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). Expessed 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 DH-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.

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 \([43]\), 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, \( D \), 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, \([57]\) 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 \( C^{(1)} \);
- it maintains the correctness and liveness invariants that in each epoch \( e \), the current committee \( C^{(e)} \) holds shares of this same secret;
- to this end, there is a sub-protocol, denoted Refresh, which enables the committee \( C^{(e)} \) of some epoch \( e \), denoted as exiting, to provide new shares of the same secret to the next committee \( C^{(e+1)} \), 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 $C'$, very roughly, to reach consensus on a set of new secret shares obtained from the exiting committee $C$. More particularly, all existing Refreshes under honest majority \cite{10,55,42,17,43,38} require a protocol for consistent broadcast with, at least, eventual termination even if the sender is corrupt, such as in \cite{35}. We dub this primitive as terminating broadcast, or BC for short. Implementing BC beyond $t < n/3$ corruptions is impossible (\cite{31}) without the extra assumption, known as \textit{ synchrony}, that communication channels deliver messages within a fixed public delay $\Delta$. If one message arrives after $\Delta$, then consistency or termination of the BC is not guaranteed, which in turn ruins the security or liveness of the PSS; see Section \ref{sec:asynchronous}. This is why practical implementations of BC require many rounds and communications of fixed delay $\Delta$ 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 \cite{18}, which is the most popular PSS in the context of key-refresh, known under the name “CMP” \cite{34}. First, it is suggested by the authors, following \cite{31}, 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 \cite{25,2}. Second, in the final version of their PSS \cite{18} 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 $t$ (\cite[p. 1.2.7]{18}), 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 \ref{tab:comparative}), then state of the art PSS \cite{10,55,43,42} 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 \textit{actual speed of the network}, which is a property known as \textit{ responsiveness} \cite{58}. 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 \cite{23,10,59,40} 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 \cite{55,39,37,20,14}, then this prevents adaptive corruptions. To compensate this requirement, \cite{39} 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.

\textbf{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 26, where 6 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:

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

\textbf{Verifiability} Provided the additional specification that $D$ uses a BC to send its (unique) message, then Y-VSS has verifiability \cite{24,6}, in the strongest sense of \cite{64}, namely: the content of the broadcast of $D$ commits $D$ to exactly one value $s$, such that $s$ is the only value that the learner $L$ can possibly output.

\textbf{Asynchronous UC security.} More generally, we specify in Section \ref{sec:asynchronous} then prove, for the first time, asynchronous proactive verifiable secret (re)sharing in the universal composability (UC) framework of \cite{15}. “Asynchronous” means that we do not resort on any functionality which would guarantee some form of synchrony, be it explicit (\cite[p. 3.2]{53}) or implicit (\cite{27} §4.2).

\textbf{Minimal trust on intermediaries.} Y-VSS remains UC secure even if all collectors are corrupt. Furthermore if every Refresh is launched 26 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 $L$): we dub this property as \textit{liveness}.

\textbf{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 \ref{sec:linear} 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 \ref{sec:asynchronous} and \ref{sec:linear} such as linear combinations with polynomial coefficients, and also \textit{multi-exponentiations}, which enables threshold BLS signing.
1.2 Baseline: re-sharing of encrypted shares. Let us describe the baseline technique of a Refresh between an exiting committee $C := C^{(r)}$ and an entering committee $C’ := C^{(r+1)}$. Let us assume that all $t+1$ honest parties in $C$ have in common a vector of $n$ ciphertexts of Shamir shares of the secret, denoted as $c_i \in [n]$. Informally, this means that the decryption of each $c_i$ under the secret key of the $i$-th party $P_i$ 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 $U \subset [n]$, there exists public coefficients $(\lambda'_U)_{i \in U}$, denoted as Lagrange coefficients, such that $s = \sum_{i \in U} \lambda'_U s_i$. 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 $C’$ obtain a vector of ciphertexts $c’_i$ of new shares of the same secret. It consists of two steps.

Resharing First, each player $P_i \in C$ 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 $s_{i \rightarrow j} := f_i(j)$. Those shares $(s_{i \rightarrow j})_{j \in [n]}$ 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 $C’: c_i \rightarrow j := \text{Enc}_{pk_j}(s_{i \rightarrow j})$, where $pk_j$ is the public key of the $j$-th member of $C’$. Finally, it appends to the vector $c_i \rightarrow [n] := (c_{i \rightarrow j})_{j \in [n]}$ a non-interactive ZK argument of knowledge (NIZK AoK), denoted $\text{π}_{pkR}$, proving that $c_i \rightarrow [n]$ is indeed an encrypted resharing of a decryption of $c_i$. We dub such a pair $(c_i \rightarrow [n], \text{π}_{pkR})$ as a publicly verifiable resharing of $c_i$, shortened as pvR.

Combination Consider any $t+1$ pvR’s: $(c_{i \rightarrow [n]}, \text{π}_{pvR})_{i \in U}$, which are generated out of the same $c_i$, where $U \subset [n]$ is some $t+1$-sized subset. We have that the Lagrange linear combination of their plaintext coordinates: $(s'_i)_j \in U := \sum_{i \in U} \lambda'_U (c_{i \rightarrow j})_{i \in U}$ is a vector of (new) Shamir shares of the same secret $s$. This is because, as depicted in Figure 1, the $(s'_i)_j \in U$ are, by construction, evaluations of the new polynomial $f'_i := \sum_{i \in U} \lambda'_U f_i(X)$, whose evaluation at zero is $\sum_{i \in U} \lambda'_U s_i = s$. 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 $(s'_i)_j \in U$:

$$c’_{i \rightarrow [n]} := \mathcal{L} \text{-combine}_{pk} \left( (c_{i \rightarrow [n]})_{i \in U} \right) := \bigoplus_{i \in U} \lambda'_U \square c_{i \rightarrow [n]}$$

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 3δ instead of 2δ, and where parties speak multiple times, hence which is not “yoso”.

Sharing. The dealer of the secret, denoted $D$, broadcasts an encrypted sharing $c_i[n]$ of its secret. In a model where $D$ is always honest, then this can simply be implemented by sending (publicly) $c_i[n]$ 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 $C$ and an entering committee $C’$, as depicted on Figure 2.

Inputs and Outputs. Each party in $C$ 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 $D$. To be valid and included in a local list, any such vector $c_i[n]$ must furthermore come appended with $t+1$ signatures issued by $C$. We call these signatures as a qvc verification certificate, denoted as qvc. Existence of a qvc guarantees that at least one honest signer, in $C$, validated that $c_i[n]$ had been correctly formed, as we are going to detail. We call such a pair $(c_i[n], qvc)$ 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 $D$.
- The liveness invariant maintained is that, if Refreshes are launched at least 3δ one after the other, then all the lists of honest parties in $C$ contain at least one VPS in common.

Hence, we depart from all previous works [10, 43], which required parties in $C$ to start a Refresh with one common unique system of shares. Assuming that both correctness and liveness invariants hold for parties in $C$, the goal of a Refresh is to achieve, at the end, the same invariant for parties in $C’$.

Publicly encrypted re-sharing. We assume existence of an intermediary committee denoted as $C_{\text{collect}}$ and called as the collectors. The mission of each collector is to deliver to all parties in $C’$ the same output as in the baseline, namely: a batch of $t+1$ publicly verifiable re-sharings, originating from the same vector of shares. For every VPS: $\text{pvS} = (c_i[n], qvc)$ in its local list, each party of $C$ multicasts to $C_{\text{collect}}$: a publicly verifiable resharing $\text{pvR}_c = (c_{i \rightarrow [n]}, \text{π}_{pvR})_c$ of its encrypted share $c_i$, as in the baseline, appended with $\text{pvS}$. 

Simulatability of PVSS, without straight-line extraction. A by-product of our UC proof is that it positively answers in Section 5.4, supported by Appendix 5.1, the question raised by Shrestha-Bhat-Kate-Nayak in [65], 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, 59]), then UC security furthermore holds without requiring straight-line extractability of witnesses from NIZKs.
\[
\begin{align*}
\text{new sharing } & c'_{i[n]} := \\
\left\{ \begin{array}{l}
\text{Enc}_{p k_{i}}'\left( s'_{i} \right) = \bigoplus_{t \in U} \lambda^{t} \otimes \text{Enc}_{p k_{i}}'\left( s_{i+n} \right) \\
= f'(t) \\
\text{Enc}_{p k_{1}}'\left( s'_{1} \right) = \bigoplus_{t \in U} \lambda^{t} \otimes \text{Enc}_{p k_{1}}'\left( s_{i+1} \right) \\
= f'(1) \\
\text{Enc}_{p k_{n}}'\left( s'_{n} \right) = \bigoplus_{t \in U} \lambda^{t} \otimes \text{Enc}_{p k_{n}}'\left( s_{i+n} \right) \\
= f'(n) \\
\end{array} \right.
\end{align*}
\]

