $\mathcal{A}$ a value, then, upon receiving some y from $\mathcal{A}$, delay-outputs y to each party in $\mathcal{C}^{(1)}$. In a model where $\mathcal{D}$ would always be honest, then $\mathcal{F}_{\text{BC}}^{\mathcal{D},\mathcal{C}^{(1)}}$ can be implemented as a mere multicast.

2.10 NIZKs. We will use *non-interactive zero knowledge arguments of knowledge*, which we dub as *NIZK AoKs*, or simply *NIZKs*. For simplicity, we capture them by the ideal functionality $\mathcal{F}_{\text{NIZK}}$, defined in [45] and recalled in Appendix A.1.

2.11 Signature scheme. Parties have access to any standard digital signature scheme. We will sometimes loosely refer to the ability to "combine" $t+1$ signatures on the *same* message. The reader may simply parse this as the concatenation of the $t+1$ signatures. More efficiently, this could also, e.g., be understood in the sense of so-called multisignatures [61].

3. Publicly Verifiable Secret (Re-)Sharing

3.1 Overview and roadmap. In Section 3.3 we recall the textbook Shamir secret sharing. In Section 3.2 we provide toy specifications on the public key encryption scheme, under which parties will encrypt the shares. These toy specifications, which may be of independent interest, will be enough for the exposition of Y-VSS. In Section 3.4 we introduce the notion of a *vector of ciphertext shares*, which enables to formalize correctness and simulatability of the baseline of Section 1.2.

3.2 Toy specifications of the public key encryption. We consider a public key encryption scheme, denoted PKE, where, for simplicity, we consider the plaintext space to be $\mathbb{Z}$. The key generation function is denoted as $\mathsf{KeyGen}(\,|\,\rho_{\text{key}} \in \mathscr{R}_{\text{key}}) \rightarrow \mathscr{sK} \times \mathscr{pK}$, where $\mathscr{R}_{\text{key}}$ is the space of key randomness. The ciphertext space is denoted as $\mathscr{C}$, and the randomized encryption function as $\mathsf{Enc} : (\mathsf{pk} \in \mathscr{pK}, x \in \mathbb{Z}\,|\,\rho_{\text{enc}} \in \mathscr{R}_{\text{enc}}) \rightarrow \mathscr{C}$, where $\mathscr{R}_{\text{enc}}$ is the space of encryption randomness. By convention, any $\mathsf{pk} \notin \mathscr{pK}$ is denoted as $\perp$, and encryption under $\perp$ returns $\perp$. We require indistinguishability between encryptions of two chosen plaintexts (IND-CPA). We introduce the following new specification (further formalized in Definition 2).

Definition 2 (Perfect correctness $\bmod\, p$ after one homomorphic linear combination of size $t+1$). We consider functions denoted as the *decryption* $\bmod\, p$ $\mathsf{Dec} : (\mathsf{sk} \in \mathscr{sK}, c \in \mathscr{C}) \rightarrow (\mathbb{F}_p, \perp)$; the *homomorphic addition* $\boxplus : \mathscr{C} \times \mathscr{C} \rightarrow \mathscr{C}$ and *scalar multiplication* $\boxdot : \mathbb{F}_p \times \mathscr{C} \rightarrow \mathscr{C}$. Then for any key randomness ρ_{key}, $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(\,|\,\rho_{\text{key}})$; for any up to $t+1$ plaintexts $(x_i)_{i \in [t+1]} \in \mathbb{F}_p^{t+1}$, randomnesses $(\rho_{\text{enc},i})_{i \in [t+1]} \in \mathscr{R}_{\text{enc}}^{t+1}$ and coefficients $(\lambda_i)_{i \in [t+1]} \in \mathbb{F}_p^{t+1}$, we require:

$$(2)\quad \mathsf{Dec}\big(\mathsf{sk}, \underset{i \in [t+1]}{\boxplus} (\lambda_i \boxdot \mathsf{Enc}(x_i\,|\,\rho_{\text{enc},i})\big) = \sum_{i=1}^{t+1} \lambda_i x_i \bmod p\,.$$

Without loss of generality we consider $\boxplus$ and $\boxdot$ to be deterministic (when not, as, e.g., in [22], then a default randomness parameter can be specified in the protocol). Definition 2 may be of independent interest, since it relaxes the requirement of correctness $\bmod\, p$ after *unlimited* linearly homomorphic operations, e.g., as in the resharing scheme [21]. This enables to instantiate PKE from strictly more schemes, as the Paillier $\bmod\, p$, and also the el-Gamal-in-the-exponent, described in Section 7. The perfect correctness guarantee implies that *the decryptor P, is binded to one unique value for the decryption*, as long as the encryptors *proved knowledge of plaintexts no larger than p, with encryption noises $\rho_{enc,i}$ within specified bounds*, and that P proves knowledge of some key generation randomness ρ_{key}, *within specified bounds*, explaining its public key. This holds even if the encryptors and/or P did not use the prescribed distributions to sample their randomnesses. Noticeably, this "decryptor-binding" guarantee is also necessary in [40, 10, 39]. Although the last two do not require homomorphic additivity, their robustness hold only if ciphertexts of publicly verifiable re-sharings *commit* their receiver to its new share. By contrast, some related specifications of PKE do not imply this decryptor-binding guarantee: [9] (Semi-HE), and [66] (committing encryption). We illustrate in Appendix F.3 how the specified ranges can be enlarged, in order to allow slack in the NIZKs of range.

3.3 Shamir secret sharing over $\mathbb{F}_p$. Let us recall the algorithm of *Shamir secret sharing*, without verifiability:

$\mathsf{Share}(s \in \mathbb{F}_p)$: sample a random polynomial f of degree at most t in the subspace of those evaluating to s at 0, output the *vector of Shamir shares of* s: $s_{[n]} := (s_i)_{i \in [n]}$, where $s_i := f(i)\ \forall i \in [n]$.

We then denote f as the (unique) *sharing polynomial defining* $s_{[n]}$. We have the *t-privacy* property that, for a fixed s, any t-sized subset $\mathcal{I} \subset [n]$ of coordinates $(s_i)_{i \in \mathcal{I}}$ output by $\mathsf{Share}(s)$, vary uniformly at random in $\mathbb{F}_p^t$. We have the $t+1$-*reconstruction* property that, for any subset $\mathcal{U} \subset [n]$ of size $t+1$, denoting the *Lagrange coefficients associated* to $\mathcal{U}$ as $\lambda_i^{\mathcal{U}} := \prod_{j \in \mathcal{U} \setminus \{i\}} \frac{-j}{i-j}$, $\forall i \in \mathcal{U}$, then the following linear map, denoted as *Lagrange combination*:

$$(3)\qquad \mathscr{L}\text{-}\mathsf{combine}^{\mathcal{U}} : (s_i)_{i \in \mathcal{U}} \longrightarrow \sum_{i \in \mathcal{U}} \lambda_i^{\mathcal{U}} s_i$$

is such that $s = \mathscr{L}\text{-}\mathsf{combine}^{\mathcal{U}}((s_i)_{i \in \mathcal{U}})$.

3.4 Baseline method of encrypted re-sharing: correctness (and simulatability). In what follows we consider as a public parameter a list of public keys $(\mathsf{pk}_i)_{i \in [n]} \in (\mathscr{pK} \sqcup \perp)^n$. In order to make compact statements, we introduce some useful terminology. For $\mathsf{pk} \in \mathscr{pK}$, we say that $\widetilde{\mathsf{sk}} \in \mathscr{sK}$ is an *explainable secret key of* pk if there exists $\rho_{\text{key}} \in \mathscr{R}_{\text{key}}$ such that $(\widetilde{\mathsf{sk}}, \mathsf{pk}) = \mathsf{KeyGen}(|\rho_{\text{key}})$. We say that $\widetilde{x} \in \mathbb{F}_p$ is an *explainable decryption* of c under some $\mathsf{pk} \in \mathscr{pK}$, if there exists an explainable secret key $\widetilde{\mathsf{sk}}$ of pk such that $\mathsf{Dec}(\widetilde{\mathsf{sk}}, c) = \widetilde{x}$. A *proof of correct decryption of c into $x \in \mathbb{F}_p$ under* pk, shortened as π_{pvD} (which stands for *publicly verifiable decryption*), is a NIZK AoK of such a $\widetilde{\mathsf{sk}}$. In particular, π_{pvD} implies that x is an explainable decryption of c under pk.

Definition 3 (Ciphertext shares, opening shares and threshold opening)**.** We say that $c_{[n]} = (c_j)_{j \in [n]} \in \mathscr{C}^n$ is a *vector of ciphertext shares* iff:

unicity for every $j \in [n]$, c_j has at most one explainable decryption s_j under pk_j. When such s_j exists, we denote it as an *opening share*;

polynomial There exists a polynomial $f \in \mathbb{F}_p[X]_{\leq t}$ such that, for every opening share s_j, $s_j = f(j)$.

We now introduce terminology:

- *if* there exists at least $t+1$ coordinates $j \in [n]$ such that c_j has an explainable decryption, then f is uniquely determined. We denote it as *the sharing polynomial.* We then say that $c_{[n]}$ is a *vector of ciphertext shares* of $s := f(0)$, s being denoted as the *(threshold) opening* of $c_{[n]}$, and, by extension, $s_j := f(j)$, $\forall j \in [n]$ as the *opening shares.*

Considering the latter extended definition, we may dub as "virtual" the opening shares s_j for which no explainable decryption of c_j under pk_j exists. In our use-case of Y-VSS, virtual shares of corrupt parties will be those for which the key pk_j is badly formed. Nevertheless, virtual shares will be a convenient computational intermediary, since the simulator in the UC proof of Section 5.1 will take as input *all* opening shares of corrupt parties, irrespective of those virtual or not. More precisely, one task of the simulator will be to simulate fake opening shares of honest parties into any arbitrary secret $s \in \mathbb{F}_p$. To this end, it will apply what we now define as the deterministic algorithm called *simulation of shares* and denoted as ShSim. On input any $s \in \mathbb{F}_p$ and $(s_i)_{i \in \mathcal{V}} \in \mathbb{F}_p^t$ for some t-sized subset $\mathcal{V} \subset [n]$, ShSim interpolates the unique polynomial $f \in \mathbb{F}_p[X]_{\leq t}$ through them, and outputs $(f(i))_{i \in [n] \setminus \mathcal{V}}$. We also formalize an algorithm, denoted as ShInfer, which does the task of infering the opening shares of corrupt parties (including virtual ones) from $t+1$ opening shares of honest parties. Namely, on input any $(s_i)_{i \in \mathcal{U}} \in \mathbb{F}_p^{t+1}$ for a $t+1$-subset $\mathcal{U} \subset [n]$, ShInfer interpolates the unique polynomial $f \in \mathbb{F}_p[X]_{\leq t}$ through them, and outputs $(f(i))_{i \in [n] \setminus \mathcal{U}}$. By definition of a vector of ciphertext shares, we deduce:

Property 4 (Perfect simulatability and inference of opening shares)**.** Let $c_{[n]} = (c_i)_{i \in [n]} \in \mathscr{C}^n$ be a *vector of ciphertext shares* for which a threshold opening $s \in \mathbb{F}_p$ exists, in the sense of Definition 3, and thus for which all n opening shares $(s_i)_{i \in [n]}$ are defined, then

- $\forall \mathcal{I} \subset [n]$ of size t, $(s_i)_{i \in [n] \setminus \mathcal{I}} = \mathsf{ShSim}(s, (s_j)_{j \in \mathcal{I}})$;
- $\forall \mathcal{U} \subset [n]$ of size $t+1$, $(s_j)_{j \in [n] \setminus \mathcal{U}} = \mathsf{ShInfer}((s_i)_{i \in \mathcal{U}})$.

We can now succinctly capture correctness and simulatability of the baseline method of Section 1.2 as follows. For readability with Figure 1, we re-name the keys as $(\mathsf{pk}'_j)_{j \in [n]}$.

Property 5. Consider any vector of Shamir shares $(s_i)_{i \in [n]}$ of some secret $s \in \mathbb{F}_p$, any $t+1$-sized subset of indices $\mathcal{U} \subset [n]$, any polynomials of degree $t+1$: $(f_i)_{i \in \mathcal{U}}$, $s_{i \to j} := f_i(j)$ $\forall i \in \mathcal{U}, j \in [n]$ and any encryption randomnesses $(\rho_{\text{enc},i,j})_{i \in \mathcal{U}, j \in [n]}$. Then, consider the vectors of encryptions of these re-sharings: $c_{i \to [n]} := \left(c_{i \to j} := \mathsf{Enc}(s_{i \to j} \mid \rho_{\text{enc},i,j})\right)_{j \in [n]}$, $\forall i \in \mathcal{U}$, we have that their Lagrange homomorphic linear combination $c'_{[n]}$, as made explicit in Equation (1), is a *vector of ciphertext shares* in the sense of Definition 3.
If at least $t+1$ keys pk_j are explainable, then $c'_{[n]}$ has sharing polynomial equal to $f' := \sum_{i \in \mathcal{U}} \lambda_i^{\mathcal{U}} f_i$ and threshold opening equal to s. *Else*, $c'_{[n]}$ has no threshold opening in the sense of Definition 3.

Proof. Definition 2 guarantees that $c'_{[n]}$ is a vector of ciphertext shares. In the case "*If*": the claim on the sharing polynomial follows from Definition 2 then unique interpolation of f' from $t+1$ values. We have $f'(0) = s$ by Equation (3) (see Appendix B.1 for more details), hence $c'_{[n]}$ has threshold opening s. □

4. Self-contained description of Y-VSS

We now present the actual version, in 2 messages delay (2δ). Before we start, let us outline the difference with the simplified version in 3 messages, presented in Section 1.3. Now, the entering committee $\mathcal{C}'$ does not need to speak at the end of a Refresh. To enable this, we somehow "pipeline" the validation messages of $\mathcal{C}'$, into the messages which $\mathcal{C}'$ will later send in the next Refresh, as exiting committee. To accommodate this, exiting and entering parties in a Refresh do not anymore have their inputs and outputs which are local lists of VPS, but instead, local lists of PPS.

The protocol Y-VSS consists of three subprotocols: for sharing, refreshing and opening a secret. They are described in Sections 4.1 to 4.3 and are further formalized in Appendices C.1 to C.3. For simplicity, all the description and the proof of Y-VSS impose that, upon receiving refresh sig (or share sig), every member of an exiting committee $\mathcal{C}$ (or the dealer $\mathcal{D}$) aborts if it cannot retrieve from the PKI a complete list of n public keys in $\mathscr{pK}$ for the entering committee $\mathcal{C}'$. We explain in Appendix C.4 how to remove this simplification. When a party does not abort, then it must be the case that the list of keys retrieved includes all those of the $t+1$ honest parties. This enables the simplification that a vector of ciphertext shares always has n opening shares, in the extended terminology of Definition 3, and a threshold opening. We thus assume from now on the public parameter of a list of n public keys for every committee, e.g., denoted as $(\mathsf{pk}_i)_{i \in [n]}$ and $(\mathsf{pk}'_i)_{i \in [n]}$ for some exiting $\mathcal{C}$ and entering $\mathcal{C}'$. The assumption implies that all $t+1$ honest parties published their keys.

4.1 Y-VSS Share

The dealer $\mathcal{D}$ generates a vector of n shares of its secret s: $(s_i = f(i))_{i \in [n]}$, under the Shamir secret-sharing scheme. Then it generates the n-sized vector of encryptions $c_{[n]} := \left(\mathsf{Enc}_{\mathsf{pk}_i}(s_i)\right)_{i \in [n]}$ of the shares under the public keys of the parties of the first committee $\mathcal{C}^{(1)}$. It also generates a NIZK AoK, denoted π_{pvS}, of: s, the sharing polynomial f, and the encryption randomnesses explaining $c_{[n]}$. Following the tradition, we call such a pair $(c_{[n]}, \pi_{\mathsf{pvS}})$ as a *publicly verifiable sharing*, denoted as a pvS and further formalized in Appendix B.3. Finally, $\mathcal{D}$ broadcasts to $\mathcal{C}^{(1)}$ $(c_{[n]}, \pi_{\mathsf{pvS}})$ appended with its signature $\sigma_{\mathcal{D}}$ on it. For consistency with the rest of the protocol, we denote this triple as $\mathsf{pps}_{\mathcal{D}} := (\sigma_{\mathcal{D}}, \pi_{\mathsf{pvS}} \xrightarrow{\mathscr{L}} c_{[n]})$ and call it as a *proven proactivized sharing* relatively to $\mathcal{C}^{(1)}$. If parties of $\mathcal{C}^{(1)}$ are delivered something else from the broadcast, which can happen if $\mathcal{D}$ is corrupt, then they replace it by a pre-defined default PPS of 0, denoted as $\mathsf{pps}_0 = \big(\bot, \bot \xrightarrow{\mathscr{L}} (\mathsf{Enc}_{\mathsf{pk}_i}(0))_{i \in [n]}\big)$.