\[
\left\{ \begin{array}{l}
\text{old sharing} \ (c_{i} \in [n]) \\
\left\{ \begin{array}{l}
s = \sum_{i \in U} \lambda^{i} s_{i} = \boxed{s = \bar{f}(0)} \\
\end{array} \right.
\end{array} \right.
\]
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 4 we discuss various other instantiations of the encryption scheme and corresponding NIZKs of resharling, all from existing and implemented ingredients. In particular, the schemes of Paillier-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 [13]. 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 \( \mathbb{N}^* = \{ 1, 2, \ldots \} \). 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 abuse notations by identifying \( \mathbb{F}_p \) with the set of representatives \( \{ 0, \ldots, p - 1 \} \) 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 \( f \ll F \) the sampling of an element in \( F \) uniformly at random. The empty string is denoted as \( \perp \). For \( m \) an integer, we denote \( [m] := \{ 1, \ldots, m \} \). 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 \( D \), which has an input \( s \in \mathbb{F}_p \) denoted as its secret; and one PPT machine denoted learner \( L \), which may output some value in \( \mathbb{F}_p \) at some point. We consider a PPT machine called as the adversary and denoted \( A \). We are aiming at a protocol which securely implements, in the UC sense of [13], the ideal functionality of verifiable secret sharing, denoted as \( F_{\text{VSS}} \) and formalized in Algorithm 3. Very informally, a protocol which UC-implements \( F_{\text{VSS}} \) guarantees that, after \( D \) completed its task, there is one unique value \( \bar{s} \) which can possibly be released to \( L \), and such that furthermore \( \bar{s} = s \) if \( D \) is “honest” (see below). It also guarantees that \( A \) is leaked no information on \( s \) if \( L \) is “honest”. See below Theorem 6 for more details, then Appendix A.2 for the rigorous general definition.
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 $C^{(e)} = (P_1^{(e)}, \ldots, P_{n}^{(e)})$, 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 $C^{(e)}_{\text{collect}}$, denoted as a committee of collectors. Members of committees as denoted as parties. All the committees $(C^{(e)}, C^{(e)}_{\text{collect}})_{e \in [n]}$, are disjoint, notwithstanding 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 $F_{\text{Sch}}$ and which is actually fully controlled by $A$. $F_{\text{Sch}}$ can send a signal startsig to any party $P_i^{(e)}$ (for any $i \in [n]$ and $e \in [n]$) at any moment. From this point, $P_i^{(e)}$ can take actions, we say that it is alive. On their side, $D$ and $L$ start the protocol alive. $F_{\text{Sch}}$ can also send a signal, denoted as sharesig, to $D$. Informally, it instructs $D$ to share its secret to $C^{(1)}$. Before $D$ receives sharesig, $A$ has the possibly to (maliciously) corrupt it, in the sense below. Likewise, $A$ can maliciously corrupt any player $P_i^{(e)}$ before it receives startsig, up to $t$ parties per committee. $A$ may also corrupt $L$ at any point. Concretely, a corrupt participant ($D$ or $L$ or a party) is forever fully controlled by $A$ and forwards to $A$ all its incoming messages or signals. We denoted the subset of corrupt parties in each committee $C^{(e)}$ by $\mathcal{T}^{(e)} \subseteq [n]$. The non-corrupt participants as denoted as honest. The honest parties are indexed as $\mathcal{H}^{(e)} = C^{(e)} \setminus \mathcal{T}^{(e)}$.

2.4 Asynchrony formalized by ideal functionalities with delayed output. Following [15] we say that an ideal functionality $F$ sends a delayed output $v$ to $R$ if it engages in the following interaction: instead of simply outputting $v$ to $R$, $F$ 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 $A$ in the request. When the adversary replies, $F$ outputs $v$ to $R$.

2.5 Public authenticated asynchronous channels $F_{\text{AT}}$ and multicast. All parties in some committee $C^{(e)}$ are connected with all parties in $C^{(e+1)}_{\text{collect}}$, themselves connected with the next committee $C^{(e+1)}$, by public authenticated message transmitting with delayed output. It is formalized as the functionality $F_{\text{AT}}$ (denoted by $F_{\text{AUTH}}$ in [15]). It is parametrized by a sender $S$ and a receiver $R$. On input (input, ssid, $v$) from $S$: then $F_{\text{AT}}^{S,R}$ provides $R$ with public delayed output (ssid, $v$). To multicast a message to a committee simply means to send it over $F_{\text{AT}}$ to all its members, e.g., by gossipping [26].

2.6 Private channels to the learner $F_{\text{ST}}$. All parties are furthermore connected to $L$ by secure message transmitting, formalized by the ideal functionality $F_{\text{ST}}$ (as in [27]). It is like $F_{\text{AT}}$, except that the content of the messages sent are not leaked anymore to $A$. Compared to the “$F_{\text{SM}}$” in [15], we do not allow to adaptively corrupt the receiver $L$ of $F_{\text{ST}}$ 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 $F_{\text{Sch}}$, to send at any point a signal refreshsig to any shareholder $P$. In the model of ephemeral roles (“yoso” [39]) which we consider, this signal imposes $P$ to send at most one batch of messages, to the next collector committee, then shut-off itself. This means that $P$ erases all its memory and quits the protocol. Likewise, when some $P$ receives open sig, it must send at most one message, to $L$ over $F_{\text{ST}}$, then shut-off itself.

2.8 Bulletin board PKI: $F_{\text{CA}}$. We consider the classical model of an entity, denoted as certification authority $F_{\text{CA}}$ in [16]. Each party $P_i$ in any committee $C^{(e)}$ can give one public key of its choice to $F_{\text{CA}}$. Then after $A$ allows it, $F_{\text{CA}}$ publishes the key. Hence, since $F_{\text{CA}}$ does not perform any check of knowledge of a secret key, it also known as “bulletin board” PKI or “bare” PKI [canetti14]. In our simple model, $F_{\text{CA}}$ also publishes the identity of the issuer of the key. But Y-VSS seamlessly carries over the model [10] where $F_{\text{CA}}$ would simply publish that the key belongs to the “i-th role of $C^{(1)}$ “.

2.9 Terminating reliable broadcast from $D$ to $C^{(1)}$. We allow a single call to the following ideal broadcast functionality $F_{\text{BC}}^{D,C^{(1)}}$, dubbed as $F_{\text{BC}}$. If $D$ is honest, $F_{\text{BC}}$ waits to receive some value $x$ from $D$, then delay-outputs $x$ to every party in $C^{(1)}$. If $D$ is corrupt, $F_{\text{BC}}$ requests from $A$ a value, then, upon receiving some $y$ from $A$, delay-outputs $y$ to each party in $C^{(1)}$. In a model where $D$ would always be honest, then $F_{\text{BC}}^{D,C^{(1)}}$ 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 $F_{\text{NIZK}}$, defined in [44] 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 [60].

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, since it relaxes the randomness parameter can be specified in the protocol). Definition 2 may be of independent interest, since it relaxes the requirement of correctness mod $p$ after unlimited linearly homomorphic operations, e.g., as in the resharing scheme 20. This enables to instantiate PKE from strictly more schemes, as the Paillier mod $p$, and also the el-Gamal-in-the-exponent, as described in Section 7. The perfect correctness guarantee implies that the decryptor $P$ is bound 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 $\rho_{key}$, 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 39, 10, 38. 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 65 (committing encryption). We illustrate in Appendix A how the specified ranges can be enlarged, in order to allow slack in the NIZKs of range.

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 $\text{KeyGen}(\|\rho_{key}\in\mathcal{R}_{\text{key}})\rightarrow\mathcal{K}_{\text{key}}\times\mathcal{P}_{\text{秘}},$ where $\mathcal{R}_{\text{key}}$ is the space of key randomness. The ciphertext space is denoted as $\mathcal{C}$, and the randomized encryption function as $\text{Enc}:(pk\in\mathcal{P}_{\text{秘}},x\in\mathbb{Z}|\rho_{enc}\in\mathcal{R}_{\text{enc}})\rightarrow\mathcal{C},$ where $\mathcal{R}_{\text{enc}}$ is the space of encryption randomness. By convention, any $pk\notin\mathcal{P}_{\text{秘}}$ is denoted as $\bot$, and encryption under $\bot$ returns $\bot$. 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 mod $p$ after one homomorphic linear combination of size $t+1$). We consider functions denoted as the decryption mod $p$ $\text{Dec}:(sk\in\mathcal{S}_{\text{秘}},c\in\mathcal{C})\rightarrow(F_{p},\bot);$ the homomorphic addition $\boxplus:\mathcal{C}\times\mathcal{C}\rightarrow\mathcal{C}$ and scalar multiplication $\boxtimes:F_{p}\times\mathcal{C}\rightarrow\mathcal{C}$.

(a) For any key randomness $\rho_{key}$, (sk, pk) ← KeyGen(\|\rho_{key}); for any up to $t+1$ plaintexts $(x_{i})_{i\in[t+1]})\in F_{p}^{t+1}$, randomnesses $(\rho_{enc,i})_{i\in[t+1]}\in\mathcal{R}_{\text{enc}}^{t+1}$ and coefficients $(\lambda_{i})_{i\in[t+1]}\in F_{p}^{t+1}$, we require:

$$\text{Dec}(sk,\boxplus(\lambda_{i}\boxtimes\text{Enc}(x_{i} | \rho_{enc,i})) = \sum_{i=1}^{t+1}\lambda_{i}x_{i} \mod p.$$ (2)

Without loss of generality we consider $\boxplus$ and $\boxtimes$ to be deterministic (when not, as, e.g., in 21, 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 mod $p$ after unlimited linearly homomorphic operations, e.g., as in the resharing scheme 20. This enables to instantiate PKE from strictly more schemes, as the Paillier mod $p$, and also the el-Gamal-in-the-exponent, as described in Section 7. The perfect correctness guarantee implies that the decryptor $P$ is bound 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 $\rho_{key}$, 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 39, 10, 38. 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 65 (committing encryption). We illustrate in Appendix A 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:

Share$(s\in\mathbb{F}_{p})$: sample a random polynomial $\tilde{f}$ 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 $s_{i} := \tilde{f}(i)$ $\forall i \in [n]$.

We then denote $\tilde{f}$ as the (unique) sharing polynomial defining $s_{[n]}$. We have the $t$-privacy property that, for a fixed $s$, any $t$-sized subset $Z \subset [n]$ of coordinates $(s_{i})_{i\in Z}$ output by Share$(s)$, vary uniformly at random in $\mathbb{F}_{p}$. 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 $\lambda_{i}^{\mathcal{U}} := \prod_{j\in\mathcal{U} \setminus\{i\}}\frac{1}{i-j}, \forall i \in \mathcal{U}$, then the following linear map, denoted as Lagrange combination:

$$\mathcal{L}$-combine$^{\mathcal{U}}: (s_{i})_{i\in\mathcal{U}} \rightarrow \sum_{i\in\mathcal{U}}\lambda_{i}^{\mathcal{U}}s_{i}$$ is such that $s = \mathcal{L}$-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 $(pk_{i})_{i\in[n]} \in (\mathcal{P}_{\text{秘}} \cup \bot)^{n}$. In order to make compact statements, we introduce some useful terminology. For $pk\in\mathcal{P}_{\text{秘}}$, we say that $sk\in\mathcal{S}_{\text{秘}}$ is an explainable secret key of pk if there exists $\rho_{key}\in\mathcal{R}_{\text{key}}$ such that $(sk,pk) = \text{KeyGen}(\|\rho_{key})$. We say that $\tilde{x}\in\mathbb{F}_{p}$ is an explainable decryption of $c$ under some $pk\in\mathcal{P}_{\text{秘}}$, if there exists an explainable secret key $sk$ of $pk$ such that $\text{Dec}(sk,c) = \tilde{x}$. A proof of correct decryption of $c$ into $x\in\mathbb{F}_{p}$ under $pk$, shortened as $\pi_{pk}$ (which stands for publicly verifiable decryption), is a NIZK AoK of such a $sk$. In particular, $\pi_{pk}$ implies that $x$ is an explainable decryption of $c$ under $pk$.

Definition 3 (Ciphertext shares, opening shares and threshold opening). We say that $c_{[n]} := (c_{j})_{j\in[n]} \in \mathcal{C}^{n}$ 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 $\tilde{f} \in F_{p}[X]_{\leq t}$ such that, for every opening share $s_{j}$, $s_{j} = \tilde{f}(j)$. 7
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 3.1 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 \(\text{ShSim}\). On input any \(s \in \mathbb{F}_p\) and \((s_i)_{i \in \mathcal{V}} \in \mathbb{F}_p^{|\mathcal{V}|}\) for some \(t\)-sized subset \(\mathcal{V} \subset [n]\), \(\text{ShSim}\) 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 \(\text{ShInf}\), which does the task of inferring the opening shares of corrupt parties (virtual ones) from \(t+1\) opening shares of honest parties. Namely, on input any \((s_i)_{i \in \mathcal{U}} \in \mathbb{F}_p^{|\mathcal{U}|}\) for a \(t+1\)-subset \(\mathcal{U} \subset [n]\), \(\text{ShInf}\) 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 \(c_{[n]} = (c_i)_{i \in [n]} \in \mathbb{C}^n\) 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 opening shares \((s_i)_{i \in \mathcal{N}}\) are defined, then

- \(\forall \mathcal{N} \subset [n]\) of size \(t\), \((s_i)_{i \in \mathcal{N}} \subseteq \text{ShSim}(s)\), \((s_j)_{j \in \mathcal{Z}}\);
- \(\forall \mathcal{U} \subset [n]\) of size \(t+1\), \((s_j)_{j \in [n] \setminus \mathcal{U}} = \text{ShInf}((s_i)_{i \in \mathcal{U}})\).

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 \((pk'_j)\) 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 \(t+1\), \((f_i)_{i \in \mathcal{U}}\), \(s_{i \rightarrow j} := f_i(j)\), \(\forall i \in \mathcal{U}, j \in [n]\) and any encryption randomnesses \((\rho_{\text{enc}, i})_{i \in \mathcal{U}}\). Then, consider the vectors of encryptions of these re-sharings: \(c_{i \rightarrow j} := (\text{Enc}(s_{i \rightarrow j} | \rho_{\text{enc}, i}))\) \(\forall i \in \mathcal{U}, j \in [n]\), 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 \(s\) at least \(t+1\) keys \(pk_j\) are explainable, then \(c'_{[n]}\) has sharing polynomial equal to \(f := \sum_{i \in \mathcal{U}} \lambda_i f_i\) 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 \(C'\) does not need to speak at the end of a Refresh. To enable this, we somehow “pipeline” the validation messages of \(C'\), into the messages which \(C'\) 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 sharesig), every member of an exiting committee \(C\) (or the dealer \(D\) ) aborts if it cannot retrieve from the PKI a complete list of \(n\) public keys in \(\mathbb{S}_N\), for the entering committee \(C'\). 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 \((pk_i)\) in \([n]\) and \((pk'_i)\) in \([n]\) for some exiting \(C\) and entering \(C'\). The assumption implies that all \(t+1\) honest parties published their keys.

4.1 Y-VSS Share

The dealer \(D\) 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 \(c_{[n]} := (\text{Enc}_{pk_i}(s_i))_{i \in [n]}\) of the shares under the public keys of the parties of the first committee \(C^{(1)}\). It also generates a NIZK AoK, denoted \(\pi_{\text{PVSS}}\) of: \(s\), the sharing polynomial \(f\), and the encryption randomnesses explaining \(c_{[n]}\). Following the tradition, we call such a pair \((c_{[n]}, \pi_{\text{PVSS}})\) a *publicly verifiable sharing*, denoted as a \(\text{PVSS}\) and further formalized in Appendix B.3. Finally, \(D\) broadcasts to \(C^{(1)}\) \((c_{[n]}, \pi_{\text{PVSS}})\) appended with its signature \(\sigma_D\) on it. For consistency with the rest of the protocol, we denote this triple as \(\text{ppS}_{D} := (\sigma_D, \pi_{\text{PVSS}} \mathbb{S}_N c_{[n]}\) and call it as a *proven proactive sharing* relatively to \(C^{(1)}\). If parties of \(C^{(1)}\) are delivered something else from the broadcast, which can happen
if \( D \) is corrupt, then they replace it by a pre-defined default PPS of \( 0 \), denoted as \( \text{pps}_0 = \{ 1, 0 \} \). In Sections 4.2.1 and 4.2.2 we describe the outputs of \( C' \) and inputs of \( C \). In Section 4.2.3 we describe the first step, which is the multiscats by each parties of \( C \) to the committee \( C'_{\text{collec}} \) of the \( t+1 \) “collectors”. In Section 4.2.4 we describe the second step, in which each collector responsively multicasts a PPS, to \( C' \). Then in Section 4.2.5 we formalize the reception by \( C' \) of these PPS and their appending to their lists of outputs.

4.2 Y-VSS Refresh(\( C, C' \))

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

4.2.1 Outputs of \( C' \). Each party \( P'_j \) of the entering committee \( C' \) starts with an empty state, initializes an empty list denoted \( \text{ListOfPps}_i \), and progressively adds up to \( t+1 \) objects in this list, denoted as \( \text{proven proactivized sharings} \), shortened as PPS. Broadly speaking, a PPS relatively to committee \( C' \), consists of a vector of ciphertext \( c'_j[n] \in \mathbb{G}^n \) which is instructed to open the secret to a “collector”. In more details, a PPS relatively to \( C' \) is a tuple of the form \( \text{pps}' = (c[n], \text{qvc}, \{ \text{pvpr}_i \}_{i \in \mathcal{U}}) \), and denoted as

\[
\text{pps}' := (\text{vps} := (c[n], \text{qvc}, \{ \text{pvpr}_i \}_{i \in \mathcal{U}} \mathcal{L} \alpha c'[n]),
\]

- \( c[n] = (c_i)_{i \in \mathcal{N}} \) is a vector of ciphertexts and \( \text{qvc} \) is the concatenation of signatures on \( c[n] \) issued by \( t+1 \) out of \( n \) parties of \( C \). We denote \( \text{qvc} \) as a \( \text{quorum verification certificate} \) because, roughly, each signer attests that \( c'[n] \) was itself formed by applying the baseline method in the previous refresh, when \( C \) was entering. Hence, we denote such a pair \( \text{vps} = (c[n], \text{qvc}) \) as a \( \text{verified proactivized sharing} \), dubbed as a VPS. We say that two VPSs are the same as soon as their \( c[n] \) are the same, notwithstanding their \( \text{qvc} \) can have different sets of signers;
- \( \mathcal{U} \subset \mathcal{N} \) is a subset of size \( t+1 \);
- \( \text{pvpr}_i = (c_i, \pi_{\text{pvpr}}, \iota') \), for \( i \in \mathcal{U} \), is what we denote as a \( \text{publicly verifiable re-sharing} \) of the coordinate \( c_i \) of the \( i \)-th party \( \text{P}_i \) in \( C \). It consists of a vector of ciphertexts \( c_i[n] \in \mathbb{G}^n \) and of a NIZK AoK, denoted as \( \pi_{\text{pvpr}, i} \) of: \( \text{P}_i \)'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 randomness, proving that \( c_i[n] \) is an encrypted sharing of \( s_i \) under the keys of \( C' \). \( \text{pvpr}s \) are further formalized in Appendix B.3.
- \( c'[n] = \bigoplus_{i \in \mathcal{U}} c_i[n] \), exactly as in Equation (1). Although \( c'[n] \) is not actually included in the \( \text{pps}' \), we display it because it is publicly efficiently computable from \( \text{pps}' \).

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

4.2.3 Encrypted re-sharing. Upon receiving the signal \( \text{refreshsig} \), each party \( \text{P}_i \in C \), for every PPS: \( \text{pps} := (\ldots, \mathcal{L} \sigma c_i[n] = (c_i)_{i \in \mathcal{N}} \) in its list \( \text{ListOfPps}_i \), forms a triple \( (c[n], \text{pvpr}_i, \sigma_i) \) as follows:
- it generates a \( \text{publicly verifiable re-sharing} \) of its encrypted share \( c_i \), denoted as \( \text{pvpr}_i = (c[n], \pi_{\text{pvpr}, i}) \). Concretely: \( \text{P}_i \) decrypts \( c_i \) into \( s_i \), generates a Shamir sharing \( (s_i, \pi_{\text{pvpr}, i}) \in \mathcal{U} \) of \( s_i \), then encrypts each coordinate \( s_i \) under the public key of \( \text{P}_j \) in \( C' \). Then it obtains the vector of encrypted shares \( c_i[n] := (c_i)_{i \in \mathcal{N}} \). It appends to it a NIZK of correct re-sharing: \( \pi_{\text{pvpr}, i} \). This is depicted in Figure I
- it generates a signature on \( c[n] \), denoted \( \sigma_i \), for which \( P_i \) attests that \( c[n] \) is obtained from a (valid) PPS. \( \text{P}_i \) multicasts to \( C'_{\text{collec}} \), all at once, all the triples obtained, then shuts-off. We denote such a triple as a \( \text{resharing message} \).