4.2 Y-VSS Refresh$(\mathcal{C}, \mathcal{C}')$

We now describe a Refresh between a committee $\mathcal{C} := \mathcal{C}^{(e)}$ which is about to exit, and the entering one $\mathcal{C}' := \mathcal{C}^{(e+1)}$. In Sections 4.2.1 and 4.2.2 we describe the outputs of $\mathcal{C}'$ and inputs of $\mathcal{C}$. In Section 4.2.3 we describe the first step, which is the multicasts by each parties of $\mathcal{C}$ to the committee $\mathcal{C}'_{\text{collec}}$ of the $t+1$ "collectors". In Section 4.2.4 we describe the second step, in which each collector responsively multicasts a PPS, to $\mathcal{C}'$. Then in Section 4.2.5 we formalize the reception by $\mathcal{C}'$ of these PPS and their appending to their lists of outputs.

4.2.1 Outputs of $\mathcal{C}'$. Each party P'_j of the entering committee $\mathcal{C}'$ starts with an empty state, initializes an empty list denoted $\mathrm{ListOfPps}'_j$, and progressively adds up to $t+1$ objects in this list, denoted as *proven proactivized sharings*, shortened as PPS. Broadly speaking, a PPS relatively to committee $\mathcal{C}'$, consists of a vector of ciphertext $c'_{[n]} \in \mathscr{C}^n$, appended with enough data proving that $c'_{[n]}$ is formed by applying the baseline method described in Section 1.2 and Equation (1). In more details, a PPS relatively to $\mathcal{C}'$ is a tuple of the form $\mathsf{pps}' = \big((c_{[n]}, \mathsf{qvc}), \{\mathsf{pvr}_i\}_{i \in \mathcal{U}}\big)$, and denoted as

$\boldsymbol{\mathsf{pps}' = \Big(\mathsf{vps} := (c_{[n]}, \mathsf{qvc}), \{\mathsf{pvr}_i\}_{i \in \mathcal{U}} \xrightarrow{\mathscr{L}} c'_{[n]}\Big)}$, where:

- $\boldsymbol{c_{[n]} = (c_i)_{i \in [n]} \in \mathscr{C}^n}$ is a vector of ciphertexts and qvc is the concatenation of signatures on $c_{[n]}$ issued by $t+1$ out of n parties of $\mathcal{C}$. We denote qvc as a *quorum verification certificate* because, roughly, each signer attests that $c_{[n]}$ was itself formed by applying the baseline method in the previous refresh, when $\mathcal{C}$ was entering. Hence, we denote such a pair $\mathsf{vps} = (c_{[n]}, \mathsf{qvc})$ as a *verified proactivized sharing*, dubbed as a VPS. We say that two VPSs are the same as soon as their $c_{[n]}$ are the same, nonwithstanding their qvc can have different sets of signers;
- $\mathcal{U} \subset [n]$ is a subset of size $t+1$;
- $\boldsymbol{\mathsf{pvr}_i = (c_{i \to [n]}, \pi_{\mathsf{pvR},i})}$, for $i \in \mathcal{U}$, is what we denote as a *publicly verifiable re-sharing* of the coordinate c_i of the i-th party $P_i \in \mathcal{C}$. It consists of a vector of ciphertexts $c_{i \to [n]} \in \mathscr{C}^n$, and of a NIZK AoK, denoted as $\pi_{\mathsf{pvR},i}$, of: P_i's decryption key, of a decryption s_i of c_i, of a (Shamir sharing) polynomial $f_i \in \mathbb{F}_p[X]_{\leq t}$ and of encryption randomnesses, proving that $c_{i \to [n]}$ is an encrypted sharing of s_i under the keys of $\mathcal{C}'$. pvRs are further formalized in Appendix B.3.
- $\boldsymbol{c'_{[n]}} = \boxplus_{i \in \mathcal{U}} \lambda_i^{\mathcal{U}} \boxdot c_{i \to [n]}$, exactly as in Equation (1). Although $c'_{[n]}$ is not actually included in the pps′, we display it because it is publicly efficiently computable from pps′.

4.2.2 Inputs of $\mathcal{C}$. Each party $P_i \in \mathcal{C}$ starts with a list of PPS: $\mathrm{ListOfPps}_i$. In particular if $\mathcal{C} = \mathcal{C}^{(1)}$, then this list consists of the unique PPS obtained from the sharing step.

4.2.3 Encrypted re-sharing. Upon receiving the signal refresh sig, each party $P_i \in \mathcal{C}$, for every PPS: $\mathsf{pps} = \big(\ldots, \ldots \xrightarrow{\mathscr{L}} c_{[n]} = (c_i)_{i \in [n]}\big)$ in its list $\mathrm{ListOfPps}_i$, forms a triple $\big(c_{[n]}, \mathsf{pvr}_i, \sigma_i\big)$ as follows:

- it generates a *publicly verifiable resharing* of its encrypted share c_i, denoted as $\mathsf{pvr}_i = (c_{i \to [n]}, \pi_{\mathsf{pvR},i})$. Concretely: P_i decrypts c_i into s_i, generates a Shamir sharing $(s_{i \to j})_{j \in [n]}$ of s_i, then encrypts each coordinate $s_{i \to j}$ under the public key of $P'_j \in \mathcal{C}'$ into $c_{i \to j}$. It obtains the vector of encrypted shares $c_{i \to [n]} := (c_{i \to j})_{j \in [n]}$. It appends to it a NIZK AoK of correct resharing: $\pi_{\mathsf{pvR},i}$. This is depicted in Figure 1.
- it generates a signature on $c_{[n]}$, denoted σ_i, by which P_i attests that $c_{[n]}$ is obtained from a (valid) PPS.

P_i multicasts to $\mathcal{C}'_{\text{collec}}$, all at once, all the triples obtained, then shuts-off. We denote such a triple as a *resharing message*.

4.2.4 Selection & combination of re-sharings into a PPS. Each collector $K'_k \in \mathcal{C}'_{\text{collec}}$ waits until it receives, for some $(t+1)$-sized subset $\mathcal{U} \subset [n]$, resharing messages from parties in $\mathcal{U}$ formed out of the *same* vector of ciphertext shares $c_{[n]}$:

$$\big(c_{[n]}, \mathsf{pvr}_i, \sigma_i\big), \quad \forall i \in \mathcal{U} \;. \tag{4}$$

Then it combines the $t+1$ signatures into a qvc for $c_{[n]}$, thereby obtaining a verified proactivized sharing: $\mathsf{vps} := (c_{[n]}, \mathsf{qvc})$. It multicasts to $\mathcal{C}'$ the PPS obtained: $\big(\mathsf{vps}, \{\mathsf{pvr}_i\}_{i \in \mathcal{U}} \xrightarrow{\mathscr{L}} \ldots\big)$, then shuts-off.

4.2.5 Ever-growing outputs of $\mathcal{C}'$. Upon receiving a PPS, any party $P'_j \in \mathcal{C}'$ adds it to its $\mathrm{ListOfPps}'_j$.

4.3 Y-VSS Opening of a secret

We consider any arbitrary committee $\mathcal{C}^{(e_o)}$ which is instructed to open the secret to a designated learner $\mathcal{L}$. Each party $P_i \in \mathcal{C}^{(e_o)}$, for every $\mathsf{pps}^{(e_o)} := \big(\ldots, \ldots \xrightarrow{\mathscr{L}} c^{(e_o)}_{[n]} = (c^{(e_o)}_i)_{i \in [n]}\big)$ in its list $\mathrm{ListOfPps}_i$ (containing up to $t+1$ elements):

- generates a decryption s_i of its encrypted share c_i, which we dub as an *opening share*, generates a NIZK of correct decryption under its public key, denoted as a $\pi_{\mathsf{pvD},i}$, to obtain the triple $(c^{(e_o)}_{[n]}, s_i, \pi_{\mathsf{pvD},i})$.

It privately sends to $\mathcal{L}$ all the (up to $t+1$) triples obtained. Upon receiving any $t+1$ such triples for some same $c^{(e_o)}_{[n]}$, $\mathcal{L}$ outputs the Lagrange linear combination of the $(s_i)_i$.

5. Analysis of Y-VSS

In Theorem 6 we state security and liveness of Y-VSS. Since the proof of liveness is even simpler than the one sketched, in Section 1.3, for the simpler variant in 3δ, we defer it to Appendix D.2. In Proposition 7 we state and prove a useful intermediary property of Y-VSS denoted as *correctness*, which is happens to be synonymous of "public verifiability" [65]. We defer to Appendix D.3 the proof of UC security when the dealer $\mathcal{D}$ is corrupt, since it essentially rephrases correctness. Then in Section 5.1 we prove UC security in the harder case where $\mathcal{L}$ is honest.

Theorem 6. *Protocol* Y-VSS *in the ($\mathcal{F}_{BC}$, $\mathcal{F}_{CA}$, $\mathcal{F}_{NIZK}$, $\mathcal{F}_{AT}$, $\mathcal{F}_{ST}$, $\mathcal{F}_{Sch}$)-hybrid model is such that:*

(Security) *it UC-implements $\mathcal{F}_{VSS}$, in the precise sense of Appendix A.2, which follows [15].*

(Liveness) *If $\mathcal{L}$ is honest, then it outputs in any execution which is complete in the following sense. Borrowing the terminology of [52], for some $1 \leq e_o$, we say that an execution is* complete up to e_o, *or* complete *for short, if:*

- *[LF] $\mathcal{A}$ responds, at some point, to every request from the functionalities ($\mathcal{F}_{BC}$, $\mathcal{F}_{CA}$, $\mathcal{F}_{NIZK}$, $\mathcal{F}_{AT}$, $\mathcal{F}_{ST}$).*
- *[LS] If $\mathcal{D}$ is honest: $\mathcal{F}_{Sch}$ sends* sharesig *to $\mathcal{D}$,* only after *$\mathcal{A}$ has allowed the publication on $\mathcal{F}_{CA}$ of all keys of honest parties in $\mathcal{C}^{(1)}$;*
- *[LK] for all $e \in [1, \ldots, e_o - 1]$ (possibly empty), then all honest parties of $\mathcal{C}^{(e+1)}$ receive* startsig *and $\mathcal{A}$ allows the publication of their keys on $\mathcal{F}_{CA}$. $\mathcal{F}_{CA}$ does not publish any further key after the first party in $\mathcal{C}^{(e)}$ received* refreshsig. *This implies that, upon receiving* refreshsig, *all parties in $\mathcal{C}^{(e)}$ are able to recover from $\mathcal{F}_{CA}$ the* same *list of public keys, including all those of honest parties.*
- *[LR] for all $e \in [1, \ldots, e_o - 1]$ (possibly empty), after all honest keys of $\mathcal{C}^{(e+1)}$ were published on $\mathcal{F}_{CA}$, but at least 2δ after the last party of $\mathcal{C}^{(e-1)}$ received* refreshsig, *then $\mathcal{F}_{Sch}$ sends* refreshsig *to each party $P_i^{(e)} \in \mathcal{C}^{(e)}$;*
- *[LO] $\mathcal{F}_{Sch}$ sends* Open *to all honest parties in $\mathcal{C}^{(e_o)}$.*

Proposition 7 (Correctness of Y-VSS)**.** *Consider the output of the broadcast of $\mathcal{D}$, if it exists. There exists a value, denoted as $\widetilde{s}$, which is fully determined from such that $\widetilde{s}$ is the only value that $\mathcal{L}$ can possibly output. More precisely: if this output is a (valid)* pvS, *then $\widetilde{s}$ is equal to its threshold opening (in particular, is $\mathcal{D}$'s secret if it is honest), else, $\widetilde{s}$ equals* 0.

Proof. Let us define $\widetilde{s}$ as: either the threshold opening of the broadcast of the $\mathcal{D}$, if it is a (valid) pvS, or, $\widetilde{s} := \mathsf{pvs}_0$ the pre-defined pvS of 0. Let us first show that correctness follows from the following:

- *Claim:* for any $e \geq 1$, then every PPS relatively to $\mathcal{C}^{(e)}$ ever formed in the execution, has its last component $c'_{[n]}$ which is a *vector of ciphertext shares* in the sense of Definition 3, and which has threshold opening equal to $\widetilde{s}$.

Indeed, when $\mathcal{L}$ receives $t+1$ triples for the same $c_{[n]}^{(e_o)}$, then since one of the senders is honest, it must be that $c_{[n]}^{(e_o)}$ was formed (" $\xrightarrow{\mathcal{L}}$ ") out of a PPS relatively to committee $\mathcal{C}^{(e_o)}$. It remains to show the *Claim*, by recursion on e. It trivially holds for $e=1$. Let us assume that it holds up to e. Let us prove that the claim holds for all PPSs relatively to committee $\mathcal{C}' := \mathcal{C}^{(e+1)}$, which will conclude the recursion. Let us consider one such PPS pps′. Its first component is a VPS: $(c_{[n]}, \mathsf{qvc})$, so since there is at least one honest signer, this proves that $c_{[n]}$ was formed (" $\xrightarrow{\mathcal{L}}$ ") out of a PPS relatively to committee $\mathcal{C} := \mathcal{C}^{(e)}$. Hence $c_{[n]}$ is a vector of ciphertext shares of $\widetilde{s}$ by the recursion assumption. Let us denote as $(s_i)_{i \in [n]}$ its opening shares. Finally, consider the $t+1$ publicly verifiable resharings: $\mathsf{pvr}_i = (c_{i \to [n]}, \pi_{\mathsf{pvR},i})$ $\forall i \in \mathcal{U}$ enclosed in pps′. For each $i \in \mathcal{U}$, the NIZK AoKs $\pi_{\mathsf{pvR},i}$ guarantees that $c_{i \to [n]}$ is of the form $(\mathsf{Enc}_{\mathsf{pk}'_j}(s_{i \to j} | \rho_{\mathrm{enc},i,j}))_{j \in [n]}$, where the $(s_{i \to j})_{j \in [n]}$ form a vector of shares of s_i. Hence, by Property 5 we conclude that $c'_{[n]}$ is a vector of ciphertext shares, with threshold opening equal to the same $\widetilde{s}$. $\square$

5.1 Proof of UC security in case of a honest dealer $\mathcal{D}$. We start by considering a real execution $\mathrm{REAL}_{\mathcal{A}}$ of Y-VSS, with adversary $\mathcal{A}$ fully controlled by the environment $\mathcal{E}$. $\mathcal{E}$ assigns its input to $\mathcal{D}$ and listens to the output (if any) of $\mathcal{L}$. Then we go through a series of hybrid games, which we show indistinguishable from one with the next, from the point of view of $\mathcal{E}$. In the final game, denoted as $\mathsf{Hyb}^{\mathsf{0Share}}$, the view of $\mathcal{A}$ is generated without any direct interaction with the honest parties. In particular, the $\mathcal{D}$ broadcasts a pvS of 0, instead of its actual secret. The only indirect interaction with honest parties happens in the opening, via $\mathcal{F}_{\mathrm{VSS}}$, which delivers the actual value of s, which helps us to simulate the opening shares of s. So what we are describing in $\mathsf{Hyb}^{\mathsf{0Share}}$ is a simulator which interacts only with $\mathcal{F}_{\mathrm{VSS}}$ and $\mathcal{E}$, which concludes the UC proof.

The purpose of the games $\mathsf{Hyb}^{\mathsf{ShSim}}$ and $\mathsf{Hyb}^{s\mathrm{Open}}$, which are not needed if $\mathcal{L}$ is honest, is to make so that the view of $\mathcal{E}$ is generated without using the private keys of honest parties of $\mathcal{C}^{(e_o)}$, nor using the plaintext shares of honest parties of $\mathcal{C}^{(e_o)}$. This allows to apply IND-CPA of PKE in subsequent games. In particular in $\mathsf{Hyb}^{\mathsf{0Refresh}}[1, n]$ we achieve that all re-sharings are actually mere $\mathsf{pvS}(0)$.