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

\[
(4) \quad (\text{vps} := ((c[n], \text{qvc}), (\text{pvpr}_i)_{i \in \mathcal{U}}), \mathcal{L} \alpha c'[n])
\]

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

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

4.3 Y-VSS Opening of a secret

We consider any arbitrary committee \( C^{(e_o)} \), which is instructed to open the secret to a designated learner \( L \). Each party \( P_i \in C^{(e_o)} \), for every \( (\text{pps}^{(e_o)} := (\ldots, \mathcal{L} \sigma c[n] = (c[n])_{i \in \mathcal{N}} \) in its list \( \text{ListOfPps}_i \) (containing up to \( t+1 \) elements):
- generates a decryption \( s_i \) of its encrypted share \( c_i \), which we dub as an \( \text{opening share} \), generates a NIZK of correct decryption under its public key, denoted as \( \pi_{\text{pvpr}, i} \), to obtain the triple \( (c[n], s_i, \pi_{\text{pvpr}, i}) \). It privately sends to \( L \) all the (up to \( t+1 \)) triples obtained. Upon receiving any \( t+1 \) such triples for some same \( c[n] \), \( L \) outputs the Lagrange linear combination of the \( (s_i) \).

5. Analysis of Y-VSS

In Theorem \( \theta \), we state security and liveness of Y-VSS. Since the proof of liveness is even simpler than the ones sketched, in Section 3.3 for the simpler variant in \( \delta \), we defer it to Appendix D.2 In Proposition \( \theta \), we state and prove a useful intermediary property of Y-VSS denoted as \( \text{correctness} \), which is happens to be synonymous of “public verifiability” \( \text{(4)} \). We defer to Appendix D.3 the proof of UC security when the dealer \( D \) 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-\( \text{VSS} \) in the \((\mathcal{F}_{BC}, \mathcal{F}_{CA}, \mathcal{F}_{NIZK}, \mathcal{F}_{AT}, \mathcal{F}_{ST}, \mathcal{F}_{SSH})\)-hybrid model is such that:

**Security** it UC-implements \( \mathcal{F}_{SSH} \), 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 [17], for some \( 1 \leq e_0 \), we say that an execution is complete up to \( e_0 \), 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 \( C^{(1)} \);

- [LK] for all \( e \in \{1, \ldots, e_0 - 1\} \) (possibly empty), then all honest parties of \( C^{(e+1)} \) receive startsig and \( \mathcal{A} \) allows the publication of their keys on \( \mathcal{F}_{CA} \). \( \mathcal{CA} \) does not publish any further key after the first party in \( C^{(e)} \) received refreshsig. This implies that, upon receiving refreshsig, all parties in \( C^{(e)} \) are able to recover from \( \mathcal{F}_{CA} \) the list of public keys, including all those of honest parties.

- [LR] for all \( e \in \{1, \ldots, e_0 - 1\} \) (possibly empty), after all honest keys of \( C^{(e+1)} \) were published on \( \mathcal{F}_{CA} \), but at least \( 2\delta \) after the last party of \( C^{(e-1)} \) received refreshsig, then \( \mathcal{F}_{Sch} \) sends refreshsig to each party \( \mathcal{P}_i^{(e)} \in C^{(e)} \), then \( \mathcal{F}_{Sch} \) sends refreshsig to all honest parties in \( C^{(e_0)} \).

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

**Proof.** Let us define \( \tilde{s} \) as: either the threshold opening of the broadcast of the \( \mathcal{D} \), if it is a (valid) pvS, or, \( \tilde{s} := pvS_0 \) 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 \( C^{(e)} \) ever formed in the execution, has its last component \( c_0^{(e)} \) which is a vector of ciphertext shares in the sense of Definition 3 and which has threshold opening equal to \( \tilde{s} \).

Indeed, when \( \mathcal{L} \) receives \( t+1 \) triples for the same \( c^{(e_0)}_0 \), then since one of the senders is honest, it must be that \( c^{(e_0)}_0 \) was formed (“\( \mathcal{L} \)”) out of a PPS relatively to committee \( C^{(e_0)} \). 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 \( C := C^{(e+1)} \), which will conclude the recursion. Let us consider one such PPS \( pps' \). Its first component is a VPS: \((c_0^{(e)}, qvc)\), so since there is at least one honest signer, this proves that \( c_0^{(e)} \) was formed (“\( \mathcal{L} \)”) out of a PPS relatively to committee \( C := C^{(e)} \). Hence \( c_0^{(e)} \) is a vector of ciphertext shares of \( \tilde{s} \) by the recursion assumption. Let us denote as \( (s_i)_i \in [n] \) its opening shares. Finally, consider the \( t+1 \) publicly verifiable resharing: \( pvR_i = (c_0^{(e)}_0, \pi_{pvR}.i) \) \( \forall i \in U \) enclosed in \( pps' \). For each \( i \in U \), the NIZK AoKs \( \pi_{pvR}.i \) guarantees that \( c_0^{(e)}_0 \) is of the form \( \{Enc_{pvR}(s_{i-1,j} \mid \gamma_{enc,i,j})\}_j \in [n] \), where the \( (s_{i-1,j})_j \in [n] \) form a vector of shares of \( s_i \). Hence, by Property 8 we conclude that \( c_0^{(e)}_0 \) is a vector of ciphertext shares, with threshold opening equal to the same \( \tilde{s} \).

\[ \Box \]

5.1 Proof of UC security in case of a honest dealer \( \mathcal{D} \).

We start by considering a real execution \( \text{REAL}_A \) of Y-\( \text{VSS} \), with adversary \( A \) 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 show indistinguishable from one with the next, from the point of view of \( \mathcal{E} \). In the final game, denoted as \( \text{Hyb}^{\text{Share}} \), the view of \( A \) 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 \( \mathcal{F}_{VSS} \), which delivers the actual value of \( s \), which helps us to simulate the opening shares of \( s \). So what we are describing in \( \text{Hyb}^{\text{Share}} \) is a simulator which interacts only with \( \mathcal{F}_{VSS} \) and \( \mathcal{E} \), which concludes the UC proof.

The purpose of the games \( \text{Hyb}^{\text{Share}} \) and \( \text{Hyb}^{\text{Open}} \), 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 \( C^{(e_0)} \), nor using the plaintext shares of honest parties of \( C^{(e_0)} \). This allows to apply IND-CPA of PKE in subsequent games. In particular in \( \text{Hyb}^{\text{Open}} \) we achieve that all re-sharings are actually mere pvSs.

**Game \( \text{REAL}_A \).** This is the actual execution of the protocol Y-\( \text{VSS} \) with environment \( \mathcal{E} \), adversary \( \mathcal{A} \) and ideal functionalities \( \mathcal{F}_{BC}, \mathcal{F}_{CA}, \mathcal{F}_{NIZK}, \mathcal{F}_{AT}, \mathcal{F}_{ST}, \mathcal{F}_{Sch} \). We make the change (not formalized by a game) that \( \mathcal{F}_{NIZK} \) 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 \( \mathcal{F}_{NIZK} \) in the actual protocol.

**Game \( \text{Hyb}^{\text{Share}} \).** Unchanged if \( \mathcal{L} \) honest. For each PPS: \( pps' = ((c_0^{(e)}_0, qvc), \{pvR_i\}_{i \in U}, c_0^{(e)}_0) \) opened by \( (c_0^{(e)}_0) \) to \( \mathcal{L} \), the opening shares of honest parties in \( C^{(e_0)} \) are now computed as \( \text{ShSim}(s_{c_0^{(e)}_0}, s_{c_0^{(e)}_0}) \in \mathcal{D} \), where:

- \( s_{c_0^{(e)}_0} \) is the threshold opening of \( c_0^{(e)}_0 \);

- \( \{i\} \subset [n] \) are the t indices of corrupt parties in \( C^{(e_0)} \);

- \( \{s_{j}^{(e)}\}_{j \in \mathcal{I}} \) are the opening shares of \( c_0^{(e)}_0 \) of corrupt parties, computed as follows. Each \( pvR_i = (c_0^{(e)}_0, \pi_{pvR,i}) \) \( \forall i \in U \), must be of the form \( \{Enc_{pvR}(s_{j-1,j} \mid \gamma_{enc,i,j})\}_j \in [n] \), where \( (s_{j-1,j})_j \in [n] \) is a vector of shares of \( s_j \). For a honest party this is automatic since it follows Y-\( \text{VSS} \). For a corrupt party this is guaranteed by the NIZK \( \pi_{pvR,i} \). We finally set \( s_{j}^{(e)} := \sum_{i \in U} \lambda_i^{(j)} \) \( \forall j \in \mathcal{I} \).

**Claim 7.1.** \( \text{REAL}_A = \text{Hyb}^{\text{Share}} \).
By the If case of Property 5, the method to compute the \((s_j')\) indeed returns the opening shares of \(c'_i[n]\). Then, by Property 4, \(\text{ShSim}\) does return the opening shares of \(c'_i[n]\) with indices in \([n] \setminus \mathcal{I}\).

**Game Hyb\textsuperscript{aOpen}.** Unchanged is \(L\) honest. Else (if \(L\) corrupt), this game differs from Hyb\textsuperscript{ShSim} in that the input 

\[
s_{c_i[n]} \text{ to } \text{ShSim} \text{ is replaced by the actual secret } s \text{ of the dealer } \mathcal{D}.
\]

**Claim 7.2.** Hyb\textsuperscript{ShSim} \(\equiv \) Hyb\textsuperscript{aOpen}.

**Proof.** By Proposition 7 for each PPS: \(\text{pps}' = (\ldots, c_i'[n])\) relatively to commit \(C(c_i)\), we have that \(c_i'[n]\) is a vector of ciphertext shares with threshold opening equal to \(s\).

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

**Games Hyb\textsuperscript{bRefresh} [e, i] for each \(e \in [e_0 - 1, \ldots, 1]\) (in this backwards order) then each \(i \in \{0, \ldots, n\}\).**

We set \(\text{Hyb}^{\text{bRefresh}}[e, 1] : = \text{Hyb}^{\text{ShSim}}\). Then for each \(e \in [e_0 - 1, \ldots, 1]\) and \(i \in \{0, \ldots, n\}\): if \(P(e)_{i + 1}\) is corrupt then we leave \(\text{Hyb}^{\text{bRefresh}}[e, i + 1] : = \text{Hyb}^{\text{bRefresh}}[e, i]\) unchanged, otherwise if it is honest, then we modify \(\text{Hyb}^{\text{bRefresh}}[e, i]\) into \(\text{Hyb}^{\text{bRefresh}}[e, i + 1]\) as follows. For each \(e\) \(\in \{0, \ldots, n\}\), in the local list of \(P(e)_{i + 1}\), in place of the re-sharing \(c_i'[n]\) of \(c_i(e)\) that \(P(e)_{i + 1}\) generates, we substitute the vector of ciphertexts \(c_i'[n] = \text{Enc}_{pk_i}(s_j)_{j \in [n]}\), where the \(t\) shares of the corrupt indices \((s_j)_{j \in \mathcal{I}}\) 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 \(\mathcal{F}_{\text{NIZK}}\) 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 \(\text{Hyb}^{\text{bRefresh}}[e - 1, 1] : = \text{Hyb}^{\text{bRefresh}}[e, n]\).

**Claim 7.3.** Hyb\textsuperscript{aOpen} \(\equiv \) Hyb\textsuperscript{bRefresh}[1, n]

**Proof.** It is enough to show that for each \(e \in [e_0 - 1, \ldots, 1]\), for \(i \leq n - 1\) such that \(P(e)_{i + 1}\) is honest, then \(\text{Hyb}^{\text{bRefresh}}[e, i]\) is indistinguishable from \(\text{Hyb}^{\text{bRefresh}}[e, i + 1]\). Let us consider one of the PPSs: \(\text{pps}' = (\ldots, \{\text{ppv}_i\}_i \in \mathcal{U}, c_i'[n])\) held by at least one honest party in \(C(c_i)\). We now change the method to infer the opening shares of \(c_i'[n]\) of corrupt parties fed into ShSim. Precisely, in the method in Hyb\textsuperscript{ShSim}, for each \(\text{ppv}_i, i \in \mathcal{U}\) (if any) generated by a corrupt party \(i\) in \(C(c_i - 1)\), we now change the method to infer the opening shares of \(c_i'[n]\) of corrupt parties fed into ShSim. We then use the secret keys of the \(t + 1\) honest parties in \(C(c_i)\) to correctly decrypt their opening shares \((s_{i - j})_j \in \mathcal{I}\) of \(c_i'[n]\). Then we use them as inputs of ShInfer, i.e., we do a polynomial interpolation from \(t + 1\) points, to obtain the desired \(t\) opening shares \((s_{i - j})_j \in \mathcal{I}\) of corrupt parties, of \(c_i'[n]\). We claim that the \((s_{i - j})_j \in \mathcal{I}\) obtained are unchanged. Indeed, the \((s_{i - j})_j \in \mathcal{I}\) used are by definition opening shares of \(c_i'[n]\). So since \(c_i'[n]\) is a vector of ciphertext shares,
then the Lagrange interpolation which we did (formalized by ShInfer in Property 5) does return \((s_i \rightarrow j)_{j \in Z}\).

6. Our Implementation, using Elgamal

The plaintext space is not \(Z\) but instead an abelian group, denoted \(G\) 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 \((Z/qZ)^m \times G^n\). Thus the relation to be proven in NIZK for re-sharing 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, [20] Fig. 10] independently formalized NIZKs of re-sharing 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 re-sharing denoted “DHPVSS” in [20] p. 5.2] (provided \(\deg(m^t) \leq n-t-1\)), 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 Refresh\((C, C')\) between committees \(C\) and \(C'\), with number of parties from 11 to 101. Namely, we measure the time, in terms of local computations, required to re-share a secret from a committee \(C\) to a new committee \(C'\). 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 \(C\) does \(t+1\) re-sharings, then each player in \(C'\) receives \(t+1\) PPSs. In-between, we consider that each collector \(K'_k\) verifies only one batch of \(t+1\) re-sharing messages. Indeed, assume that \(K'_k\) receives \(t+1\) re-sharing messages out of the same VPS: \((c_{[ni]}, qvc)\) but which do not pass its verification. This means for \(K'_k\) that this batch contains a proof of openly mis-behaving of some party in \(C\), 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 \(C'\), 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 Refresh\((C, C')\) between two committees \(C\) and \(C'\), 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 \(C\) 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 \(C'\), for each of the \(t+1\) PPS which it receives (we consider the worst-case): of verification of the \(t+1\) NIZKs of re-sharing \((\pi_{v_{ps}}, (c_{[ni]}))_{i \in I})\), of the \(t+1\) signatures qvc on \(c_{[ni]}\), and of applying the homomorphic Lagrange linear combination to obtain \(c_{[ni]}\).

**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 re-sharing a ciphertext and for producing a NIZK proof, NIZK.Verify the verification of a NIZK proof of re-sharing and Lagrange Comb the combination of \(t+1\) resharing 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.

<table>
<thead>
<tr>
<th># of players</th>
<th>11</th>
<th>21</th>
<th>51</th>
<th>101</th>
</tr>
</thead>
<tbody>
<tr>
<td>pvReshare</td>
<td>6.1 ms</td>
<td>11.1 ms</td>
<td>27.6 ms</td>
<td>52.3 ms</td>
</tr>
<tr>
<td>NIZK.Verify</td>
<td>5.9 ms</td>
<td>6.34 ms</td>
<td>15.3 ms</td>
<td>30.1 ms</td>
</tr>
<tr>
<td>Lagrange Comb (Fig 4)</td>
<td>4.8 ms</td>
<td>17.1 ms</td>
<td>100.8 ms</td>
<td>433.0 ms</td>
</tr>
<tr>
<td>Sign</td>
<td>0.74 ms</td>
<td>0.79 ms</td>
<td>0.95 ms</td>
<td>1.21 ms</td>
</tr>
<tr>
<td>Sign.Verify</td>
<td>1.4 ms</td>
<td>1.5 ms</td>
<td>1.6 ms</td>
<td>1.9 ms</td>
</tr>
</tbody>
</table>

**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 \(F_p := Z/pZ\), and thus when the plaintext
which sends the same PPSK each

7.4 Linear combinations / multiexponentiations of secrets over rings / groups. The baseline method since [3] 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 \(K_i^{(e_{\phi})}\) for each \(K_i^{(e_{\phi})}\in \sigma_{\text{coll}}\), computes the homomorphic linear combination of the PPSs of these secrets which it received from \(K_i\in K_{\text{coll}}\). Then it sends the decryption to \(\mathcal{L}\), appended with the PPSs (all at once for all \(K_i^{(e_{\phi})}\)). Liveness follows from the fact that there is at least a honest \(K_i^{(e_{\phi})}\in \sigma_{\text{coll}}\) which sends the same PPSs to all parties in \(\sigma_{\text{coll}}\).

We make the observation that the baseline extends to opening the image by a group homomorphism \(\varphi\), 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 \(\varphi\) 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 \(\varphi\), which commute with \(\varphi\). Namely, which are such that threshold opening-then-\(\varphi\) equals \(\varphi\) 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 [8] 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 [70, 62, 29, 17, 3, 68, 66, 69], including “Opt-CHURP” [55]. 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 PSSs 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 \(\Delta_{BC}\). 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 \(\Delta_{BC}\), then Exp-CHURP-A [55, 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 [42] 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 \(\Delta_{BC}\) assumption.

References


The communication complexity is the total number of bits sent by honest parties in both the exiting and entering committee to complete a Refresh, $n|BC(n)$, denote the bitsize needed to implement $n$ broadcasts resp. $n$ multicasts, each with input size $O(n)$ resp. $O(n^2)$. As reported in [63], 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 [48], in $|BC(1)|=O(kn^4)$. Known amortization techniques would improve this into $|BC(\ell)|=O((kn^4+n^4k)$ asymptotically in $\ell$.

(2) $2\delta$ denotes 2 consecutive message deliveries, $5BC$ denotes 5 consecutive broadcasts and $\delta_{BC}$ is the publication delay on the PKI.

(3) The $|BC(n)|$ complexity of [42] 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-shop, 2004.

(5) SRS designates a structured random string, needed for KZG. This assumption could be downgraded in [42, 55] if specifying another polynomial parameter for the public keys.

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

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Network</th>
<th>Latency</th>
<th>Communication</th>
<th>Dynamic</th>
<th>Yoso</th>
<th>Setup</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSS [17]</td>
<td>Synch</td>
<td>$\delta_{BC} + 2\delta_{PV}$</td>
<td>$n</td>
<td>BC(n)</td>
<td>$</td>
<td>No</td>
</tr>
<tr>
<td>groth [10]</td>
<td>Synch</td>
<td>$\delta_{BC} + \Delta_{BC}$</td>
<td>$n</td>
<td>BC(n)</td>
<td>$</td>
<td>Yes</td>
</tr>
<tr>
<td>Exp-CHURP-A [55]</td>
<td>Synch</td>
<td>$\delta_{BC} + 5\Delta_{BC}$</td>
<td>$n</td>
<td>BC(n)</td>
<td>$</td>
<td>Yes</td>
</tr>
<tr>
<td>DPSS [42]</td>
<td>Synch</td>
<td>$5\Delta_{BC}$</td>
<td>$</td>
<td>MC(n^2)</td>
<td>$</td>
<td>Yes</td>
</tr>
<tr>
<td>Y-SSS</td>
<td>Asynch</td>
<td>$\delta_{BC} + 2\delta$</td>
<td>$n</td>
<td>MC(n^2)</td>
<td>$</td>
<td>Yes</td>
</tr>
</tbody>
</table>


[63] 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).


Appendix

A Complements on the model

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

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

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

We present in Algorithm 8 the ideal functionality of a non-interactive zero-knowledge arguments of knowledge, $\mathcal{F}_{NIZK}$ is parameterized 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 $(x, w) \in \mathcal{R}$ then deletes $w$ from its memory. If the verification passes, then $\mathcal{F}_{NIZK}$ delay-outputs to $P$ a string $\pi$. Upon subsequent input the same string $\pi$ and $x$ from any verifier, $\mathcal{F}_{NIZK}$ then confirms to the verifier that $P$ knows some witness for $x$. We denote $\Pi$ the space of such strings $\pi$.

$\mathcal{F}_{NIZK}$

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

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

**Verification:** On input $(verify, sid, ssid, x, \pi)$ from any party $V$, check whether $(x, \pi)$ 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 $(x, w) \in \mathcal{R}$ and in that case, store $(x, \pi)$. If $(x, \pi)$ is stored, return (verification, sid, ssid, 1) to $V$, else return (verification, sid, ssid, 0).

Algorithm 8: Non-interactive zero-knowledge functionality

A.2 Full modelisation of $\gamma$-$VSS$ 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” $\mathcal{A}$. 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 $s_{out} \in {\mathbb{F}_p}$, then it immediately informs $\mathcal{E}$ of $s_{out}$.