Game $\mathbf{REAL}_{\mathcal{A}}$. This is the actual execution of the protocol Y-VSS with environment $\mathcal{E}$, adversary $\mathcal{A}$ and ideal functionalities $\mathcal{F}_{\mathrm{BC}}$, $\mathcal{F}_{\mathrm{CA}}$, $\mathcal{F}_{\mathrm{NIZK}}$, $\mathcal{F}_{\mathrm{AT}}$, $\mathcal{F}_{\mathrm{ST}}$, $\mathcal{F}_{\mathrm{Sch}}$. We make the change (not formalized by a game) that $\mathcal{F}_{\mathrm{NIZK}}$ does not check validity of witnesses (if any) received from honest parties nor from $\mathcal{D}$. This does not change its outputs, since honest participants always provide correct witnesses when querying $\mathcal{F}_{\mathrm{NIZK}}$ in the actual protocol.

Game $\mathsf{Hyb}^{\mathsf{ShSim}}$. Unchanged if $\mathcal{L}$ honest. For each PPS: $\mathsf{pps}' = \left((c_{[n]}, \mathsf{qvc}), \{\mathsf{pvr}_i\}_{i \in \mathcal{U}} \xrightarrow{\mathcal{L}} c'_{[n]}\right)$ opened by $\mathcal{C}^{(e_o)}$ to $\mathcal{L}$, the opening shares of honest parties in $\mathcal{C}^{(e_o)}$ are now computed as $\mathsf{ShSim}(s_{c'_{[n]}}, (s'_j)_{j \in \mathcal{I}'})$, where:

- $s_{c'_{[n]}}$ is the threshold opening of $c'_{[n]}$;
- $\mathcal{I}' \subset [n]$ are the t indices of corrupt parties in $\mathcal{C}^{(e_o)}$;
- $(s'_j)_{j \in \mathcal{I}'}$ are the opening shares of $c'_{[n]}$ of corrupt parties, computed as follows. Each $\mathsf{pvr}_i = (c_{i \to [n]}, \pi_{\mathsf{pvR},i})$, $\forall i \in \mathcal{U}$, must be of the form $c_{i \to [n]} = \left(\mathsf{Enc}_{\mathsf{pk}'_j}(s_{i \to j} | \rho_{\mathrm{enc},i,j})\right)_{j \in [n]}$, where $(s_{i \to j})_{j \in [n]}$ is a vector of shares of s_i. For a honest party this is automatic since it follows Y-VSS. For a corrupt party this is guaranteed by the NIZK $\pi_{\mathsf{pvR},i}$. We finally set $s'_j := \sum_{i \in \mathcal{U}} \lambda_i^{\mathcal{U}} s_{i \to j}$, $\forall j \in \mathcal{I}'$

Claim 7.1. *$REAL_{\mathcal{A}} \equiv$* $\mathsf{Hyb}^{\mathsf{ShSim}}$.

By the *If* case of Property 5, the method to compute the (s'_j) indeed returns the opening shares of $c'_{[n]}$. Then, by Property 4, ShSim does return the opening shares of $c'_{[n]}$ with indices in $[n] \setminus \mathcal{I}$.

Game $\mathsf{Hyb}^{s\mathrm{Open}}$. Unchanged is $\mathcal{L}$ honest. Else (if $\mathcal{L}$ corrupt), this game differs from $\mathsf{Hyb}^{\mathsf{ShSim}}$ in that the input $s_{c_{[n]}}$ to ShSim is replaced by the actual secret s of the dealer $\mathcal{D}$.

Claim 7.2. $\mathsf{Hyb}^{\mathsf{ShSim}} \equiv \mathsf{Hyb}^{s\mathrm{Open}}$.

Proof. By Proposition 7, for each PPS: $\mathsf{pps}' = (\ldots, \ldots, c'_{[n]})$ relatively to committee $\mathcal{C}^{(e_o)}$, we have that $c'_{[n]}$ is a vector of ciphertext shares with threshold opening equal to s. □

From this point, neither the secret keys of honest parties in $\mathcal{C}^{(e_o)}$, nor their plaintext shares, are used anymore to generate the view of $\mathcal{E}$.

Games $\mathsf{Hyb}^{\mathsf{0Refresh}}[e, i]$ for each $e \in [e_o\text{-}1, \ldots, 1]$ (in this backwards order) then each $i \in [0, \ldots, n]$. We set $\mathsf{Hyb}^{\mathsf{0Refresh}}[e_o\text{-}1, 0] := \mathsf{Hyb}^{\mathsf{ShSim}}$. Then for each $e \in [e_o\text{-}1, \ldots, 1]$ and $i \in [0, \ldots, n]$: if $P^{(e)}_{i+1}$ is corrupt then we leave $\mathsf{Hyb}^{\mathsf{0Refresh}}[e, i+1] := \mathsf{Hyb}^{\mathsf{0Refresh}}[e, i]$ unchanged, otherwise if it is honest, then we modify $\mathsf{Hyb}^{\mathsf{0Refresh}}[e, i]$ into $\mathsf{Hyb}^{\mathsf{0Refresh}}[e, i+1]$ as follows. For each $\mathsf{pps} = (\ldots, \ldots \xrightarrow{\mathscr{L}} c^{(e)}_{[n]} = (c^{(e)}_i)_{i \in [n]})$ in the local list of $P^{(e)}_{i+1}$, in place of the resharing $c^{(e)}_{i \to [n]}$ of $c^{(e)}_i$ that $P^{(e)}_{i+1}$ generates, we substitute the vector of ciphertexts $c^{(e)}_{i \to [n]} := \mathsf{Enc}_{\mathsf{pk}'_j}(s_j)_{j \in [n]}$, where the t shares of the corrupt indices $(s_j)_{i \in \mathcal{I}^{(e+1)}}$ are sampled uniformly at random in $\mathbb{F}_p^t$, while the $t+1$ others are set to 0. In particular, *the secret decryption key of $P^{(e)}_{i+1}$ is not used anymore*. Notice that $\mathcal{F}_{\mathrm{NIZK}}$ still issues proofs of correct resharing, since it does not check any witness from honest parties. When reaching $i = n$, if $e \geq 2$, then we set $\mathsf{Hyb}^{\mathsf{0Refresh}}[e - 1, 0] := \mathsf{Hyb}^{\mathsf{0Refresh}}[e, n]$.

Claim 7.3. $\mathsf{Hyb}^{s\mathrm{Open}} \equiv \mathsf{Hyb}^{\mathsf{0Refresh}}[1, n]$

Proof. It is enough to show that for each $e \in [e_o\text{-}1, \ldots, 1]$, for $i \leq n-1$ such that $P^{(e)}_{i+1}$ is honest, then $\mathsf{Hyb}^{\mathsf{0Refresh}}[e, i]$ is indistinguishable from $\mathsf{Hyb}^{\mathsf{0Refresh}}[e, i+1]$. Let us consider one of the PPSs: $\mathsf{pps} = (\ldots, \ldots \xrightarrow{\mathscr{L}} c^{(e)}_{[n]} = (c^{(e)}_i)_{i \in [n]}$ in the list of $P^{(e)}_{i+1}$. The high level idea simply consists in showing indistinguishability between the two re-sharings $c_{i \to [n]}$ of c_i: the actual one, in $\mathsf{Hyb}^{\mathsf{0Refresh}}[e, i]$, and the bogus one, in $\mathsf{Hyb}^{\mathsf{0Refresh}}[e, i+1]$. Let us make the simplifying *Assumption* that PKE perfectly hides the plaintext of the coordinates of $c_{i \to [n]}$ encrypted under the keys of honest parties. Then, the view of $\mathcal{E}$ is fully determined by the t plaintext coordinates $(s_i)_{i \in \mathcal{I}}$, where $\mathcal{I} \subset [n]$ is the t-sized subset of corrupt parties. But by t-privacy of Shamir sharing, they also vary uniformly at random also in the actual $c_{i \to [n]}$. So the two distributions of views are equal. It remains to substantiate this *Assumption*, which will conclude this sktech proof. The reason is that we have that the view of $\mathcal{E}$ is generated without using (i) the secret decryption keys of parties in $\mathcal{C}^{e+1}$, (ii) nor the plaintext shares of $c_{i \to [n]}$ with indices of the honests parties in $\mathcal{C}^{(e+1)}$. Indeed, if $e = e_o - 1$ then (i) and (ii) are thanks to $\mathsf{Hyb}^{s\mathrm{Open}}$, while if $e < e_o - 1$ then (i) and (ii) are thanks to $\mathsf{Hyb}^{\mathsf{0Refresh}}[e+1, n]$. So by (i) and (ii) we are in the conditions of applicability of IND-CPA of PKE.

In Corollary 11 of Appendix B.4 we give a formal statement and proof of the indistinguishability between the two $c_{i \to [n]}$, denoted as "IND-CPA of re-sharing". In Appendix D.1 we formalize the reduction from distinguishing between $\mathsf{Hyb}^{\mathsf{0Refresh}}[e, i]$ and $\mathsf{Hyb}^{\mathsf{0Refresh}}[e, i+1]$. A subtlety in both the reduction and statement is that the input of the re-sharing is a ciphertext c_i, not a plaintext secret. So in order to escape issues with IND-CCA, we have to consider an adversary concatenating enough entities so that it can be extracted, from the whole, a plaintext of c_i. Another subtlety is that in the case $e = e_o\text{-}1$, then the view of $\mathcal{E}$ is generated from the plaintext sub-shares of corrupt parties, via ShSim. However, it could be the case that the keys of corrupt parties are badly generated, so that $\mathcal{E}$ is unable to decrypt their plaintext sub-shares. So in order to make the reduction work, we somehow leak to $\mathcal{E}$ these plaintext shares then capture this leakage in Corollary 11. □

Game $\mathsf{Hyb}^{\mathsf{0Share}}$. We modify $\mathsf{Hyb}^{\mathsf{0Refresh}}[1, n]$ in that the dealer $\mathcal{D}$ plays the protocol as if it had input 0, even though it still sends s to $\mathcal{F}_{\mathsf{VSS}}$.

Claim 7.4. $\mathsf{Hyb}^{\mathsf{0Refresh}} \equiv \mathsf{Hyb}^{\mathsf{0Share}}$

Proof. Since $\mathsf{Hyb}^{\mathsf{0Refresh}}[1, n]$, to generate the view of $\mathcal{E}$, we neither use the private decryption keys of honest parties of $\mathcal{C}^{(1)}$, nor the plaintext shares of the pvS of the dealer. Thus we can apply IND-CPA of encrypted sharing (Proposition 9) to the pvS of $\mathcal{D}$, which is encrypted under the public keys of $\mathcal{C}^{(1)}$. □

Game $\mathsf{Hyb}^{\mathsf{ShInfer}}_{\mathsf{Open}}$. If $\mathcal{L}$ is honest, this game is identical to $\mathsf{Hyb}^{\mathsf{0Share}}$. Else (is $\mathcal{L}$ is corrupt): for each PPS: $\mathsf{pps}' = (\ldots, \{\mathsf{pvr}_i\}_{i \in \mathcal{U}}, c'_{[n]})$ held by at least one honest party in $\mathcal{C}^{(e_o)}$, we now change the method to infer the opening shares of $c'_{[n]}$ of corrupt parties fed into ShSim. Precisely, in the method in $\mathsf{Hyb}^{\mathsf{ShSim}}$, for each pvr_i, $i \in \mathcal{U}$ (if any) generated by a corrupt party i in $\mathcal{C}^{(e_o - 1)}$, the plaintext coordinates $(s_{i \to j})_{j \in \mathcal{I}}$ of corrupt parties were extracted from the NIZK AoK of resharing. Instead, we now infer the opening shares of pvr_i of corrupt parties as follows. We use the secret keys of the $t+1$ honest parties in $\mathcal{C}^{(e_o)}$ to correctly decrypt their opening shares $(s_{i \to j})_{j \in [n] \setminus \mathcal{I}}$ of $c'_{[n]}$. Then we use them as inputs of $\mathsf{ShInfer}$, i.e., we do a polynomial interpolation from $t+1$ points, to obtain the desired t opening shares $(s_{i \to j})_{j \in \mathcal{I}}$ of corrupt parties, of $c'_{[n]}$. We claim that the $(s_{i \to j})_{j \in \mathcal{I}}$ obtained are unchanged. Indeed, the $(s_{i \to j})_{j \in [n] \setminus \mathcal{I}}$ used are by definition opening shares of $c'_{[n]}$. So since $c'_{[n]}$ is a vector of ciphertext shares, then the Lagrange interpolation which we did (formalized by ShInfer in Property 5) does return $(s_{i\to j})_{j\in\mathcal{I}}$.

6. Our Implementation, using Elgamal

The plaintext space is not $\mathbb{Z}$ but instead an abelian group, denoted $\mathbb{G}$ in additive notation, of known prime order q and in which DDH is hard (as in the second item Section 7.4) The observation which we make is that all elementary tasks needed for re-sharing, i.e.: key generation, decryption, polynomial evaluation and encryption, are linear maps over source and target spaces of the form $(\mathbb{Z}/q\mathbb{Z})^{n_0}\times\mathbb{G}^{n_1}$. Thus the relation to be proven in NIZK for resharing is itself a linear map. It could be proven using the elementary protocol of [4, p. 4.1], which is compressible using Bulletproofs-like techniques [4, p. 4.3]. After the first version of this work, [21, Fig. 10] independently formalized NIZKs of resharing for Elgamal, denoted as "HEPVSS". They use an even simpler proof of preimage of linear maps (their Fig. 2, with proof equal to one element of the source and one of the target) which does not even require a public uniform random string. We thus implemented Y-VSS using HEPVSS. We used about 1000 lines of Go code, with operations in the 254-bit "pairing-friendly" BN254 Barreto-Naehrig curve [7] over a 254-bit base field. Our code uses the gnark-crypto [12] library, and we use EdDSA for signing and SHA256 for hashing. We run this program on a laptop that we had access to, featuring Apple M1 CPU, i5-2500 running at 3.2GHz with 8 cores and 8GB RAM. The software configurations included gnark-crypto 0.7 and Golang 1.18.2. We also implemented Y-VSS using the alternative PKE and NIZK of resharing denoted "DHPVSS" in [21, p. 5.2] (provided $\deg(m^*)\leq n-t-1$). DHPVSS has a proof of size of only 3 group elements and turned out to be 2.5 times faster. But we do not further report on it, since "DHPVSS" requires a one-time-pad-sized PKI.

We first present in Fig. 4 the performance of $\mathsf{Refresh}(\mathcal{C},\mathcal{C}')$ between committees $\mathcal{C}$ and $\mathcal{C}'$, with number of parties from 11 to 101. Namely, we measure the time, in terms of local computations, required to reshare a secret from a committee $\mathcal{C}$ to a new committee $\mathcal{C}'$. We consider the scenario taking the worst-case computation time, in which all collectors of both the previous and the current Refresh are honest and all the $t+1$ PPSs from them are received in time. Hence, each player in $\mathcal{C}$ does $t+1$ resharings, then each player in $\mathcal{C}'$ receives $t+1$ PPSs. In-between, we consider that each collector K'_k verifies only one batch of $t+1$ resharing messages. Indeed, assume that K'_k receives $t+1$ re-sharing messages out of the same VPS: $(c_{[n]},\mathsf{qvc})$ but which do not pass its verification. Then this means for K'_k that this batch contains a proof of openly mis-behaving of some party in $\mathcal{C}$, so we consider that this deterrence is enough to exclude such open mis-behavior. As can be seen, about 90% of the time was spent performing the final local computation by members of $\mathcal{C}'$, namely, to verify what they receive before adding it to their lists of outputs. All these operations are by definition parallelizable by at least a factor $t+1$, thus the time should be divided by the number of CPUs. This is illustrated in Appendix F.1, e.g., in Figure 4 we approximately used 2 cores (out of 8), technically: 2 Go-routines.

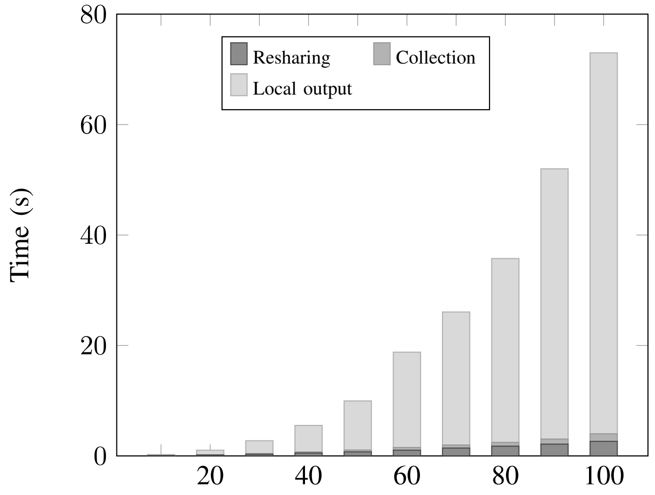


Figure 4: Total worst-case time of a $\mathsf{Refresh}(\mathcal{C},\mathcal{C}')$ between two committees $\mathcal{C}$ and $\mathcal{C}'$, measured in seconds, using approximately 2 cores (out of 8). The x-axis is the size n of both committees, ranging from 11 to 101. We specify for each setting the time spent in each of the high-level subroutine: **Encrypted re-sharing**: for one party in $\mathcal{C}$ to generate $t+1$ pvRs and signatures (Section 4.2.3), **Selection & combination** for a single collector to verify $t+1$ pvRs and signatures (Section 4.2.4) , **Local output** for a party in $\mathcal{C}'$, for each of the $t+1$ PPS which it receives (we consider the worst-case): of verification of the $t+1$ NIZKs of resharing $((\pi_{\mathsf{pvR},i})_{i\in\mathcal{U}})$, of the $t+1$ signatures qvc on $c_{[n]}$, and of applying the homomorphic Lagrange linear combination to obtain $c'_{[n]}$.

Microbenchmarks. To better understand the sources of overhead in our protocol, we measure the costs of the underlying primitives. Here, pvReshare designates the operation for resharing a ciphertext and for producing a NIZK proof, NIZK.Verify the verification of a NIZK proof of resharing and Lagrange Comb the combination of $t+1$ resharings following Fig. 1. Finally, Sign designates the operation to sign a ciphertext and Sign.Verify the verification of such a signature. In Table 5, we report the cost in terms of computing time on a CPU.

	# of players			
	11	21	51	101
pvReshare	6.1 ms	11.1 ms	27.6 ms	52.3 ms
NIZK.Verify	3.9 ms	6.34 ms	15.3 ms	30.1 ms
Lagrange Comb (Fig. 1)	4.8 ms	17.1 ms	100.8 ms	433.0 ms
Sign	0.74 ms	0.79 ms	0.95 ms	1.21 ms
Sign.Verify	1.4 ms	1.5 ms	1.6 ms	1.9 ms

Table 5: Cost of Operations (Average over 100 trials)

7. Efficient generalizations and comparison

7.1 Elgamal in the exponent. A desirable use-case is when the secret s is in $\mathbb{F}_p:=\mathbb{Z}/p\mathbb{Z}$, and thus when the plaintext space of PKE is $\mathbb{Z}/p\mathbb{Z}$, which is not the case for Elgamal. The observation which we make is that this can be achieved by encoding the secret *in the exponent* of Elgamal, as in [62, p11]. Namely, consider as in Section 6: $\mathbb{G}$ an abelian group of prime order q, in additive notations, with hard DDH, and g some a public generator. Then, encryption of $s \in \mathbb{Z}/p\mathbb{Z}$ consists in Elgamal encryption of $s.g$. Decryption proceeds by Elgamal decryption, followed by computing the discrete logarithm, i.e., extracting s from $s.g$, then taking $\bmod p$. This scheme satisfies Definition 2, i.e., has perfect correctness $\bmod p$ after one homomorphic addition of size $t{+}1$, as soon as discrete logarithms of size up to $p^2(t{+}1)$ are efficiently computable. An upgrade of Elgamal in the exponent is introduced in [44]. First, it is IND-CCA, and last, it enables the public sharing of large secrets in the form of several efficiently decryptable small "chunks" (along with efficient NIZK proofs of smallness), such that non-interactive additions on secrets are still enabled.

7.2 Paillier mod p. Paillier encryption does not support unlimited homomorphic additions modulo a fixed p, since the plaintext space is instead a $\mathbb{Z}/N_i\mathbb{Z}$ for a composite N_i, which is furthermore unavoidably different for each party P_i calling KeyGen. The observation which we make is that, if the parameters of KeyGen are chosen such that every N_i returned is at least of size: $(t{+}1)p^2 \leq N_i/2$, then Paillier satisfies Definition 2, i.e., perfect correctness $\bmod p$ after *one* homomorphic addition of size $t{+}1$. See Appendix F.3 for efficient NIZKs.

7.3 Lattice-based schemes. A variant of Regev over $\mathbb{Z}/p\mathbb{Z}$ is provided in [9, §2.1]. They sketch bounds denoted M and R which, concretely, imply correctness $\bmod p$ after one homomorphic linear combination of size $t{+}1$, as soon as $(t{+}1)p^2 \leq M$ and $(t{+}1)p \leq R$. To enforce *perfect* correctness $\bmod p$, one should furthermore restrict to a finite interval $\mathscr{R}_{\text{key}}$ the randomness of the key generation, concretely, by cutting-off the queues of the Gaussian. The same cut-off should be done in other lattice-based schemes such as BFV/BGV [50]. See Appendix F.3 for efficient NIZKs.

7.4 Linear combinations / multiexponentiations of secrets over rings / groups. The baseline method since [8] to securely open a linear combination of shared secrets, is that parties locally evaluate the linear combination of the shares then send it to the learner $\mathcal{L}$. To enable this method in Y-VSS, the local lists of parties are indexed by the collector from which they received the PPS. Then, to open a linear combination of secrets, each party in $\mathcal{C}^{(e_o)}$, for each $K_k^{(e_o)} \in \mathcal{C}_{\text{collec}}^{(e_o)}$, computes the homomorphic linear combination of the PPSs of these secrets which it received from $K_k \in \mathcal{C}_{\text{collec}}^{(e_o)}$. Then it sends the decryption to $\mathcal{L}$, appended with the PPSs (all at once for all $K_k^{(e_o)}$). Liveness follows from the fact that there is at least a honest $K_k^{(e_o)} \in \mathcal{C}_{\text{collec}}^{(e_o)}$ which sends the same PPSs to all parties in $\mathcal{C}^{(e_o)}$.

We make the observation that the baseline extends to opening the image by a group homomorphism φ, e.g., a multi-exponentiation of r public group elements by r secrets in $\mathbb{F}_p$. The idea is that parties decrypt their shares of the PPSs, evaluate the homomorphism φ on the shares, then send to $\mathcal{L}$ the image, along with a NIZK of correct decryption-then-evaluation. But to make this idea work, we need linear secret sharing schemes both in the source and in the target spaces of φ, which commute with φ. Namely, which are such that threshold opening-then-φ equals φ on the shares-then-threshold opening. For instance when the target is an abelian group $\mathbb{G}$ of order p, the secret sharing is the variant of Shamir's scheme, using polynomials over $\mathbb{G}$ ([4, §6]). This is precisely what we do in our implementation in Section 6. See also [3] for more formalism. In Appendix F.2 we give other examples along with suitable secret sharing schemes.

7.5 Comparison and details on related works. In Table 6 we compare to existing PSS schemes which have both security and liveness, i.e., termination, up to a corrupt minority $t \leq n/2$, as Y-VSS. We do not further compare with schemes which are not both under this threshold [71, 13, 63, 31, 17, 3, 69, 67, 70], including "Opt-CHURP" [56]. Notice that UC security encompasses both correctness and what related works name as "privacy". Notice that related works call "robustness" the combination of correctness and liveness. All the PSS of Table 6 assume a synchronous broadcast channel which would deliver to all parties the messages posted, within a worst-case latency which we denote as Δ_{BC}. When this assumption fails, then the security of these PSS is not guaranteed anymore. For example, when the broadcast from an honest P_i is not received within Δ_{BC}, then Exp-CHURP-A [56, p. C.1.3] forces honest parties to publicly expose what they sent to P_i, which provides enough information to the adversary to reconstruct the stored secret. Likewise in [43, p8], when a sending or broadcast from some honest P_i is not received in time by an honest P_j, then P_j accuses P_i, which must then publicly expose the content of this message, which provides substantial information to the adversary on the stored secret. See also [5] for a related issue, when not receiving an accusation in time. Hence, the performances of Y-VSS cannot totally be compared with the ones of any existing scheme, in that it is the first not to base neither its security nor liveness on such Δ_{BC} assumption.

References

[1] M. Abspoel, R. Cramer, I. Damgård, D. Escudero, and C. Yuan. "Efficient Information-Theoretic Secure Multiparty Computation over Z/p^{k}Z via Galois Rings". In: *TCC*. 2019.

[2] Amis. *implementation of CMP MPC with echo broadcast.* https://github.com/getamis/alice/tree/master/crypto/tss/ecdsa/cggmp. 2022.

[3] D. F. Aranha, A. Dalskov, D. Escudero, and C. Orlandi. "Improved Threshold Signatures, Proactive Secret Sharing, and Input Certification from LSS Isomorphisms". In: *LATINCRYPT*. 2021.

Scheme	Network	Latency [(2)]	Communication [(1)]	Dynamic	Yoso	Setup
PSS [48]	Synch	$\delta_{\text{PKI}} + 3\Delta_{\text{BC}}$	$n\lvert\text{BC}(n)\rvert$	No	No	PKI
pvR [10], Groth [44], GHL [39]	Synch	$\delta_{\text{PKI}} + \Delta_{\text{BC}}$	$n\lvert\text{BC}(n)\rvert$	Yes	Yes	PKI & URS [(4)]
Exp-CHURP-A[56, §C.1]	Synch	$\delta_{\text{PKI}} + 5\Delta_{\text{BC}}$	$n\lvert\text{BC}(n)\rvert$	Yes	No	PKI & SRS [(5)]
DPSS [43]	Synch	$5\Delta_{\text{BC}}$	$\lvert\text{BC}(n)\rvert$ [(3)]	Yes	No	SRS [(5)]
Y-VSS	Asynch	$\boldsymbol{\delta_{\text{PKI}} + 2\delta}$	$n\lvert\text{MC}(n^2)\rvert$	Yes	Yes	PKI [(4)]

(1) The *communication complexity* is the total number of bits sent by honest parties in both the exiting and entering committee to complete a Refresh. $n\lvert\text{BC}(n)\rvert$ resp. $n\lvert\text{MC}(n^2)\rvert$ denote the bitsize needed to implement n broadcasts resp. n multicasts, each with input size $O(n)$ bits resp. $O(n^2)$ bits. As reported in [64], the state of the art communication complexity for broadcast in expected constant rounds under tight honest majority and no common coin setup, is still the one of [49], in $\lvert\text{BC}(1)\rvert = O(\kappa n^4)$. Known amortization techniques would improve this into $\lvert\text{BC}(\ell)\rvert = O(\ell n^3 + n^4\kappa)$ asymptotically in ℓ.

(2) 2δ denotes 2 consecutive message deliveries, 5BC denotes 5 consecutive broadcasts and δ_{PKI} is the publication delay on the PKI.

(3) The $\lvert\text{BC}(n)\rvert$ complexity of [43] is in the amortized regime of sharing more than n secrets

(4) URS designates a public uniform random string, possibly known long before players publish their keys. Uniformity allows generation by nothing-up-my-sleeve sampling or distributed beacons [20]. A URS is required for the NIZKs in both [10, 39]. A URS is also required in [44, 39] as a *common parameter for the public keys* (the scheme being CCA in the former), enabling to amortize the sizes of ciphertexts. Y-VSS does not require a URS in at least two cases: in our implemented instantiation with Elgamal (for secrets which are group elements), and in general, when NIZKs are built from [45], using a PKI setup.

(5) SRS designates a structured random string, needed for KZG. This assumption could be downgraded in [43, 56] if specifying another polynomial commitment. Exp-CHURP-A [56] has also another potential source of setup, due to black-box NIZKs (Fig. 15 line 7, Fig. 16 line 5).

Table 6: Comparison of Refreshs of proactive secret sharing schemes under honest majority: $t < n/2$.

[4] T. Attema, R. Cramer, and M. Rambaud. "Compressed Σ-Protocols for Bilinear Group Arithmetic Circuits and Application to Logarithmic Transparent Threshold Signatures". In: *ASIACRYPT*. 2021.

[5] J.-P. Aumasson and O. Shlomovits. *Attacking Threshold Wallets*. Eprint 2020/1052. 2020.

[6] M. Backes, A. Kate, and A. Patra. "Computational Verifiable Secret Sharing Revisited". In: *ASIACRYPT*. 2011.

[7] P. S. Barreto and M. Naehrig. "Pairing-friendly elliptic curves of prime order". In: *SAC*. 2005.

[8] J. C. Benaloh. "Secret Sharing Homomorphisms: Keeping Shares of a Secret Secret (Extended Abstract)". In: *CRYPTO*. 1986.

[9] R. Bendlin, I. Damgård, C. Orlandi, and S. Zakarias. "Semi-homomorphic Encryption and Multiparty Computation". In: *EUROCRYPT*. 2011.

[10] F. Benhamouda et al. "Can a Public Blockchain Keep a Secret?" In: *TCC*. 2020.

[11] D. Boneh and V. Shoup. *A Graduate Course in Applied Cryptography*. Version 0.5 Jan 2020. 2020.

[12] G. Botrel, T. Piellard, Y. E. Housni, A. Tabaie, and I. Kubjas. *ConsenSys/gnark-crypto: v0.6.1*. 2022.

[13] C. Cachin, K. Kursawe, A. Lysyanskaya, and R. Strobl. "Asynchronous Verifiable Secret Sharing and Proactive Cryptosystems". In: *ACM CCS*. 2002.

[14] M. Campanelli, B. David, H. Khoshakhlagh, A. Konring, and J. B. Nielsen. "Encryption to the Future: A Paradigm for Sending Secret Messages to Future (Anonymous) Committees". In: *ASIACRYPT*. 2022.

[15] R. Canetti. "Universally composable security: A New Paradigm for Cryptographic Protocols". In: *FOCS*. 2001.

[16] R. Canetti. "Universally composable signature, certification, and authentication". In: *IEEE CSF Workshop, 2004.* corrected version of August 15. 2004.

[17] R. Canetti, R. Gennaro, S. Goldfeder, N. Makriyannis, and U. Peled. "UC Non-Interactive, Proactive, Threshold ECDSA with Identifiable Aborts". In: *CCS*. 2020.

[18] R. Canetti, N. Makriyannis, and U. Peled. *UC Non-Interactive, Proactive, Threshold ECDSA*. ePrint 2020/492, merged into CCS'20. 2020.

[19] R. Canetti, D. Shahaf, and M. Vald. "Universally composable authentication and key-exchange with global PKI". In: *PKC*. 2016.

[20] I. Cascudo and B. David. "ALBATROSS: publicly AttestabLe BATched Randomness based On Secret Sharing". In: *ASIACRYPT*. 2020.

[21] I. Cascudo, B. David, L. Garms, and A. Konring. "YOLO YOSO: Fast and Simple Encryption and Secret Sharing". In: *ASIACRYPT*. 2022.

[22] G. Castagnos and F. Laguillaumie. "Linearly Homomorphic Encryption from DDH". In: *CT RSA*. 2015.

[23] G. Castagnos, F. Laguillaumie, and I. Tucker. "Threshold Linearly Homomorphic Encryption on $Z/2^kZ$". In: *ASIACRYPT*. 2022.

[24] J. Chen and S. Micali. "Algorand: A Secure and Efficient Distributed Ledger". In: *Theor. Comput. Sci.* (2019).

[25] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. "Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults (Extended Abstract)". In: *FOCS*. 1985.

[26] Coinbase. *announcing CMP MPC with echo broadcast.* https://www.coinbase.com/blog/hierarchical-threshold-signature-scheme-an-approach-to-distinguish-singers-in-threshold?source=rss----c114225aeaf7---4. 2022.

[27] S. Coretti, A. Kiayias, C. Moore, and A. Russell. "The Generals' Scuttlebutt: Byzantine-Resilient Gossip Protocols". In: *CCS*. 2022.

[28] R. Cramer, I. B. Damgård, and J. B. Nielsen. *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, 2015.

[29] Y. Desmedt and S. Jajodia. *Redistributing secret shares to new access structures and its applications*. Tech Report, George Mason U. July 1997.

[30] Y. G. Desmedt and Y. Frankel. "Homomorphic Zero-Knowledge Threshold Schemes over any Finite Abelian Group". In: *SIAM Journal on Discrete Mathematics* (1994).

[31] Y. Desmedt and K. Morozov. "Parity Check based redistribution of secret shares". In: *ISIT*. 2015.