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

- $REAL_{A}$: an actual execution of the protocol $\Pi$, with adversary $\mathcal{A}$ fully controlled by $\mathcal{E}$, and functionalities $\mathcal{F}_{CA}$, $\mathcal{F}_{Sch}$, $\mathcal{F}_{BC}$, $\mathcal{F}_{ST}$, $\mathcal{F}_{AT}$, $\mathcal{F}_{NIZK}$, as depicted in Figure 9.

- IDEAL-$\gamma_{VSS}$: 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 $\mathcal{F}_{\gamma_{VSS}}$ 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 $\mathcal{F}_{\gamma_{VSS}}$. Concretely, they perform what is commonly called as the dummy protocol, which consists in $\mathcal{D}$ (if honest) gives its input to $\mathcal{F}_{\gamma_{VSS}}$, then $\mathcal{L}$ (if honest) outputs what it receives from $\mathcal{F}_{\gamma_{VSS}}$, while parties on their side do nothing.

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

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

An alternative model of proactivity By contrast, in the later work [6], proactivity is captured at the level of the functionality. 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 refresh$\_\text{sig}$ to some party $P$, the scheduler can shut-it-off after all chains of $\ell$ consecutive asynchronous events completed, in a sense which can be be made precise from [11]. There, $\ell$ 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 pv$\text{S}$hare and resharing pv$\text{R}$ehare, we formalize the data structure outputs, which are publicly verifiable sharing (pv$\text{S}$) and resharing (pv$\text{R}$), 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 $U \subset [n]$ and, for each $i \in U$, any vector of shares $(s_{i \rightarrow j})_{j \in [n]}$ 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:

\begin{equation}
\text{s'}_{[n]} := \mathcal{L}\text{-combine}^U((s_{i \rightarrow j})_{i \in U}) \text{ is a sharing of } s,
\end{equation}

where, concretely, $s'_{[n]} := (s'_j := \sum_{i \in U} \lambda_i^U s_{i \rightarrow j})_{j \in [n]}$. The proof is because, consider the (re-)sharing polynomials: $(f_i(X))_{i \in U}$ defining the $(s_{i \rightarrow j})_{i \in U}$, then, since polynomial evaluation commutes with linear combinations, the new shares $(s'_{i \rightarrow j})_{j \in [n]}$ are the evaluations of $f'(X) := \sum_{i \in U} \lambda_i^U f_i(X)$, but by construction $f'(0) = s$. An illustration is provided in Figure 10.

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

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

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

For $c \in \mathcal{C}$, we say that $x \in \mathcal{F}_p$ is an explainable plaintext of $c$ under some $pk \in p\mathcal{K}$, if there exists $\rho_{\text{enc}} \in \mathcal{R}_{\text{enc}}$ such that $c = \text{Enc}(pk, x | \rho_{\text{enc}})$.

Definition 8. Perfect correctness mod $p$ after one homomorphic linear combination of size $t+1$ is the guarantee that, for any $pk \in p\mathcal{K}$, for any up to $t+1$ ciphertexts $(c_i)_{i \in [t+1]} \in \mathcal{F}_p^{t+1}$, any explainable plaintexts $(\tilde{x}_i)_{i \in [t+1]} \in \mathcal{F}_p^{t+1}$ of them under $pk$, and any $(\lambda_i)_{i \in [t+1]} \in \mathcal{F}_p^{t+1}$, then, for any explainable decryption $\tilde{y}$ under $pk$ of $\boxplus_{i \in [t+1]} (\lambda_i \boxplus c_i)$, we have $\tilde{y} = \sum_{i=1}^{t+1} \lambda_i \tilde{x}_i$.

Notice that $\tilde{y}$ 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 $\tilde{y}$ 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($pk \in p\mathcal{K}, sk \in \mathcal{K}, c \in \mathcal{C}$) $\rightarrow (\mathcal{F}_p, \Pi)$ the publicly verifiable decryption algorithm. It outputs $x := \text{Dec}(sk, c)$ along with a NIZK AoK, denoted as $\pi_{\text{pvDec}}$, of a secret key sk of $pk$, such that $x = \text{Dec}(sk, c)$. We denote such $\pi_{\text{pvDec}}$ 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. A $\mathcal{D}(pk \in p\mathcal{K}, c \in \mathcal{C})$, called publicly verifiable decryption of $c$ under key $pk$, is a pair $(x \in \mathcal{F}_p, \pi_{\text{pvDec}} \in \Pi)$ where $\pi_{\text{pvDec}}$ is a proof of correct decryption of $c$ into $x$, under key $pk$. 

Figure 10: IDEAL-$F_{\text{VSS}}$ execution: honest entities perform the dummy protocol with $F_{\text{VSS}}$; the simulator $S$ interacts on the right with $F_{\text{VSS}}$ with the same interface as $A$ and corrupt entities in the dummy protocol, and on the left, interacts with $E$ with the same interface as the dummy adversary in the real protocol.
B.3 Publicly verifiable secret sharing (pvS) and resharing (pvR). We consider as a public parameter a list of public keys $(pk_i)_{i \in [n]} \in (\mathcal{P}K \cup \bot)^n$, which in our use-case will be the ones of the entering committee $C$. 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-lexer, then proceeds as in pvShare. Both algorithms also return proofs of correctness, denoted $\pi_{pvS}$ and $\pi_{pvR}$ and defined below.

- **pvShare** ($s \in \mathbb{F}_p$, $\rho_{\text{enc}} \in \mathbb{R}^n_{\text{enc}}$, $Q \in \mathbb{F}_p[X_i^{(0)}] \rightarrow (\mathbb{C}^n, \Pi)$
  
  Set $s_i := (s+Q)(i) \forall i \in [n]$, output $c_i := \left(\text{Enc}(pk_i, s_i)_{\rho_{\text{enc},i}}\right)_{i \in [n]}$ and $\pi_{pvS}$.

- **pvReshare** ($pk \in \mathbb{P}K$, $sk \in \mathbb{S}K$, $c \in \mathcal{C}$, $\rho_{\text{enc}} \in \mathbb{R}^n_{\text{enc}}$, $Q \in \mathbb{F}_p[X_i^{(0)}] \rightarrow (\mathbb{C}^n, \Pi)$:
  
  set $s := \text{Dec}(sk, c)$, output $c_i := \text{pvShare}(s, \rho_{\text{enc}})$ and $\pi_{pvR}$.

$\pi_{pvS}$ is what we call a proof of plaintext sharing knowledge, it is a NIZK AoK of a $s \in \mathbb{F}_p$, a $\rho_{\text{enc}} \in \mathbb{R}^n_{\text{enc}}$ and a $Q \in \mathbb{F}_p[X_i^{(0)}]$, such that $c_i = \text{pvShare}(s, \rho_{\text{enc},Q})$. $\pi_{pvR}$ is what we define as a proof of plaintext re-sharing knowledge of $c$ into $c_i$, such that $\pi_{pvR}$ is a NIZK AoK of an explainable decryption $\tilde{s}$ of $c$ under $pk$, a $\rho_{\text{enc}}$ and a $Q \in \mathbb{F}_p[X_i^{(0)}]$, denoted as an explainable re-sharing polynomial such that $c_i = \text{pvReshare}(pk, sk, c, \rho_{\text{enc}, Q})$.

The evaluations $s_j := (Q+s)(j)$ are denoted as explainable (plaintext) re-sharing shares.

- a **pvS**, called as a public verifiable secret sharing, is a pair $(c_i \in \mathbb{C}^n, \pi_{pvS} \in \Pi)$ where $\pi_{pvS}$ is a proof of plaintext secret knowledge for $c_i$.

- a **pvR** $(pk \in \mathbb{P}K, c \in \mathcal{C})$, called as a publicly verifiable re-sharing of $c$ from $pk$, is a pair $(c_i', \pi_{pvR} \in \Pi)$ where $\pi_{pvR}$ is proof of plaintext re-sharing of $c$, into $c_i'$, under key $pk$.

Notice that a pvR proves in particular knowledge of a plaintext secret, thus is a fortiiori 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 $A$ 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 $(\text{Enc}_{pk_i}(s_i))_{i \in [n]}$, where the $t$ plaintexts $(s_i)_{i \in \mathbb{I}}$ with coordinates of corrupt parties vary uniformly at random in $\mathbb{F}_p^t$, while the $t+1$ other $(s_i)_{i \in \mathbb{I} \setminus \mathbb{J}}$ are all equal to 0.

**Proposition 9** (IND-CPA of encrypted sharing). For any threshold $1 \leq t$, any PPT machine $A$ has negligible advantage in the following game with an oracle $O$. $A$ gives to $O$: a subset of $t$ indices $\mathbb{I} \subset [n]$, and a list of $t$ public keys $(pk_i)_{i \in \mathbb{I}} \in (\mathbb{P}K \cup \bot)^t$. $O$ generates $(\ldots, pk_i) \leftarrow \text{KeyGen()}$ for $i \in [n] \setminus \mathbb{I}$ which it shows to $A$. $O$ tosses $b \in \{0, 1\}$. Then $A$ is allowed to query $O$ an unlimited number of times as follows. $A$ gives to $O$ any $s \in \mathbb{F}_p$ of its choice, then $O$ with:

- if $b = 1$: encrypted generating $(s_i)_{i \in [n]} := \text{Share}(s)$, returns $(\text{Enc}_{pk_i}(s_i))_{i \in [n]}$; if $b = 0$: $\mathcal{D}_0$ samples $s_i \in \mathbb{F}_p \forall i \in \mathbb{I}$, sets $s_i := 0 \forall i \in [n] \setminus \mathbb{I}$, returns $(\text{Enc}_{pk_i}(s_i))_{i \in [n]}$.

At some point $A$ outputs $b'$ and wins if $b=b'$.