[32] C. Dwork, N. Lynch, and L. Stockmeyer. "Consensus in the Presence of Partial Synchrony". In: *J. ACM* (1988).

[33] J. Fan and F. Vercauteren. "Somewhat Practical Fully Homomorphic Encryption". In: *IACR ePrint* (2012).

[34] S. Fehr. "Span Programs over Rings and How to Share a Secret from a Module". MA thesis. ETH Zurich, 1998.

[35] Fireblocks. *"MPC key shares are automatically refreshed in minutes-long intervals"*. https://www.fireblocks.com/what-is-mpc/. 2022.

[36] M. Fitzi and J. B. Nielsen. "On the Number of Synchronous Rounds Sufficient for Authenticated Byzantine Agreement". In: *DISC*. 2009.

[37] R. Gennaro, M. O. Rabin, and T. Rabin. "Simplified VSS and Fast-Track Multiparty Computations with Applications to Threshold Cryptography". In: *PODC*. 1998.

[38] C. Gentry, S. Halevi, B. Magri, J. B. Nielsen, and S. Yakoubov. "Random-Index PIR and Applications". In: *TCC*. 2021.

[39] C. Gentry and S. H. and Vadim Lyubashevsky. "Practical Non-interactive PVSS with Thousands of Parties". In: *EUROCRYPT*. 2022.

[40] C. Gentry et al. "YOSO: You Only Speak Once: Secure MPC with Stateless Ephemeral Roles". In: *CRYPTO*. 2021.

[41] O. Goldreich, S. Micali, and A. Wigderson. "Proofs That Yield Nothing but Their Validity or all languages in NP have zero-knowledge proof systems". In: *J. ACM* (1991).

[42] S. Goldwasser and Y. Lindell. "Secure Computation without Agreement". In: *DISC*. 2002.

[43] V. Goyal, A. Kothapalli, E. Masserova, B. Parno, and Y. Song. "Storing and Retrieving Secrets on a Blockchain". In: *PKC*. 2022.

[44] J. Groth. *Non-interactive distributed key generation and key resharing*. ePrint 2021/339. 2021.

[45] J. Groth and R. Ostrovsky. "Cryptography in the Multi-string Model". In: *CRYPTO*. 2007.

[46] V. Gupta and K. Gopinath. "An extended verifiable secret redistribution protocol for archival systems". In: *ARES*. 2006.

[47] S. Halevi, H. Krawczyk, A. Miao, and T. Rabin. "Threshold Cryptography as a Service (in the Multi-server and YOSO Models)". In: *CCS*. 2022.

[48] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung. "Proactive Secret Sharing". In: *CRYPTO*. 1995.

[49] J. Katz and C.-Y. Koo. "On Expected Constant-Round Protocols for Byzantine Agreement". In: (2009).

[50] A. Kim, Y. Polyakov, and V. Zucca. "Revisiting Homomorphic Encryption Schemes for Finite Fields". In: *ASIACRYPT*. 2021.

[51] E. Kim et al. "How to Securely Collaborate on Data: Decentralized Threshold HE and Secure Key Update". In: *IEEE Access* (2020).

[52] L. Lamport. "Lower Bounds for Asynchronous Consensus". In: *Distrib. Comput.* (2006).

[53] Y. Lindell, A. Nof, and S. Ranellucci. "Fast Secure Multiparty ECDSA with Practical Distributed Key Generation". In: *CCS*. 2018.

[54] C.-D. Liu-Zhang, J. Loss, U. Maurer, T. Moran, and D. Tschudi. "MPC with Synchronous Security and Asynchronous Responsiveness". In: *ASIACRYPT*. 2020.

[55] V. Lyubashevsky, N. K. Nguyen, and M. Plancon. "Lattice-Based Zero-Knowledge Proofs and Applications: Shorter, Simpler, and More General". In: *CRYPTO*. 2022.

[56] S. K. D. Maram et al. "Churp: Dynamic-committee proactive secret sharing". In: *ACM CCS*. 2019.

[57] A. U. and Matthieu Rambaud. *Share & Shrink: Ad-Hoc Threshold FHE with Short Ciphertexts and its Application to Almost-Asynchronous MPC*. ePrint 2022/378. 2022.

[58] R. Ostrovsky and M. Yung. "How to Withstand Mobile Virus Attacks". In: *PODC*. 1991.

[59] R. Pass and E. Shi. "Thunderella: Blockchains with Optimistic Instant Confirmation". In: *EUROCRYPT*. 2018.

[60] M. Rambaud and A. Urban. Iacr ePrint 2021/503. 2021.

[61] T. Ristenpart and S. Yilek. "The Power of Proofs-of-Possession: Securing Multiparty Signatures against Rogue-Key Attacks". In: *EUROCRYPT*. 2007.

[62] B. Schoenmakers. "A Simple Publicly Verifiable Secret Sharing Scheme and its Application to Electronic Voting". In: *CRYPTO*. 1999.

[63] D. Schultz, B. Liskov, and M. Liskov. "MPSS: Mobile Proactive Secret Sharing". In: *ACM Trans. Inf. Syst. Secur.* (2010).

[64] N. Shrestha, A. Bhat, A. Kate, and K. Nayak. "Synchronous Distributed Key Generation without Broadcasts v1". In: *ePrint 2021/1635 v1 of 2021-12-17* (2021).

[65] M. Stadler. "Publicly Verifiable Secret Sharing". In: *EUROCRYPT*. 1996.

[66] A. Takahashi and G. Zaverucha. *Verifiable Encryption from MPC-in-the-Head*. eprint 2021/1704. 2021.

[67] R. Vassantlal, E. Alchieri, B. Ferreira, and A. Bessani. "COBRA: Dynamic Proactive Secret Sharing for Confidential BFT Services". In: *IEEE S&P*. 2022.

[68] T. M. Wong, C. Wang, and J. M. Wing. “Verifiable Secret Redistribution for Archive Systems”. In: *IEEE SISW*. 2002.

[69] Y. Y. and Yu Xia and Srinivas Devadas. *Shanrang: Fully Asynchronous Proactive Secret Sharing with Dynamic Committees*. ePrint 2022/164. 2022.

[70] T. Yurek, Z. Xiang, Y. Xia, and A. Miller. *Long Live The Honey Badger: Robust Asynchronous DPSS and its Applications*. ePrint 2022/971. 2022.

[71] L. Zhou, F. B. Schneider, and R. Van Renesse. “APSS: Proactive secret sharing in asynchronous systems”. In: *ACM TISSEC* (2005).

Appendix

A Complements on the model

A.1 Ideal functionalities $\mathcal{F}_{\text{CA}}$ and $\mathcal{F}_{\text{NIZK}}$. We present in Algorithm 7 the ideal functionality of a bulletin board of public keys, defined in §Section 2.8 as $\mathcal{F}_{\text{CA}}$.

$\mathcal{F}_{\text{CA}}$

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.

Algorithm 7: The certification authority functionality, $\mathcal{F}_{\text{CA}}$.

We present in Algorithm 8 the ideal functionality of *non-interactive zero-knowledge arguments of knowledge*. $\mathcal{F}_{\text{NIZK}}$ is parametrized by a NP relation $\mathcal{R}$. 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.

A.2 Full modelisation of Y-VSS in the UC framework. We consider a PPT machine denoted as the *environment* $\mathcal{E}$. $\mathcal{A}$ is fully controlled by $\mathcal{E}$. In particular, $\mathcal{A}$ forwards to $\mathcal{E}$ all its incoming signals. Such $\mathcal{A}$ is known as “dummy” [15]. Let us recall from [15] that this restriction on $\mathcal{A}$ is enough to prove a protocol UC-secure. $\mathcal{E}$ assigns its input secret $s \in \mathbb{F}_p$ to the dealer. When the learner $\mathcal{L}$ outputs some value $s_{\text{out}} \in \mathbb{F}_p$, then it immediately informs $\mathcal{E}$ of s_{out}.

Following the model [15], we say that a protocol Π UC-implements $\mathcal{F}_{\text{VSS}}$ if there exists a PPT machine $\mathcal{S}$ denoted *simulator*, also known as “ideal adversary”, such that every PPT $\mathcal{E}$ has negligible advantage in distinguishing between the following two executions:

$\mathcal{F}_{\text{NIZK}}$

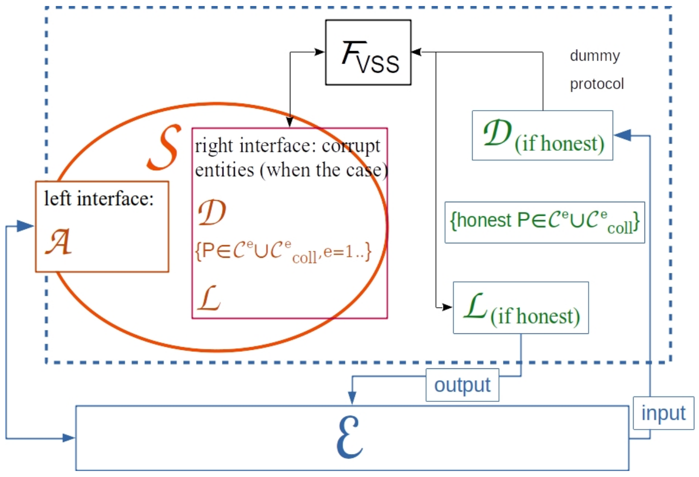
The functionality is parameterized with an NP relation R of an NP language L and a prover P.

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

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

Algorithm 8: Non-interactive zero-knowledge functionality

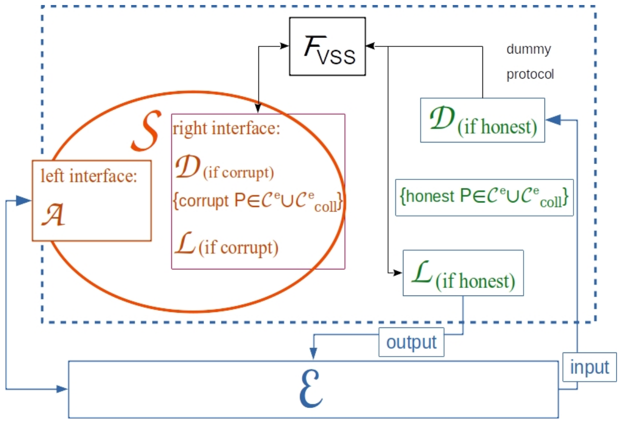
- REAL$_{\mathcal{A}}$: an actual execution of the protocol Π, with adversary $\mathcal{A}$ fully controlled by $\mathcal{E}$, and functionalities $\mathcal{F}_{\text{CA}}$, $\mathcal{F}_{\text{Sch}}$, $\mathcal{F}_{\text{BC}}$, $\mathcal{F}_{\text{ST}}$, $\mathcal{F}_{\text{AT}}$, $\mathcal{F}_{\text{NIZK}}$, as depicted in Figure 9;
- IDEAL$_{\mathcal{F}_{\text{VSS}}, \mathcal{S}}$: an execution denoted as *ideal*, where $\mathcal{S}$ interacts with $\mathcal{E}$ on behalf of $\mathcal{A}$. In Figure 10 we denote this as the “left interface”. On the other side, $\mathcal{S}$ interacts with $\mathcal{F}_{\text{VSS}}$ on behalf of the corrupt entities and also of $\mathcal{A}$, which we dub as its “right interface”. The honest entities are connected to $\mathcal{E}$ as in a real execution (namely: $\mathcal{D}$ and/or $\mathcal{L}$ if honest). But on the other side, they only interact with $\mathcal{F}_{\text{VSS}}$. Concretely, they perform what is commonly called as the *dummy* protocol, which consists in $\mathcal{D}$ (if honest) gives its input to $\mathcal{F}_{\text{VSS}}$, then $\mathcal{L}$ (if honest) outputs what it receives from $\mathcal{F}_{\text{VSS}}$, while parties on their side do nothing.



$\mathcal{F}_{\text{VSS}}$
dummy
protocol
$\mathcal{S}$
right interface: corrupt entities (when the case)
$\mathcal{D}$
$\{P \in \mathcal{C}^e \cup \mathcal{C}^e_{\text{coll}}, e=1..\}$
$\mathcal{L}$
left interface:
$\mathcal{A}$
$\mathcal{D}$ (if honest)
$\{\text{honest } P \in \mathcal{C}^e \cup \mathcal{C}^e_{\text{coll}}\}$
$\mathcal{L}$ (if honest)
output
$\mathcal{E}$
input

Figure 9: $REAL_{\mathcal{A}}$ execution of Π with dummy adversary $\mathcal{A}$. The Environment $\mathcal{E}$ has no other interface with the system (the big rectangle in dotted blue) than: its full control on $\mathcal{A}$, its power to give the input to $\mathcal{D}$ if it is honest, and its power to learn the output of $\mathcal{L}$ if it is honest.

A.3 Comments on the model and on other models.



$\mathcal{F}_{\text{VSS}}$
dummy
protocol
$\mathcal{S}$
right interface:
$\mathcal{D}$ (if corrupt)
{corrupt $P \in \mathcal{C}^e \cup \mathcal{C}^e_{\text{coll}}$}
$\mathcal{L}$ (if corrupt)
left interface:
$\mathcal{A}$
$\mathcal{D}$ (if honest)
{honest $P \in \mathcal{C}^e \cup \mathcal{C}^e_{\text{coll}}$}
$\mathcal{L}$ (if honest)
output
input
$\mathcal{E}$

Figure 10: $IDEAL_{\mathcal{F}_{\text{VSS}},\mathcal{S}}$ execution: honest entities perform the dummy protocol with $\mathcal{F}_{\text{VSS}}$; the simulator $\mathcal{S}$ interacts on the right with $\mathcal{F}_{\text{VSS}}$ with the same interface as $\mathcal{A}$ and corrupt entities in the dummy protocol, and on the left, interacts with $\mathcal{E}$ with the same interface as the dummy adversary in the real protocol.

Our model of proactivity We do not capture proactivity at the level of the functionality $\mathcal{F}_{\text{VSS}}$. We instead capture it at the level of the protocol, by giving the power of the adversary to fully control the scheduler $\mathcal{F}_{\text{Sch}}$ which sends the signals startsig and refreshsig, in Section 2.7. Notice that in [13], the signal refreshsig is sent by an external global clock. Our model is at least as strong, since security holds whatever the choices of $\mathcal{A}$ of when and to whom to send the signals.

An alternative model of proactivity By contrast, in the later work [70, B], proactivity is captured at the level of the funtionality. Their functionality is indeed specified to return to parties one unique consistent set of commitments to re-sharings. So is not implementable under asynchrony and honest majority.

Extension to non-"yoso" refreshes Our model generalizes to protocols in which parties are allow to speak more than once, as follows. After it sent a refreshsig to some party P, the scheduler can shut-it-off after all chains of ℓ consecutive asynchronous events completed, in a sense which can be be made precise from [52]. There, ℓ captures the number of interactions needed to complete a Refresh.

B Details on Publicly Verifiable (Re-)Sharing

In Appendix B.3 we introduce publicly verifiable secret sharing pvShare and resharing pvReshare, we formalize the data structures output, which are publicly verifiable sharing (pvS) and resharing (pvR), then state their properties.

B.1 Shamir resharing. Consider any vector of shares $(s_i)_{i\in[n]}$ of some s. Consider any $t+1$-subset $\mathcal{U} \subset [n]$ and, for each $i \in \mathcal{U}$, any vector of shares $s_{i\to[n]} = (s_{i\to j})_{j\in[n]}$ of s_i. We dub the latter as a *re-sharing of* s_i, or also as *sub-shares of* s_i. Then the vector, dubbed as *new* or *refreshed shares*:

(5) $s'_{[n]} := \mathcal{L}\text{-combine}^{\mathcal{U}}\big((s_{i\to[n]})_{i\in\mathcal{U}}\big)$ is a sharing of s,

where, concretely, $s'_{[n]} := \big(s'_j := \sum_{i\in\mathcal{U}} \lambda_i^{\mathcal{U}} s_{i\to j}\big)_{j\in[n]}$. The proof is because, consider the (re-)sharing polynomials: $(f_i(X))_{i\in\mathcal{U}}$ defining the $(s_{i\to[n]})_{i\in\mathcal{U}}$, then, since polynomial evaluation commutes with linear combinations, the new shares $(s'_j)_{j\in[n]}$ are the evaluations of $f'(X) := \sum_{i\in\mathcal{U}} \lambda_i^{\mathcal{U}} f_i(X)$, but by construction $f'(0) = s$. An illustration is provided in Figure 1.

B.2 Encryption scheme. We specify the public key encryption scheme needed, called PKE, as the following list of spaces, efficiently computable algorithms and properties: $s\mathcal{K}$ and $p\mathcal{K}$ the *spaces of secret and public keys*, $\mathcal{R}_{\text{key}}$ the *set of key randomness*, $\mathcal{R}_{\text{enc}}$ the *set of encryption randomness*, the *plaintext space* $\mathbb{Z}$, $\mathcal{C}$ the *ciphertext space*;

- **KeyGen**$(\,|\,\rho_{\text{key}} \in \mathcal{R}_{\text{key}}) \to s\mathcal{K} \times p\mathcal{K}$ the *key generation function*;
- **Enc**$(\mathsf{pk} \in p\mathcal{K}, x \in \mathbb{Z}\,|\,\rho_{\text{enc}} \in \mathcal{R}_{\text{enc}}) \to \mathcal{C}$ the *encryption function*;
- **Dec**$(\mathsf{sk} \in s\mathcal{K}, c \in \mathcal{C}) \to \mathbb{F}_p$ the *decryption* $\bmod p$ *function*;
- $\boxplus : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$ and $\boxdot : \mathbb{F}_p \times \mathcal{C} \to \mathcal{C}$ the *linearly homomorphic addition and scalar multiplication.*

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

For $c \in \mathcal{C}$, 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 $\rho_{\text{enc}} \in \mathcal{R}_{\text{enc}}$ such that $c = \mathsf{Enc}(\mathsf{pk}, x|\rho_{\text{enc}})$.

Definition 8. *Perfect correctness* $\bmod p$ *after one homomorphic 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 $(c_i)_{i\in[t+1]} \in \mathcal{C}^{t+1}$, for any explainable plaintexts $(\widetilde{x}_i)_{i\in[t+1]} \in \mathbb{F}_p^{t+1}$ of them under pk, for any $(\lambda_i)_{i\in[t+1]} \in \mathbb{F}_p^{t+1}$, then, for any explainable decryption $\widetilde{y}$ under pk of $\boxplus_{i\in[t+1]}(\lambda_i \boxdot c_i)$, we have $\widetilde{y} = \sum_{i=1}^{t+1} \lambda_i \widetilde{x}_i$.

Notice that $\widetilde{y}$ can exist only if there exists an explainable secret key for pk. Definition 8 implies a property which may be denoted as *decryptor-binding* or *unicity*: *if* such a $\widetilde{y}$ exists, then it is uniquely determined by the $(c_i)_{i\in[t+1]} \in \mathcal{C}^{t+1}$.

Let us enrich decryption with public verifiability.

- **pvDec**$(\mathsf{pk} \in p\mathcal{K}, \mathsf{sk} \in s\mathcal{K}, c \in \mathcal{C}) \to (\mathbb{F}_p, \Pi)$ the *publicly verifiable decryption algorithm.* It outputs $x := \mathsf{Dec}(\mathsf{sk}, c)$ along with a NIZK AoK, denoted as π_{pvD}, of a secret key sk of pk, such that $x = \mathsf{Dec}(\mathsf{sk}, c)$. We denote such π_{pvD} as a proof as a "*proof of correct decryption of* c *into* x *under key* pk.

We now give the name to the data structure output by pvDec.

- a $\mathsf{pvD}\big(\mathsf{pk}\in\mathcal{pK},c\in\mathcal{C}\big)$, called *publicly verifiable decryption of c under key* pk, is a pair $(x\in\mathbb{F}_p,\pi_{\mathsf{pvD}}\in\Pi)$ where π_{pvD} is a proof of correct decryption of c into x, under key pk.

B.3 Publicly verifiable secret sharing (pvS) and resharing (pvR). We consider as a public parameter a list of public keys $(\mathsf{pk}'_i)_{i\in[n]}\in(\mathcal{pK}\sqcup\bot)^n$, which in our use-case will be the ones of the entering committee $\mathcal{C}'$. By convention, encryption under $\bot$ equals $\bot$. pvShare, on input $s\in\mathbb{F}_p$, returns a vector of Shamir shares encrypted under the public keys. pvReshare, on input a ciphertext $c\in\mathcal{C}$, decrypts it with the secret key sk of the re-sharer, then proceeds as in pvShare. Both algorithms also return proofs of correctness, denoted π_{pvS} and π_{pvR} and defined below.

- $\mathsf{pvShare}\Big(s\in\mathbb{F}_p\,|\,\boldsymbol{\rho}_{\mathbf{enc}}\in\mathcal{R}^n_{\mathrm{enc}},Q\in\mathbb{F}_p[X]^{(0)}_t\Big)\rightarrow\quad(\mathcal{C}^n,\Pi)$
Set $s_i:=(s+Q)(i)\ \forall i\in[n]$, output
$c_{[n]}:=\Big(\mathsf{Enc}\big(\mathsf{pk}'_i,s_i|\rho_{\mathrm{enc},i}\big)\Big)_{i\in[n]}$ and π_{pvS}.
- $\mathsf{pvReshare}\Big(\mathsf{pk}\in\mathcal{pK},\mathsf{sk}\in\mathcal{sK},c\in\mathcal{C}\,|\boldsymbol{\rho}_{\mathbf{enc}}\in\mathcal{R}^n_{\mathrm{enc}},$
$Q\in\mathbb{F}_p[X]^{(0)}_t\Big)\rightarrow\quad(\mathcal{C}^n,\Pi)$: set $s:=\mathsf{Dec}(\mathsf{sk},c)$, output $c_{[n]}:=\mathsf{pvShare}\big(s,|\boldsymbol{\rho}_{\mathbf{enc}},Q\big)$ and π_{pvR}.

π_{pvS} is what we call as a *proof of plaintext sharing knowledge*, it is a NIZK AoK of a $\widetilde{s}\in\mathbb{F}_p$, a $\widetilde{\boldsymbol{\rho}_{\mathbf{enc}}}\in\mathcal{R}^n_{\mathrm{enc}}$ and a $\widetilde{Q}\in\mathbb{F}_p[X]^{(0)}_t$, such that $c'_{[n]}=\mathsf{pvShare}\big(\widetilde{s}\,|\,\widetilde{\boldsymbol{\rho}_{\mathbf{enc}}},\widetilde{Q}\big)$.

π_{pvR} is what we define as a *proof of plaintext re-sharing knowledge of c into $c'_{[n]}$, under key* pk it is a NIZK AoK of an explainable decryption $\widetilde{s}$ of c under pk, a $\widetilde{\boldsymbol{\rho}_{\mathbf{enc}}}$ and a $\widetilde{Q}\in\mathbb{F}_p[X]^{(0)}_t$, denoted as an *explainable re-sharing polynomial* such that $c'_{[n]}=\mathsf{pvReshare}(\mathsf{pk},\widetilde{\mathsf{sk}},c\,|\,\widetilde{\rho_{\mathrm{enc}}},\widetilde{Q})$. The evaluations $s_j:=(\widetilde{Q}+s)(j)$ are denoted as *explainable (plaintext) resharing shares*.

- a **pvS**, called as a *public verifiable secret sharing*, is a pair $(c_{[n]}\in\mathcal{C}^n,\pi_{\mathsf{pvS}}\in\Pi)$ where π_{pvS} is a proof of plaintext secret knowledge for $c_{[n]}$.
- a $\mathbf{pvR}\big(\mathsf{pk}\in\mathcal{pK},c\in\mathcal{C}\big)$, called as a *publicly verifiable resharing of c from* pk, is a pair $(c'_{[n]}\in\mathcal{C}^n,\pi_{\mathsf{pvR}}\in\Pi)$ where π_{pvR} is proof of plaintext re-sharing of c, into $c'_{[n]}$, under key pk.

Notice that a pvR proves in particular knowledge of a plaintext secret, thus is a fortiori a pvS. Although pvRs are not signed, we will make constant use that they publicly determine their issuer, identified with its public key pk. The reason is that the NIZK appended to the pvR proves knowledge of the secret key sk associated to the public key pk.

B.4 IND-CPA of publicly verifiable (re)-sharing. The next Proposition 9 states that any PPT adversary $\mathcal{A}$ has negligible advantage in distinguishing between a vector of encrypted shares of any chosen secret, and a sample in some fixed distribution. This distribution, which is formalized as "$\mathcal{D}_0$" below, is the one of vectors of ciphertexts of the form $(\mathsf{Enc}_{\mathsf{pk}_i}(s_i))_{i\in[n]}$, where the t plaintexts $(s_i)_{i\in\mathcal{I}}$ with coordinates of corrupt parties vary uniformly at random in $\mathbb{F}^t_p$, while the $t+1$ other $(s_i)_{i\in[n]\setminus\mathcal{I}}$ are all equal to 0.

Proposition 9 (IND-CPA of encrypted sharing)**.** *For any threshold $1\leq t$, any PPT machine $\mathcal{A}$ has negligible advantage in the following game with an oracle $\mathbb{O}$. $\mathcal{A}$ gives to $\mathbb{O}$: a subset of t indices $\mathcal{I}\subset[n]$, and a list of t public keys $(\mathsf{pk}_i)_{i\in\mathcal{I}}\in(\mathcal{pK}\sqcup\bot)^t$. $\mathbb{O}$ generates $(\dots,\mathsf{pk}_i)\leftarrow\mathsf{KeyGen}()$ for $i\in[n]\backslash\mathcal{I}$ which it shows to $\mathcal{A}$. $\mathbb{O}$ tosses $b\xleftarrow{\$}\{0,1\}$. Then $\mathcal{A}$ is allows to query $\mathbb{O}$ an unlimited number of times as follows. $\mathcal{A}$ gives to $\mathbb{O}$ any $s\in\mathbb{F}_p$ of its choice, then $\mathbb{O}$ with:*
if $b=1$: encrypted sharing *generates $(s_i)_{i\in[n]}:=\mathsf{Share}(s)$, returns $(\mathsf{Enc}_{\mathsf{pk}_i}(s_i))_{i\in[n]}$;*
if $b=0$: $\mathcal{D}_0$ *samples $s_i\xleftarrow{\$}\mathbb{F}_p$ $\forall i\in\mathcal{I}$, sets $s_i:=0$ $\forall i\in[n]\setminus\mathcal{I}$, returns $(\mathsf{Enc}_{\mathsf{pk}_i}(s_i))_{i\in[n]}$.*
At some point $\mathcal{A}$ outputs b' and wins if $b{=}b'$.

For technical reasons, in the proof of Y-VSS (Section 5.1, games $\mathsf{Hyb}^{0\mathsf{Refresh}}[e,i]$) we will actually need the following slightly stronger version. We state it as Proposition 10 below, and note that it directly implies Proposition 9. The slight difference with Proposition 9 is that the oracle now give directly in the clear its t shares to the adversary. This gives strictly more power to $\mathcal{A}$ than in Proposition 9, in the case where $\mathcal{A}$ would have badly generated some of its t public keys.

Proposition 10 (IND-CPA of pvS with plaintext adversary shares)**.** *For any threshold $1\leq t$, any PPT machine $\mathcal{A}$ has negligible advantage in the following game with an oracle $\mathbb{O}$. $\mathcal{A}$ gives to $\mathbb{O}$ a subset of t indices $\mathcal{I}\subset[n]$. $\mathbb{O}$ generates $(\dots,\mathsf{pk}_i)\leftarrow\mathsf{KeyGen}()$ for $i\in[n]\backslash\mathcal{I}$ which it shows to $\mathcal{A}$. $\mathbb{O}$ tosses $b\xleftarrow{\$}\{0,1\}$. Then $\mathcal{A}$ is allowed to query $\mathbb{O}$ an unlimited number of times as follows. $\mathcal{A}$ queries $\mathbb{O}$ with any $s\in\mathbb{F}_p$ of its choice, then $\mathbb{O}$ responds as:*
if $b=1$: enc. sharing + $\mathcal{A}$'s shares *generates $(s_i)_{i\in[n]}:=\mathsf{Share}(s)$, returns $(\mathsf{Enc}_{\mathsf{pk}_i}(s_i))_{i\in[n]\backslash\mathcal{I}}$ and $(s_i)_{i\in\mathcal{I}}$.*
if $b=0$: $\mathcal{D}_0$ with $\mathcal{A}$'s shares *samples $(s_i)_{i\in\mathcal{I}}\xleftarrow{\$}\mathbb{F}^t_p$, sets $s_i:=0$ $\forall i\in[n]\setminus\mathcal{I}$, returns both $(\mathsf{Enc}_{\mathsf{pk}_i}(s_i))_{i\in[n]\backslash\mathcal{I}}$ and $(s_i)_{i\in\mathcal{I}}$.*
At some point $\mathcal{A}$ outputs b' and wins if $b{=}b'$.

Proof. We are going to bound the advantage of any adversary $\mathcal{A}$, by the maximum advantage of an adversary $\mathcal{A}_{\mathsf{PKE}}$ against oracle $\mathbb{O}_{\mathsf{PKE}}$ of the following $t{+}1$-keys variant indistinguishability game for PKE. The latter is upper-bounded by $n\text{-}t$ times the advantage for one-message indistinguishability, see e.g. [11, Thm 5.1]. $\mathbb{O}_{\mathsf{PKE}}$ samples $(n\text{-}t)$ PKE public keys $(\mathsf{pk}_h)_{h\in[n\text{-}t]}$ which it gives to $\mathcal{A}_{\mathsf{PKE}}$. $\mathbb{O}_{\mathsf{PKE}}$ secretly tosses a bit $b\in\{0,1\}$. Upon receiving, from $\mathcal{A}_{\mathsf{PKE}}$, $(n\text{-}t)$ chosen plaintexts $(s_h)_{h\in[n\text{-}t]}$, then $\mathbb{O}_{\mathsf{PKE}}$ returns:
- if $(b=1)$: correct encryptions, i.e., $(\mathsf{Enc}(\mathsf{pk}_h,s_h))_{h\in[n\text{-}t]}$;
- if $(b=0)$: encryptions of 0, i.e., $(\mathsf{Enc}(\mathsf{pk}_h,0))_{h\in[n\text{-}t]}$.
The reduction is as follows. $\mathcal{A}_{\mathsf{PKE}}$ initiates $\mathcal{A}$, receives a set of indices $\mathcal{I}$ from $\mathcal{A}$. $\mathcal{A}_{\mathsf{PKE}}$ receives a list of $n\text{-}t$ keys $(\mathsf{pk}_h)_{h\in[n\text{-}t]}$ from $\mathbb{O}_{\mathsf{PKE}}$. Then it forwards to $\mathcal{A}$ the $n\text{-}t$ keys $(\mathsf{pk}_h)_{h\in[n]\backslash\mathcal{I}}$ received from $\mathbb{O}_{\mathsf{PKE}}$, of which it renumbered the indices into $[n]\setminus\mathcal{I}$.

Upon receiving one challenge plaintext s from $\mathcal{A}$, $\mathcal{A}_{\mathsf{PKE}}$ generates a Shamir sharing of it: $(s_i)_{i\in[n]}:=\mathsf{Share}(s)$. It then queries $\mathfrak{O}_{\mathsf{PKE}}$ with the n-t plaintexts: $(s_h)_{h\in[n]\setminus\mathcal{I}}$. Upon receiving the response ciphertexts $(c_h)_{h\in[n]\setminus\mathcal{I}}$ from $\mathfrak{O}_{\mathsf{PKE}}$, it forwards them to $\mathcal{A}$, along with the t plaintext shares of indices of corrupt players $(s_i)_{i\in\mathcal{I}}$. $\mathcal{A}_{\mathsf{PKE}}$ outputs the same bit b' as $\mathcal{A}$. Analysis:

- in the case where the ciphertexts $\{c_h\}_{h\in[n]\setminus\mathcal{I}}$ are encryptions of the actual n-t shares $\{s_h\}_{h\in[n]\setminus\mathcal{I}}$, then $\mathcal{A}$ receives from $\mathcal{A}_{\mathsf{PKE}}$ a correctly generated encrypted sharing of s;
- in the case where the ciphertexts $\{c_h\}_{h\in[n]\setminus\mathcal{I}}$ are encryptions of 0, then we have that the t plaintext shares $\big((s_i)_{i\in\mathcal{I}}$ vary uniformly at random, independently of the rest of the view. This follows from what was denoted as "t-privacy" of Shamir sharing, in Section 3.3.