For technical reasons, in the proof of Y-VSS (Section 5.1) games Hyb$\text{\textbf{ref}}[c, i]$ 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 $A$ than in Proposition 9 in the case where $A$ 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 $A$ has negligible advantage in the following game with an oracle $O$. $A$ gives to $O$ a subset of $t$ indices $\mathbb{I} \subset [n]$. $O$ generates $(\ldots, pk_i) \leftarrow \text{KeyGen()}$ for $i \in [n] \setminus \mathbb{I}$ which it shows to $A$. $O$ tosses $b \in \{0, 1\}$. Then $A$ is allowed to query $O$ an unlimited number of times as follows. $A$ queries $O$ with any $s \in \mathbb{F}_p$ of its choice, then $O$ responds as:

- if $b = 1$: enc. sharing + $A$’s shares generating $(s_i)_{i \in [n]} := \text{Share}(s)$, returns $(\text{Enc}_{pk_i}(s_i))_{i \in [n] \setminus \mathbb{I}}$ and $(s_i)_{i \in \mathbb{I}}$.

if $b = 0$: $\mathcal{D}_0$ with $A$’s shares samples $(s_i)_{i \in \mathbb{I}} \in \mathbb{F}_p$, sets $s_i := 0 \forall i \in [n] \setminus \mathbb{I}$, returns both $(\text{Enc}_{pk_i}(s_i))_{i \in [n] \setminus \mathbb{I}}$ and $(s_i)_{i \in \mathbb{I}}$.

At some point $A$ outputs $b'$ and wins if $b=b'$.

**Proof**. We are going to bound the advantage of any adversary $A$, by the maximum advantage of an adversary $A_{PKE}$ against oracle $O_{PKE}$ of the following $t+1$-keys variant indistinguishability game for PKE. The latter is upper-bounded by $n-t$ times the advantage for one-message indistinguishability, see e.g. [17] Thm 5.1). $O_{PKE}$ samples $(n-t)$ PKE public keys $(pk_h)_{h \in [n-t]}$ which it gives to $A_{PKE}$. $O_{PKE}$ secretly tosses a bit $b \in \{0, 1\}$. Upon receiving, from $A_{PKE}$, $(n-t)$ chosen plaintexts $(s_h)_{h \in [n-t]}$, then $O_{PKE}$ returns:

- if $(b = 1)$: correct encryptions, i.e., $(\text{Enc}(pk_h, s_h))_{h \in [n-t]}$;
- if $(b = 0)$: encryptions of 0, i.e., $(\text{Enc}(pk_h, 0))_{h \in [n-t]}$.

The reduction is as follows. $A_{PKE}$ initiates $A$, receives a set of indices $\mathbb{I}$ from $A$. $A_{PKE}$ receives a list of $n-t$ keys $(pk_h)_{h \in [n-t]}$ from $O_{PKE}$. Then it forwards to $A$ the $(n-t)$ keys $(pk_h)_{h \in [n-t]}$ received from $O_{PKE}$, of which it renumbered the indices into $[n] \setminus \mathbb{I}$.

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

- in the case where the ciphertexts \(\{c_h\}_{h \in [n]\setminus I}\) are encryptions of the actual \(n-t\) shares \(\{s_h\}_{h \in [n]\setminus I}\), then \(A\) receives from \(A_{PKE}\) a correctly generated encrypted sharing of \(s\);
- in the case where the ciphertexts \(\{c_h\}_{h \in [n]\setminus I}\) are encryptions of 0, then we have that the \(t\) plaintext shares \((s_i)_{i \in I}\) 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 \(b \in \{0,1\}\), \(A\) is faced with the same distribution as the one generated by the oracle \(\Theta\) of the game of Proposition 10 for the same \(b\) and query \(s\). Thus the distinguishing advantage of \(A_{PKE}\) is the same as the one of \(A\).

\(\blacksquare\)

The next corollary Corollary 11 states that any PPT adversary \(A\), 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 \(D_0\) of Proposition 10. To avoid issues with IND-CCA, we impose in addition that \(A\) sufficiently explains how \(c\) was formed, namely: as one homomorphic linear combination of fresh encryptions, in order to extract from \(A\) 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 \(A\) has negligible advantage in the following game with an oracle \(\Theta\). \(A\) gives to \(\Theta\) a subset of \(t\) indices \(I \subset [n]\). \(\Theta\) generates \((\ldots, pk_i) \leftarrow \text{KeyGen}(\cdot)\) for \(i \in [n] \setminus I\) which it shows to \(A\). It also generates \((sk, pk) \leftarrow \text{KeyGen}(\cdot)\) and gives \(pk\) to \(A\). \(\Theta\) tosses \(b \leftarrow \{0,1\}\). Then \(A\) is allowed to query \(\Theta\) an unlimited number of times as follows. \(A\) queries \(\Theta\) with any \(c\) appended with the following, denoted as “explanations”: \((m_i)_{i \in [t+1]} \in \mathbb{F}_{p}^{t+1}\) and \((\rho_{nc,i})_{i \in [t+1]} \in \mathbb{R}^{t+1}\). \((ld_i)_{i \in [t+1]} \in \mathbb{F}_{p}^{t+1}\), such that \(c = \sum_{i \in [t+1]} (l_i \triangleleft \text{Enc}(m_i, \rho_{nc,i}))\). \(\Theta\) responds as:

\[\text{if } b = 1: \text{enc. re-sharing} + A\text{'s shares} \text{decryption}
\]

\[s := \text{Dec}(sk, c), \text{ generates } (s_i)_{i \in [n]} := \text{Share}(s), \text{ returns } (\text{Enc}_{pk_i}(s_i))_{i \in [n] \setminus I} \text{ and } (s_i)_{i \in I};\]

\[\text{if } b = 0: D_0 \text{ with } A\text{'s samples} \text{ samples } s_i \leftarrow \mathbb{F}_p, \forall i \in I, \text{ sets } s_i := 0 \forall i \in [n] \setminus I, \text{ returns both } (\text{Enc}_{pk_i}(s_i))_{i \in [n] \setminus I} \text{ and } (s_i)_{i \in I}.
\]

At some point \(A\) outputs \(b'\) and wins if \(b = b'\).

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

\(\blacksquare\)

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 \(P_i^{(1)} \in C^{(1)}\) generates \((pk_i^{(1)}, sk_i^{(1)}) \leftarrow \text{KeyGen}(\cdot)\) then registers it to \(\mathcal{F}_{CA}\). Upon receiving sharesig, the dealer \(D:\) retrieves the list of keys \((pk_i^{(1)})_{i \in [n]}\) and generates \(psv \leftarrow \text{pvShare}(s)\) from its secret input \(s\), using the \(n\) encryption keys \((pk_i^{(1)})_{i \in [n]}\). Then, \(D\) broadcasts \(psv\) to \(C^{(1)}\) with its signature on it then shuts-off.

Each party \(P_i^{(1)} \in C^{(1)}\), upon receiving an output of the broadcast of \(D:\) if it is a \(psv \in \mathcal{psv}\), then it sets its local list \(\text{ListOpPps}_i := \{psv\}\). Else, which happens only if the dealer is corrupt, then it sets it as \(\{psv_0\}\) where \(psv_0\) is a fixed predefined \(\mathcal{psv}\) of 0.

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

C.3 Y-VSS-Open. Each \(P_i^{(e_{\alpha})} \in C^{(e_{\alpha})}\), upon receiving the signal “open”, denoting \((pk_i, sk_i)\) its key pair. We consider any arbitrary committee \(C^{(e_{\alpha})}\) which is instructed to open the secret to a designated learner \(L:\) Every party \(P_i \in C^{(e_{\alpha})}\), for every \(\text{pps}_{\alpha} := (\ldots, e_{\alpha}(e_{\alpha})_{i \in [n]}\) in its list \(\text{ListOpPps}_i:\)

- generates a publicly verifiable decryption of \(c_i:\)

\[ (s_i, \pi_{\text{pvs},i}) \text{, to obtain the triple } (c_{\alpha}(c_{\alpha})_{i \in [n]}, s_i, \pi_{\text{pvs},i}) \]

It then sends to \(L\), via \(\mathcal{F}_{SY}\), all such triples at once. Upon receiving \(t+1\) triples of the form \((c_{\alpha}, s_i, \pi_{\text{pvs},i})\) for \(i \in \text{some } (t+1)\)-sized subset \(U\), all with the same \(c_{\alpha}\) \(L\) reconstructs the secret from the Lagrange linear combination of the \((s_i)_{i \in U}\), given by Equation 5, with coefficients \((\lambda_i)_{i \in U}\).

C.4 Handling for unpublished keys. The modification to be done for the general case is: upon receiving refresh sigs (or the sharesig), a party retrieves from the PKI all the public keys available of the entering committee, say \(C'\), makes a list of them, say \(\mathcal{pk}'\), in which it sets to \(\perp\) the ones not retrieved. Encryption with key \(\perp\) is by convention \(\perp\). It appends the hash of \(\mathcal{pk}'\) in all its messages related to \(C'\), and subsequently ignores the incoming messages related to \(C'\) appended with a different hash, i.e., which use a different list of keys than \(\mathcal{pk}'\). We also adapt the proof of correctness, as follows, to handle executions in which there exists some committee \(C'\) for which at most \(t\) published keys are explainable. For any vector of ciphertext shares \(c_{\alpha}\) under such keys, then strictly less than \(t\) coordinates
• a **VPS**\(^{(e)}\), called as a verified proactivized sharing relatively to committee \(C^{(e)}\), is a tuple of the form 
\[ \text{vps}^{(e)} = (c_{i[n]}, qvc^{(e)}), \]
where: 
- \(c_{i[n]} = (c_i)_{i \in [n]} \in \mathbb{G}^n\), and 
- \(qvc^{(e)} = \{\sigma_i\}_{i \in U}\), called as a quorum verification certificate, is a set of \(t + 1\) signatures on \(c_{i[n]}\) issued by some \(t + 1\) subset \(U \subset [n]\) of parties of \(C^{(e)}\).

For brevity we denote \(\text{vps}[i] := c_i\).

• a **ReshareMsg**\(_{(i)}^{(e)}\) for \(i \in [n]\), called as a reshar ing message from \(P_i \in C^{(e)}\), is a triple \((c_i^{(e)}, \text{pvr}_i^{(e)}, \sigma_i^{(e)})\), where:
- \(c_i^{(e)} = (c_i^{(e)})_{i \in [n]} \in \mathbb{G}^n\):
- \(\text{pvr}_i^{(e)} \in \text{pvR}(\text{pk}_i^{(e)}, c_i^{(e)})\), i.e., is a publicly verifiable resharding of \(c_i^{(e)}\) under the key \(\text{pk}_i^{(e)}\) of \(P_i\), encrypted under the keys of \(C^{(e+1)}\);
- and \(\sigma_i^{(e)}\) is a signature of \(P_i^{(e)}\) on \(c_i^{(e)}\).