Thus in both cases $b\in\{0,1\}$, $\mathcal{A}$ is faced with the same distribution as the one generated by the oracle $\mathfrak{O}$ of the game of Proposition 10 for the same b and query s. Thus the distinguishing advantage of $\mathcal{A}_{\mathsf{PKE}}$ is the same as the one of $\mathcal{A}$. $\square$

The next corollary Corollary 11 states that any PPT adversary $\mathcal{A}$, which provides a ciphertext c of its choice, has negligible advantage in distinguishing between a vector of encrypted resharing of c, and a sample in the same fixed distribution as the $\mathfrak{D}_0$ of Proposition 9. To avoid issues with IND-CCA, we impose in addition that $\mathcal{A}$ sufficiently explains how c was formed, namely: as one homomorphic linear combination of fresh encryptions, in order to extract from $\mathcal{A}$ a plaintext of c. For simplicity and compatibility with Definition 8 with we state it only for $n=2t+1$.

Corollary 11 (IND-CPA of re-sharing)**.** *Any PPT machine $\mathcal{A}$ has negligible advantage in the following game with an oracle $\mathfrak{O}$. $\mathcal{A}$ gives to $\mathfrak{O}$ a subset of t indices $\mathcal{I}\subset[n]$. $\mathfrak{O}$ generates $(\dots,\mathsf{pk}_i)\leftarrow\mathsf{KeyGen}()$ for $i\in[n]\setminus\mathcal{I}$ which it shows to $\mathcal{A}$. It also generates $(\mathsf{sk},\mathsf{pk})\leftarrow\mathsf{KeyGen}()$ and gives pk to $\mathcal{A}$. $\mathfrak{O}$ tosses $b\overset{\$}{\leftarrow}\{0,1\}$. Then $\mathcal{A}$ is allowed to query $\mathfrak{O}$ an unlimited number of times as follows. $\mathcal{A}$ queries $\mathfrak{O}$ with any c appended with the following, denoted as "explanations": $(m_i)_{i\in[t+1]}\in\mathbb{F}_p^{t+1}$ and $(\rho_{enc,i})_{i\in[t+1]}\in\mathcal{R}_{enc}^{t+1}$ and $(ld_i)_{i\in[t+1]}\in\mathbb{F}_p^{t+1}$, such that $c=\boxplus_{i\in[t+1]}\big(\lambda_i\boxdot\mathsf{Enc}(m_i|\rho_{enc,i})\big)$. $\mathfrak{O}$ responds as:*
if $b=1$: enc. re-sharing + $\mathcal{A}$'s shares *decrypts $s:=\mathsf{Dec}(\mathsf{sk},c)$, generates $(s_i)_{i\in[n]}:=\mathsf{Share}(s)$, returns $(\mathsf{Enc}_{\mathsf{pk}_i}(s_i))_{i\in[n]\setminus\mathcal{I}}$ and $(s_i)_{i\in\mathcal{I}}$;*
if $b=0$: $\mathfrak{D}_0$ with $\mathcal{A}$'s shares *samples $s_i\overset{\$}{\leftarrow}\mathbb{F}_p$ $\forall i\in\mathcal{I}$, sets $s_i:=0$ $\forall i\in[n]\setminus\mathcal{I}$, returns both $(\mathsf{Enc}_{\mathsf{pk}_i}(s_i))_{i\in[n]\setminus\mathcal{I}}$ and $(s_i)_{i\in\mathcal{I}}$.*
At some point $\mathcal{A}$ outputs b' and wins if $b=b'$.

Proof. By perfect correctness after one homomorphic linear combination (Definition 8), we have $s=\sum_{i\in[t+1]}\lambda_i m_i$. Hence, informally, the view of $\mathcal{A}$ is exactly the same as in Proposition 10. More precisely, we have the following lossless reduction. Consider an adversary $\mathcal{A}'$ in Proposition 10. $\mathcal{A}'$ runs internally $\mathcal{A}$, then upon receiving a request, decrypts c into s, then gives s to $\mathfrak{O}$ the oracle of Proposition 10, then forwards to $\mathcal{A}$ the response of $\mathfrak{O}$, then outputs the same as $\mathcal{A}$. $\square$

C Formalization of protocol Y-VSS

In Appendices C.1 to C.3 we formalize Y-VSS.Share, .Refresh and .Open. In Appendix C.4 we explain how parties continue the protocol even if some keys of the next committee are not published when they receive the signal to refresh.

C.1 Formalization of Y-VSS Share. Every party $P_i^{(1)}\in\mathcal{C}^{(1)}$ generates $(\mathsf{pk}_i^{(1)},\mathsf{sk}_i^{(1)})\leftarrow\mathsf{KeyGen}()$ then registers it to $\mathcal{F}_{\mathrm{CA}}$. Upon receiving $\mathsf{share sig}$, the dealer $\mathcal{D}$ retrieves the list of keys $\{\mathsf{pk}_i^{(1)}\}_{i\in[n]}$ and generates $\mathsf{pvs}\leftarrow\mathsf{pvShare}(s)$ from its secret input s, using the n encryption keys $\{\mathsf{pk}_i^{(1)}\}_{i\in[n]}$. Then, $\mathcal{D}$ broadcasts pvs to $\mathcal{C}^{(1)}$ with its signature on it then shuts-off.

Each party $P_i^{(1)}\in\mathcal{C}^{(1)}$, upon receiving an output of the broadcast of $\mathcal{D}$: if it is a $\mathsf{pvs}\in\mathsf{pvS}$, then it sets its local list $\mathrm{ListOfPps}_i:=\{\mathsf{pvs}\}$. Else, which happens only if the dealer is corrupt, then it sets it as $\{\mathsf{pvs}_0\}$ where pvs_0 is a fixed predefined pvS of 0.

C.2 Formalization of Y-VSS Refresh. We first present the data structures in Figure 11, then the $\mathsf{Refresh}(\mathcal{C},\mathcal{C}')$ protocol between an exiting committee $\mathcal{C}$ and an entering committee $\mathcal{C}'$ in Algorithm 12.

C.3 Y-VSS-Open. Each $P_i^{(e_o)}\in\mathcal{C}^{(e_o)}$, upon receiving the signal "open", denoting $(\mathsf{pk}_i,\mathsf{sk}_i)$ its key pair, We consider any arbitrary committee $\mathcal{C}^{(e_o)}$ which is instructed to open the secret to a designated learner $\mathcal{L}$. Each party $P_i\in\mathcal{C}^{(e_o)}$, for every $\mathsf{pps}^{(e_o)}:=\big(\dots,\dots,c_{[n]}^{(e_o)=(c_i^{(e_o)})_{i\in[n]}}\big)$ in its list $\mathrm{ListOfPps}_i$:
- generates a publicly verifiable decryption of c_i: $(s_i,\pi_{\mathsf{pvD},i})$, to obtain the triple $(c_{[n]}^{(e_o)},s_i,\pi_{\mathsf{pvD},i})$

It then sends to $\mathcal{L}$, via $\mathcal{F}_{\mathrm{ST}}$, all such triples at once. Upon receiving $t{+}1$ triples of the form $(c_{[n]}^{(e_o)},s_i,\pi_{\mathsf{pvD},i})$ for i in some $(t{+}1)$-sized subset $\mathcal{U}$, all with the same $c_{[n]}^{(e_o)}$, $\mathcal{L}$ reconstructs the secret from the Lagrange linear combination of the $(s_i)_{i\in\mathcal{U}}$, given by Equation (3), with coefficients $(\lambda_i^{\mathcal{U}})_{i\in\mathcal{U}}$.

C.4 Handling for unpublished keys. The modification to be done for the general case is: upon receiving $\mathsf{refresh sigs}$ (or the $\mathsf{share sig}$), a party retreives from the PKI all the public keys available of the entering committee, say $\mathcal{C}'$, makes a list of them, say $\mathbf{pk}'$, in which it sets to $\perp$ the ones not retrieved. Encryption with key $\perp$ is by convention $\perp$. It appends the hash of $\mathbf{pk}'$ in all its messages related to $\mathcal{C}'$, and subsequently ignores the incoming messages related to $\mathcal{C}'$ appended with a different hash, i.e., which use a different list of keys than $\mathbf{pk}'$. We also adapt the proof of correctness, as follows, to handle executions in which there

- a $\mathsf{VPS}^{(e)}$, called as a *verified proactivized sharing relatively to committee* $\mathcal{C}^{(e)}$, is a tuple of the form $\mathsf{vps}^{(e)}{=}(c_{[n]},\mathsf{qvc}^{(e)})$, where: $c_{[n]} = (c_i)_{i\in[n]}\in\mathscr{C}^n$, and $\mathsf{qvc}^{(e)}{=}\{\sigma_i\}_{i\in\mathcal{U}}$, called as a *quorum verification certificate*, is a set of $t{+}1$ signatures on $c_{[n]}$ issued by some $t{+}1$ subset $\mathcal{U}\subset[n]$ of parties of $\mathcal{C}^{(e)}$.
  For brevity we denote $\mathbf{vps}[i] := c_i$.
- a $\mathsf{ReshareMsg_i}^{(e)}$ for $i\in[n]$, called as a *resharing message from* $P_i\in\mathcal{C}^{(e)}$, is a triple $(c_{[n]}^{(e)},\mathsf{pvr}_i^{(e)},\sigma_i^{(e)})$, where:
  - $c_{[n]}^{(e)} = (c_i^{(e)})_{i\in[n]}\in\mathscr{C}^n$
  - $\mathsf{pvr}_i^{(e)}\in\mathsf{pvR}\big(\mathsf{pk}_i^{(e)},c_i^{(e)}\big)$, i.e., is a publicly verifiable resharing of $c_i^{(e)}$ under the key $\mathsf{pk}_i^{(e)}$ of P_i, encrypted under the keys of $\mathcal{C}^{(e+1)}$;
  - and $\sigma_i^{(e)}$ is a signature of $P_i^{(e)}$ on $c_{[n]}^{(e)}$.
- a $\mathsf{PPS}^{(1)}$, called as a *proven proactivized sharing relatively to committee* $\mathcal{C}^{(1)}$, is either a pvS signed by the dealer $\mathcal{D}$, denoted as $\mathsf{pps}^{(1)}:=(\sigma_{\mathcal{D}},\pi_{\mathsf{pvS}}\xrightarrow{\mathscr{L}}c_{[n]}\in\mathscr{C}^n)$ or, the public default $\mathsf{PPS}^{(1)}$ of zero: $\mathsf{pps}_0^{(1)} = \big(\bot,\bot\xrightarrow{\mathscr{L}}(\mathsf{Enc}_{\mathsf{pk}_i}(0))_{i\in[n]}\big)$.
- a $\mathsf{PPS}^{(e+1)}$, called as a *proven proactivized sharing relatively to committee* $\mathcal{C}^{(e+1)}$, is a tuple of the form $\mathsf{pps}^{(e+1)}\big((c_{[n]}^{(e)},\mathsf{qvc}^{(e)}),\{\mathsf{pvr}_i^{(e)}\}_{i\in\mathcal{U}}\big)$, where:
  - $\mathcal{U}\in[n]$ is a $(t{+}1)$-sized subset;
  - $(c_{[n]}^{(e)} = (c_i^{(e)})_{i\in[n]},\mathsf{qvc}^{(e)})$ is a $\mathsf{VPS}^{(e)}$; [Notice that there is no constraint on the subset of issuers, in $\mathcal{C}^{(e)}$, of the signatures in the $\mathsf{qvc}^{(e)}$. In practice in Y-VSS, honest collectors use those issued by $\mathcal{U}$.];
  - $\mathsf{pvr}_i^{(e)}\in\mathsf{pvR}\big(\mathsf{pk}_i^{(e)},c_i^{(e)}\big)\ \forall i\in\mathcal{U}$, i.e., each $\mathsf{pvr}_i^{(e)}$ is a publicly verifiable resharing of $c_i^{(e)}$ under the key $\mathsf{pk}_i^{(e)}$ of P_i, encrypted under the keys of $\mathcal{C}^{(e+1)}$;

  We denote it as
  $$\mathsf{pps}^{(e+1)}:=\big((c_{[n]}^{(e)},\mathsf{qvc}^{(e)}),\{\mathsf{pvr}_i^{(e)}\}_{i\in\mathcal{U}}\xrightarrow{\mathscr{L}}c_{[n]}^{(e+1)}\big)$$
  where, denoting $\mathsf{pvr}_i^{(e)} = (c_{i\to[n]}^{(e)},\pi_{\mathsf{pvR},i})\ \forall i$, we have $c_{[n]}^{(e+1)}:=\boxplus_{i\in\mathcal{U}}\big(\lambda_i^{\mathcal{U}}\boxdot c_{i\to[n]}^{(e)}\big)\in\mathscr{C}^n$ (as in Equation (1)).

Figure 11: Data Structures for Refresh. The public keys of committee $\mathcal{C}^{(e)}$ are denoted as $(\mathsf{pk}_i^{(e)})_{i\in[n]}$, for all $e\in\mathbb{N}^*$

$\mathsf{Refresh}(\mathcal{C},\mathcal{C}')$

Inputs: Each party $P_i\in\mathcal{C}$ has a key pair $(\mathsf{pk}_i,\mathsf{sk}_i)$, and a list $\mathrm{ListOfPps}_i$ of at most $t{+}1$ PPSs relatively to $\mathcal{C}$.
Outputs: For each player $P_i'\in\mathcal{C}'$, a list $\mathrm{ListOfPps}_i$ of PPS relatively to committee $\mathcal{C}'$.

(PKI) At signal startsig, each party $P_i'\in\mathcal{C}'$ generates $(\mathsf{sk_i'},\mathsf{pk}_i')\leftarrow\mathsf{KeyGen}()$ and publishes (pk_i');
(Resharing) At signal refreshsig, each $P_i\in\mathcal{C}$ retrieves the keys of $\mathcal{C}'$: $(\mathsf{pk}_i')_{i\in[n]}$. Then for each $\mathsf{pps}{=}\big(\dots,\dots,c_{[n]}{=}(c_i)_{i\in[n]}\big)$ in its list, it generates:
- $\mathsf{pvr}_i = (c_{i\to[n]},\pi_{\mathsf{pvR},i}):=\mathsf{pvReshare}(\mathsf{pk}_i,\mathsf{sk_i},c_i)$.
- a signature σ_i on $c_{[n]}$;
- forms the *resharing message* $\mathsf{ReshareMsg_i}$: $\big(c_{[n]},\mathsf{pvr}_i,\sigma_i\big)$.

P_i multicasts to $\mathcal{C}'_{\text{collec}}$, in one single batch, all the resharing messages which it formed, then shuts-off.
(Selection & combination) Each collector $K_k'\in\mathcal{C}'_{\text{collec}}$ waits until it receives, for some $(t{+}1)$-sized subset $\mathcal{U}\subset[n]$, resharing messages from parties in $\mathcal{U}$ formed out of the *same* vector of ciphertext shares $c_{[n]}$:

$$\big(c_{[n]},\mathsf{pvr}_i,\sigma_i\big),\ \ \forall i\in\mathcal{U}\ . \tag{6}$$

Then it combines the $t{+}1$ signatures into a qvc for $c_{[n]}$, thereby obtaining a verified proactivized sharing: $\mathsf{vps}:=(c_{[n]},\mathsf{qvc})$. Hence, it obtains the PPS: $\big(\mathsf{vps},\ (\mathsf{pvr}_i)_{i\in\mathcal{U}},\ c'_{[n]}:=\boxplus_{i\in\mathcal{U}}\lambda_i^{\mathcal{U}}\boxdot c_{i\to[n]}\big)$, which it multicasts to $\mathcal{C}'$ then shuts-off.
(Outputs) Upon receiving a PPS, any party $P_j'\in\mathcal{C}'$ adds it to its $\mathrm{ListOfPps}_j'$.

Algorithm 12: $\mathsf{Refresh}(\mathcal{C},\mathcal{C}')$

exists some committee $\mathcal{C}'$ for which at most t published keys are explainable. For any vector of ciphertext shares $c_{[n]}$ under such keys, then strictly less than t coordinates have an explainable decryption. Thus, $c_{[n]}$ has no threshold opening in the sense of Definition 3. We then say that $c_{[n]}$ is *unusable*. Indeed, no $t{+}1$ opening shares can possibly be formed, so no PPS can possibly be formed out of $c_{[n]}$, nor any Open can complete out of $c_{[n]}$. We then need to relax the *Claim* in the proof of correctness (Proposition 7), by allowing that a VPS can alternatively be unusable. Then in the proof of this *Claim*, we slightly precise the conclusion by observing that the VPS: $(c_{[n]},\mathsf{qvc})$ is necessarily usable, and thus has threshold opening equal to $\widetilde{s}$.

We also complement the proof of liveness by proving that, if some unusable VPS is ever created, then it must be the case that some honest party could not publish its key in time, and thus the conditions under which we guarantee liveness actually do not hold.

However in the proof of privacy, if the key of some honest party was not published in time, then we cannot anymore compute $t{+}1$ honest opening shares by decryption. Thus, we cannot anymore apply ShInfer. Thus, assuming such executions exist, we must require straight-line extractable NIZKs to enable the simulator to extract the opening shares of corrupt players, as in the Game $\mathsf{Hyb}^{0\mathsf{Share}}$ in the proof in Section 5.1.

D More details on the proof of Y-VSS

In Appendix D.1 we formalize the reduction from the proof of indistinguishability between games $\mathsf{Hyb}^{0\mathsf{Refresh}}[e,i]$ and $\mathsf{Hyb}^{0\mathsf{Refresh}}[e,i+1]$, into Corollary 11. Then, in Appendix D.2 we adapt the proof of liveness sketched in Section 1.3, into a formal proof of liveness of the actual Y-VSS in 2δ. In Appendix D.3 we formalize the proof of UC security in the case of a corrupt dealer $\mathcal{D}$.

D.1 Reduction of the transition to $\mathsf{Hyb}^{0\mathsf{Refresh}}[e,i+1]$ into IND-CPA of resharing. We consider an adversary $\mathcal{A}_{\text{reshare}}$ in the game of Corollary 11. It initiates a concatenation, which we denote as $\mathscr{M}$, of all the system up to the e-th committee as in the game $\mathsf{Hyb}^{0\mathsf{Refresh}}[e,i]$: $\mathcal{E}$ and the dummy adversary and all corrupt parties, $\mathcal{D}$, the ideal functionalities, and all honest parties in $\mathcal{C}^{(1)},\dots,\mathcal{C}^{(e)}$. Upon receiving keys $(\mathsf{pk}_i)_{i\in[n]\setminus\mathcal{I}}$ from $\mathfrak{S}$, it publishes them on behalf of honest parties in $\mathcal{C}^{(e+1)}$ (and, by construction, also publishes the keys of corrupt parties in $\mathcal{C}^{(e+1)}$, when they are instructed to by $\mathcal{E}$). Consider a point (if any), where the simulated $P_{i+1}^{(e)}$ receives a $\mathsf{PPS}^{(e_o\text{-}1)}$, out of which it deduces, by Lagrange combination, an encrypted share c_i. Then this means that $\mathscr{M}$ also provided to $\mathcal{A}$ the explanation of $\mathscr{M}$. Namely, $\mathcal{A}_{\text{reshare}}$ could extract from the $\mathsf{PPS}^{(e_o\text{-}1)}$ provided by $\mathscr{M}$: $t{+}1$ explainable fresh encryptions, of which c_i is the Lagrange homomorphic linear combination. Then, $\mathcal{A}_{\text{reshare}}$ immediately removes $P_{i+1}^{(e)}$ from $\mathscr{M}$. Then $\mathcal{A}_{\text{reshare}}$ queries $\mathfrak{S}$ with this material, and is returned a vector of ciphertexts, denoted $c_{i\to[n]}$, along with the plaintext sub-shares $(s_{i\to j})_{j\in\mathcal{I}^{(e+1)}}$ of $c_{i\to[n]}$ for the corrupt indices. Then $\mathcal{A}_{\text{reshare}}$ re-integrates $P_{i+1}^{(e)}$ into $\mathscr{M}$, in which it overwrites its actual re-sharing by $c_{i\to[n]}$. Then $\mathcal{A}_{\text{reshare}}$ finishes to run the execution. Noticeably, thanks to $\mathsf{Hyb}^{0\mathsf{Refresh}}[e+1,n]$ $\mathcal{A}_{\text{reshare}}$ does not need to know the secret keys corresponding to those published on behalf of honest parties in $\mathcal{C}^{(e+1)}$, to continue the execution. Noticeably, if $e{=}e_o\text{-}1$, then $\mathcal{A}_{\text{reshare}}$ inputs into ShSim the plaintext sub-shares $(s_{i\to j})_{j\in\mathcal{I}^{e_o}}$ of $c_{i\to[n]}$ with indices the corrupt parties in $\mathcal{C}^{(e_o)}$. *This last precision explains why we specified leakage of the plaintext (sub)-shares in the game of Corollary 11.* Then, $\mathcal{A}_{\text{reshare}}$ outputs $b=1$ if $\mathcal{E}$ outputs i, or outputs $b=0$ if $\mathcal{E}$ outputs 0. Since the view of $\mathcal{E}$ is generated exactly as in $\mathsf{Hyb}^{0\mathsf{Refresh}}[e,i]$ if $b=1$ and as in $\mathsf{Hyb}^{0\mathsf{Refresh}}[e,i+1]$ if $b=0$, we have that the distinguishing advantage of $\mathcal{A}_{\text{reshare}}$ is as least as large as the one of $\mathcal{E}$, which concludes the proof.

D.2 Proof of liveness. We consider an execution which satisfies all conditions stated in Theorem 6 for liveness, i.e., [LS], [LK], [LR] and [LO]. Let us make the assumption that all parties in the committee $\mathcal{C}^{(e_o)}$ which receives $\mathsf{open\,sig}$, have at least one PPS: pps in common in their lists $\mathrm{ListOfPps}_i$ when they receive the $\mathsf{open\,sig}$. Since by [LO], all their messages to the learner $\mathcal{L}$ are delivered, $\mathcal{L}$ will obtain $t{+}1$ verifiable plaintext decrypted shares of this same pps. Thus it will be able to output, which concludes liveness. It thus remains to prove the assumption.

Lemma 12. *For any $1\leq e\leq e_o$, consider the local lists $\mathrm{ListOfPps}_i$ of honest parties in $\mathcal{C}^{(e)}$ when they receive the* $\mathsf{refresh\,sig}$ *(or* $\mathsf{open\,sig}$, *for $\mathcal{C}^{(e_o)}$), then there is at least one common element* $\mathsf{pps}\in\mathsf{PPS}$ *in all these lists.*

Proof. We prove the statement of the lemma by induction on e. *Case $e{=}1$*: by [LS], all honest parties in $\mathcal{C}^{(1)}$ have received the same output from the broadcast of the dealer $\mathcal{D}$ before they receive $\mathsf{refresh\,sig}$ or $\mathsf{open\,sig}$. Either this output is a correct pvS, in which case they all have their local lists $\mathrm{ListOfPps}_i$ equal to this single $\{\mathsf{pvS}\}$. Or if not the case, they all have their local lists $\mathrm{ListOfPps}_i$ equal to the same default $\{\mathsf{pps}_0\}$.

$e\Rightarrow e+1$ Let us assume the statement true until the committee $\mathcal{C}$ of some epoch $e<e_o$, and let us prove it for the next committee $\mathcal{C}'$. This will imply that statement for all e, by induction on e. After all parties of $\mathcal{C}$ receive $\mathsf{refresh\,sig}$, if we add a δ delay, then at this point there is at least one honest collector K_k' in $\mathcal{C}'_{\text{collec}}$ which will receive at $t{+}1$ reshare messages out of the same common pps from all $t{+}1$ honest players. Thus it is able to form a PPS: $\mathsf{pps}=(\mathsf{vps},\{\mathsf{pvr}_i\}_i,c'_{[n]})$ out of them and multicast it. If we add a δ delay, by [LR], no honest party in $\mathcal{C}'$ has yet received $\mathsf{refresh\,sig}$ or $\mathsf{open\,sig}$. So at this point, all honest parties in $\mathcal{C}'$ are still alive and have received this same pps, so included it its local list $\mathrm{ListOfVps}'_i$. □

D.3 Proof of UC security in case of a corrupt dealer $\mathcal{D}$. The simulator $\mathcal{S}_c$ for a corrupt dealer is very simple. It simulates honest parties, in each committee, following the protocol and reacting to the signals, exactly as instructed by Y-VSS. If the learner $\mathcal{L}$ is also corrupt, then this behavior perfectly simulates honest participants.

If instead $\mathcal{L}$ is honest (dummily interacting with $\mathcal{F}_{\text{VSS}}$), which we now describe, then the *moment* when it outputs and its *value* output are learned by $\mathcal{E}$. Thus, $\mathcal{S}_c$ must manage so that those two parameters are the *same* as in the simulated execution. $\mathcal{S}_c$ waits for the broadcast from $\mathcal{D}$, via the (simulated) broadcast functionality $\mathcal{F}_{\text{BC}}$, to deliver an output to at least one of the simulated honest parties in $\mathcal{C}^{(1)}$. If this output is not a (valid) pvS then it sets $\widetilde{s}:=0$. Otherwise, $\mathcal{S}_c$ internally computes $t{+}1$ opening shares of it [using the secret decryption keys of the simulated honest parties in $\mathcal{C}^{(1)}$] then applies $\mathscr{L}\text{-}\mathsf{combine}^{\mathcal{U}}$, to obtain some $\widetilde{s}\in\mathbb{F}_p$. $\mathcal{S}_c$ sends $(\mathsf{share},\widetilde{s})$ to $\mathcal{F}_{\text{VSS}}$. Then, if this ever happens, at the moment when $t{+}1$ opening shares are delivered to the simulated $\mathcal{L}$, then $\mathcal{S}_c$ immediately sends "$\mathsf{open\,sig}$ " to $\mathcal{F}_{\text{VSS}}$. Thus, we have that the real dummy $\mathcal{L}$ is delivered its output $\widetilde{s}$, by $\mathcal{F}_{\text{VSS}}$, exactly at the same moment. Moreover, by Proposition 7 [unicity] and [output], the ouptut in the simulated execution is equal to the one the real dummy $\mathcal{L}$.

In conclusion, the simulated execution is identical to a real one from the point of view of $\mathcal{E}$.

E More context on the baseline method

The baseline technique of Section 1.2, of re-sharing of secret shares, is known for long in a simpler "vanilla" version, in which the sub-shares are not publicly encrypted, but instead sent by private channels. It is credited to Michael Rabin circa 1988, by [28, §5.6], in the context of multiplication of secrets. It surfaced in a written form in [37]. Meanwhile, the vanilla version was rediscovered by [29] in the context of proactivity, then used in [68, 13, 46, 31]. They ([29]) state it for secrets in any space equipped with a linear secret sharing scheme, which enables the generality claimed in Theorem 1. It turns out that the construction of publicly verifiable vectors of encrypted secret shares was first suggested in [41, §3.2]. In the context of proactivity, it turns out that the same baseline as above, of *encrypted resharing*, is also considered in [10, 44]. Their model makes it trivial to apply the baseline, as follows. For simplicity of this description we do as if they also assumed and leveraged homomorphic additivity of ciphertexts. Their invariant is that all members of the exiting committee $\mathcal{C}$ have a common view on a *single*, consistent, vector of ciphertext shares $c_{[n]}$. There is a global clock, which, at some point, simultaneously instructs to all parties of $\mathcal{C}$ to refresh. At this point, each party generates a pvR of $c_{[n]}$ then broadcasts it. The broadcast is assumed to terminate within a fixed delay Δ_{BC}. In these works, it is embodied by a public ledger. After Δ_{BC}, each party in $\mathcal{C}'$ sets $\mathcal{U}$ equal to the $t+1$ first indices of $\mathcal{C}$ for which which the broadcasts returned a (valid) pvR, and computes the new vector of encrypted shares $c'_{[n]}$ out of them. Since all players in $\mathcal{C}'$ are returned the same outputs of the n broadcasts, they are guaranteed to see the same set $\mathcal{U}$, and thus to end up with the same $c'_{[n]}$. Said otherwise, these n terminating broadcasts implement consensus on $\mathcal{U}$.

F More on efficiency and generalizations

F.1 Parallelized implementation of Refresh$(\mathcal{C}, \mathcal{C}')$. In Fig. 13 we present the performance of the implementation of a Refresh$(\mathcal{C}, \mathcal{C}')$, with number of parties ranging from 11 to 101. We display the result for different number of cores used, ranging from 1 to 4, to show that operations can indeed be parallelized. More precisely, we approximated the number of cores by the number of Go-routines. We observe a total computation time which is roughly divided by the number of cores used, although slightly higher than the exact division, because of some hardware/software incompatibilities.

F.2 Other instantiations of linear combinations of secrets over rings / groups.

With polynomial coefficients. Consider secrets which are in a polynomial ring, e.g., for simplicity, of the form $\mathbb{F}_p[X]/f(X)$. The goal is that additions of shares, or

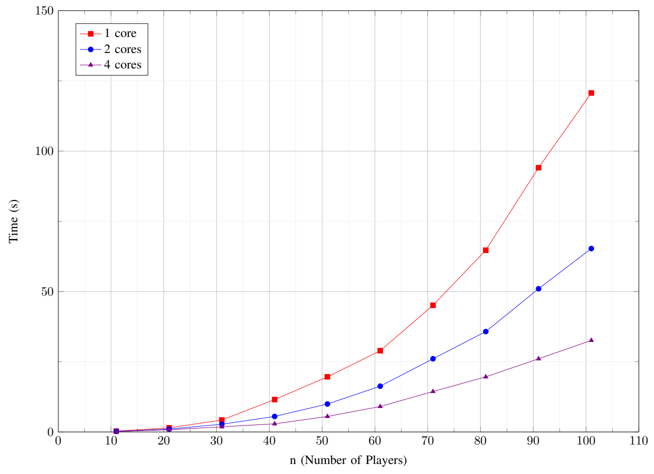


Figure 13: Total worst-case time of Refresh$(\mathcal{C}, \mathcal{C}')$ between two committees $\mathcal{C}$ and $\mathcal{C}'$ measured in seconds on the y-axis. The x-axis is the size n of both committees, ranging from 11 to 101. We specify for each setting the time spent in each of the high-level subroutine:**Encrypted re-sharing** for each parties in $\mathcal{C}$ to generate $t+1$ pvR and signatures, **Collection** for a single leader to verify the $t+1$ pvRs and signatures (cf Fig. 1), **Combination** for a party in $\mathcal{C}'$ to combine $t+1$ pvR (including the verification of the NIZKs and the $t+1$ signatures) for the $t+1$ PPSs it receives. We display the result for different number of cores used, ranging from 1 to 4.

multiplications of shares by a public polynomial, modulo $f(X)$, result in the same operations on the secret. To this end, we need a slight variant of Shamir's secret sharing scheme which is linear with respect to these operations. The simplest example of such sharing scheme is provided [51, IV. A], then a more general one in [57], in the context of threshold RLWE-based schemes. An example of PKE compatible with these operations on the plaintexts is BFV [33, 50]. However in practical applications the secrets are themselves in the (large) polynomial ring of ciphertexts of some threshold FHE scheme, such as BFV, so this would require to instantiate PKE with another BFV, of larger plaintext space. So in this case it is preferable to use the alternative method proposed in [60], with any arbitrary PKE.

Over rings of machine integers $\mathbb{Z}/2^k\mathbb{Z}$. To this end we need Shamir sharing over $\mathbb{Z}/2^k\mathbb{Z}$, as described in [30, 34, 1]. A very recent example of encryption scheme compatible with linearly homomorphic operations in $\mathbb{Z}/2^k\mathbb{Z}$ is [23].

F.3 Allowing slack in the NIZKs of smallness. In all the schemes considered above, if the parameters are such that $(t+1)p$ is strictly smaller than the upper bounds provided, then perfect correctness holds even against a malicious encryptor which would choose a plaintext and randomness larger than the bounds specified (p, $\mathcal{R}_{\mathrm{key}}$, $\mathcal{R}_{\mathrm{enc}}$). In turn, this allows to use NIZKs of smallness in which a malicious prover can pass verification with input size larger, by some "slack", than the maximum size that a honest prover is able to input. Turning to Paillier, one can observe that NIZKs of re-sharing are easier done by Pedersen committing to the

Paillier plaintext share, proving equality with the opening of the commitment, then performing NIZKs on this commitment. State of the art implementations of such NIZKs are in [53, §6.2] and [17], applied to threshold ECDSA. This same observation is made, in GHL [39], for lattice-based schemes. They bring optimized NIZK relations (instantiated with Bulletproofs) for resharing, recently improved in [55].