• a **PPS**\(^{(1)}\), called as a proven proactivized sharing relatively to committee \(C^{(1)}\), is either a pvS signed by the dealer \(D\), denoted as \(\text{pps}^{(1)} := (\sigma_D, \pi_{pvS} D, c_{[n]} \in \mathbb{G}^n)\) or, the public default \(\text{pps}^{(1)}\) of zero: \(\text{pps}_0^{(1)} = (\bot, \bot, (\text{Enc}_{\text{pk}_i(0)})_{i \in [n]})\).

• a **PPS**\(^{(e+1)}\), called as a proven proactivized sharing relatively to committee \(C^{(e+1)}\), is a tuple of the form 
\[ \text{pps}^{(e+1)}((c_{i[n]}^{(e)}, qvc^{(e)}), \{\text{pvr}_i^{(e)}\}_{i \in U}), \]
where:
- \(U \in [n]\) is a \((t+1)\)-sized subset;
- \((c_{i[n]}^{(e)} = (c_{i}^{(e)})_{i \in [n]}, qvc^{(e)})\) is a VPS\(^{(e)}\), [Notice that there is no constraint on the subset of issuers, in \(C^{(e)}\), of the signatures in the qvc\(^{(e)}\). In practice in Y-VSS, honest collectors use those issued by \(U\)];
- \(\text{pvr}_i^{(e)} \in \text{pvR}(\text{pk}_i^{(e)}, c_i^{(e)})\) \(\forall i \in U\), i.e., each \(\text{pvr}_i^{(e)}\) is a publicly verifiable resharding of \(c_i^{(e)}\) under the key \(\text{pk}_i^{(e)}\) of \(P_i\), encrypted under the keys of \(C^{(e+1)}\).

We denote it as 
\[ \text{pps}^{(e+1)} := ((c_{i[n]}^{(e)}, qvc^{(e)}), \{\text{pvr}_i^{(e)}\}_{i \in U} \rightarrow c_{[n]}^{(e+1)}) \]
where, denoting \(\text{pvr}_i^{(e)} = (c_{i[n]}^{(e)}, \pi_{pvR,i}) \forall i\), we have 
\[ c_{i[n]}^{(e+1)} := \bigoplus_{i \in U} (\lambda_i^{U \sqcap c_{i[n]}^{(e)}}) \in \mathbb{G}^n \] (as in Equation 1).

Figure 11: Data Structures for Refresh. The public keys of committee \(C^{(e)}\) are denoted as \((pk_i^{(e)})_{i \in [n]}\), for all \(e \in \mathbb{N}^+\).
D More details on the proof of Y-VSS

In Appendix D.1 we formalize the reduction from the proof of indistinguishability between games $\text{Hyb}^{\text{Refresh}}[e, i]$ and $\text{Hyb}^{\text{Refresh}}[e, i + 1]$, into Corollary 11. Then, in Appendix D.2 we adapt the proof of liveness sketched in Section 13 into a formal proof of liveness of the actual Y-VSS in $2\delta$. In Appendix D.3 we formalize the proof of UC security in the case of a corrupt dealer $D$.

D.1 Reduction of the transition to $\text{Hyb}^{\text{Refresh}}[e, i + 1]$ into IND-CPA of resharing. We consider an adversary $A_{\text{reshare}}$ in the game of Corollary 11. It initiates a concatenation, which we denote as $M$, of all the system up to the $e$-th committee as in the game $\text{Hyb}^{\text{Refresh}}[e, i]$: $E$ and the dummy adversary and all corrupt parties, $D$, the ideal functionalities, and all honest parties in $C^{(1)}, \ldots, C^{(e)}$. Upon receiving keys $(pk_i)_{i \in [n]}$ from $G$, it publishes them on behalf of honest parties in $C^{(e+1)}$ (and, by construction, also publishes the keys of corrupt parties in $C^{(e+1)}$, when they are instructed to by $E$). Consider a point (if any), where the simulated $P^{(e)}_{i+1}$ receives a $\text{PPS}^{(e+1)}_{i}$, out of which it deduces, by Lagrange combination, an encrypted share $c_i$. Then this means that $M$ also provided to $A$ the explanation of $M$. Namely, $A_{\text{reshare}}$ could extract from the $\text{PPS}^{(e+1)}_{i}$ provided by $M$: $t+1$ explainable fresh encryptions, of which $c_i$ is the Lagrange homomorphic linear combination. Then, $A_{\text{reshare}}$ immediately removes $P^{(e)}_{i+1}$ from $M$. Then $A_{\text{reshare}}$ queries $G$ with this material, and is returned a vector of ciphertexts, denoted $c_{i \rightarrow [n]}$, along with the plaintext sub-shares $(s_{i \rightarrow j})_{j \in [e]}$ of $c_{i \rightarrow [n]}$ for the corrupt indices. Then $A_{\text{reshare}}$ re-integrates $P^{(e)}_{i+1}$ into $M$, in which it overwrites its actual re-sharing by $c_{i \rightarrow [n]}$. Then $A_{\text{reshare}}$ finishes to run the execution. Noticeably, thanks to $\text{Hyb}^{\text{Refresh}}[e + 1, n]$, $A_{\text{reshare}}$ does not need to know the secret keys corresponding to those published on behalf of honest parties in $C^{(e+1)}$, to continue the execution. Noticeably, if $e = e_a - 1$, then $A_{\text{reshare}}$ inputs into ShSim the plaintext sub-shares $(s_{i \rightarrow j})_{j \in [e]}$ of $c_{i \rightarrow [n]}$ with indices the corrupt parties in $C^{(e_a)}$. This last precision explains why we specified leakage of the plaintext (sub-)shares in the game of Corollary 12. Then, $A_{\text{reshare}}$ outputs $b = 1$ if $E$ outputs $i$, or outputs $b = 0$ if $E$ outputs $0$. Since the view of $E$ is generated exactly as in $\text{Hyb}^{\text{Refresh}}[e, i]$ if $b = 1$ and as in $\text{Hyb}^{\text{Refresh}}[e, i + 1]$ if $b = 0$, we have that the distinguishing advantage of $A_{\text{reshare}}$ is as least as large as the one of $E$, 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 $C^{(e_a)}$ which receives open sig, have at least one PPS: $\text{pps}$ in common in their lists $\text{ListOfPPS}_i$, when they receive the open sig. Since by [LO], all their messages to the learner $L$ are delivered, $L$ will obtain $t+1$ verifiable plaintext decrypted shares of this same $\text{pps}$. Thus it will be able to output, which concludes liveness. It thus remains to prove the assumption.

Lemma 12. For any $1 \leq e \leq e_0$, consider the local lists $\text{ListOfPPS}_i$ of honest parties in $C^{(e)}$ when they receive the refresh sig (or open sig, for $C^{(e_a)}$), then there is at least one common element $\text{pps} \in \text{PPS}$ 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 $C^{(1)}$ have received the same output from the broadcast of the dealer $D$ before they receive refresh sig or open sig. Either this output is a correct $\text{pvS}$, in which case they all have their local lists $\text{ListOfPPS}_i$ equal to this single $\{\text{ppS}\}$. Or if not the case, they all have their local lists $\text{ListOfPPS}_i$ equal to the same default $\{\text{pps}_0\}$.

Let us assume the statement true until the committee $C$ of some epoch $e < e_0$, and let us prove it for the next committee $C'$. This will imply that statement for all $e$, by induction on $e$. After all parties of $C$ receive refresh sig, if we add a $\delta$ delay, then at this point there is at least one honest collector $K_{i}'$ in $C_{\text{collect}}$ which will receive at $t+1$ refresh messages out of the same common $\text{pps}$ from all $t+1$ honest players. Thus it is able to form a $\text{PPS}$: $\text{pps} = \langle \text{pps}_i \langle \text{pps}_i \rangle, c_{[n]} \rangle$ out of them and multicast it. If we add a $\delta$ delay, by [LR], no honest party in $C'$ has yet received refresh sig or open sig. So at this point, all honest parties in $C'$ are still alive and have received this same $\text{pps}$, so included it its local list $\text{ListOfPPS}_i$.

D.3 Proof of UC security in case of a corrupt dealer $D$. The simulator $S_c$ 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 $L$ is also corrupt, then this behavior perfectly simulates honest participants.

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

---

Section 5.1
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 [27] §5.6, in the context of multiplication of secrets. It surfaced in a written form in [36]. Meanwhile, the vanilla version was rediscovered by [28] in the context of proactivity, then used in [67, 13, 45, 29]. They ([25]) 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 [40, §3.2]. In the context of proactivity, it turns out that the same baseline as above, of encrypted resharing, is also considered in [10, 43]. 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 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 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 3BC. In these works, it is embodied by a public ledger. After 3BC, each party in C’ sets U equal to the t+1 first indices of 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 C’ are returned the same outputs of the n broadcasts, they are guaranteed to see the same set U, and thus to end up with the same c’[n]. Said otherwise, these n terminating broadcasts implement consensus on U.

F More on efficiency and generalizations

F.1 Parallelized implementation of Refresh(C, C’). In Fig. 13 we present the performance of the implementation of a Refresh(C, 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 f(x) = \sum_{i=0}^{n} a_i x^i. The goal is that additions of shares, or 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 [50, IV. A], then a more general one in [50], in the context of threshold RLWE-based schemes. An example of PKE compatible with these operations on the plaintexts is BFV [32, 49]. 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 [59], with any arbitrary PKE.

Over rings of machine integers \Z/2^n\Z. To this end we need Shamir sharing over \Z/2^n\Z, as described in [30, 33]. A very recent example of encryption scheme compatible with linearly homomorphic operations in \Z/2^n\Z is [22].

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, R_{key}, R_{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 Pailler, one can observe that NIZKs of re-sharing are easier done by Pedersen committing to the Pailler 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...
This same observation is made, in GHL [38], for lattice-based schemes. They bring optimized NIZK relations (instantiated with Bulletproofs) for resharing, recently improved in [54].