

Breaking the $t < n/3$ Consensus Bound: Asynchronous Dynamic Proactive Secret Sharing under Honest Majority

Christophe Levrat, Matthieu Rambaud, and Antoine Urban

Télécom Paris, Institut Polytechnique de Paris, France

Version 4 - April 4, 2023¹

Abstract. A proactive secret sharing scheme (PSS), expressed in the dynamic-membership setting, enables a committee of n holders of secret-shares, dubbed as “players” to securely hand-over new shares of the same secret to a new committee. We dub such a sub-protocol as *refresh*. All existing PSS under an honest majority, require the use of a *broadcast* (BC) in each refresh. BC is costly to implement, and its security relies on timing assumptions on the network. So the privacy of the secret and/or its guaranteed delivery, either depend on network assumptions, or, on the reliability of a public ledger. By contrast, PSS over asynchronous channels do not have these constraints. However, all of them (but one, with exponential complexity) use asynchronous verifiable secret sharing (AVSS) and consensus (MVBA and/or ACS), which are impossible under asynchrony beyond $t < n/3$ corruptions *whatever the setup*. We present a PSS, named “asynchronous-proactive secret sharing” (APSS), which is the first PSS under honest majority with guaranteed output delivery in a completely asynchronous network. More generally, APSS allows any flexible threshold $t < n$, such that privacy and correctness are guaranteed up to t corruptions, and liveness as soon as $t+1$ players behave honestly. Correctness can be lifted to any number of corruptions, provided a linearly homomorphic commitment scheme. Moreover, each refresh completes at the record speed of 2δ , where δ is the *actual* message delivery delay. APSS demonstrates that proactive refreshes are possible as long as players of the *first* committee only, have a common view on a set of (publicly committed or encrypted) shares. Despite not providing consensus on a unique set of shares, APSS surprisingly enables “Yoso”-MPC without broadcast, as demonstrated in a follow-up work. In particular, it allows to open the evaluation of any linear map over secrets *non-interactively, without consensus*. This holds in various algebraic structures. APSS can also be directly integrated into the asynchronous Schnorr threshold signing scheme “Roast” (CCS’22). Of independent interest, we:

- provide the first UC formalization (and proof) of proactive AVSS, furthermore for arbitrary thresholds;
- provide additional mechanisms enabling players of a committee to start a Refresh then erase their old shares, synchronously up to δ from each other;
- improve by 50x the verification speed of the NIZKs of encrypted re-sharing of [Cascardo et al, Asiacrypt’22], by using novel optimizations of batch Schnorr proofs of knowledge.

We demonstrate efficiency of APSS with an implementation which uses this optimization as baseline.

Contents

1	Introduction	2
1.1	The so-far curse of broadcast (BC) and synchrony in proactive refresh	3
1.2	Main contributions	5
1.3	APSS: making APSS0 fully asynchronous	8
2	Preliminaries and Model for APSS0	9
2.1	General Notation and Parameters	9
2.2	Polynomials, Shamir secret (re-)sharing, and interpolation	9

⁵**Change log** w.r.t. Version 3 of 2022-10-19: (a) NIZKs of resharings with 50x faster verification than [Cascardo et al, Asiacrypt’22], (b) interactive scheduling mechanism enabling to remove the synchrony assumption at the end of epochs, (c) other complexity improvements.

2.3	Participants: Dealer, Learner and (Dynamic) Committees of players	10
2.4	Adversary \mathcal{A} , Static Corruptions	10
2.5	Specification of the functionality to implement	11
2.6	Asynchronous Message Transmitting, (bare) Multicast, and Actual Network Delay δ	11
2.7	Bulletin board PKI: $\mathcal{F}_{\text{bPKI}}$, and Signatures	11
2.8	Single-Shot Reliable Broadcast $\text{RB}^{\mathcal{D} \rightarrow \mathcal{P}_{[n_1]}^1}$ from \mathcal{D} to $\mathcal{P}_{[n_1]}^1$	12
2.9	NIZKs	12
3	LHE with Limited Evaluations and Bilateral Binding	12
4	Protocol APSS0	14
4.1	Share	14
4.2	Refresh $(\mathcal{P}_{[n]}, \mathcal{P}'_{[n']})$	14
4.3	Open	15
4.4	Latency and Correctness	16
4.5	Communication Complexity	17
5	Proof of Theorem 1	17
5.1	Roadmap	17
5.2	Simulators \mathcal{S} for a corrupt \mathcal{D}	17
5.3	Simulators \mathcal{S} for an honest \mathcal{D}	18
6	APSS = APSS0 + δ -synchronized start & termination	19
6.1	Description of Refreshing squad	20
7	Generalisations and Applications	20
7.1	Enabling to open linear maps non-interactively over several secrets	20
7.2	Integration in existing threshold schemes for signing, VRF, decryption	21
7.3	Resharing based on homomorphic commitments to shares, and relaxing PKE	21
7.4	APSS2: unbounded correctness and no signatures	22
7.5	Initial sharing from DKG	22
7.6	Integration in the asynchronous threshold Schnorr signing “ROAST” (CCS’22)	22
8	Faster NIZKs of Encrypted Resharing, Implementation	22
8.1	Basic proof of correct resharing	22
8.2	Optimization for faster verification	23
A	Complements on the model	28
B	Complements on APSS0: Q & A	32
C	Complements on the proof of APSS0	37
D	Details and proof of Refreshing squad	43
E	Fixing the related asynchronous <i>mobile-PSS</i> model: <i>mobile</i> corruptions, in a <i>static</i> committee	45
F	More on generalizations and applications	48
G	Related specifications of (asynchronous and/or proactive) VSS	51
H	Existing PSS with $t < n/3$ resilience	55
I	Further related works	60

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 up to a threshold number of participants. A mainstream primitive is known as *verifiable secret sharing* (VSS). A dealer \mathcal{D} engages in a protocol, denoted as *Share*, with a committee $\mathcal{P}_{[n]}$ of n machines called players. If it is honest, then all players output in *Share*. There is a parameter t , such that, informally, t is the *corruption threshold for privacy and correctness*, whereas $n - t$ is the *reconstruction threshold*. Very roughly, standalone specifications of a VSS are: [*Correctness, a.k.a. verifiability:*] as soon as one honest player outputs in *Share*, there exists a well defined value, s , which we say is *committed*. It is furthermore equal to the secret input of \mathcal{D} if it is honest. Consider a number of secrets for which *Share* completed. Subsequently, for any (possibly external) entity \mathcal{L} , dubbed as *learner*, there is

a protocol between any $\mathcal{P}_{[n]}$ and \mathcal{L} , denoted as **Open**. It is parametrized by a public linear combination A taking the secrets as inputs. It guaranteed that the only value that \mathcal{L} can output, is the evaluation of A on the committed secrets. *[Secrecy:]* if \mathcal{D} is honest, then \mathcal{A} cannot distinguish between two chosen secret inputs of \mathcal{D} . *[Liveness (Completeness)]* if **Share** terminated *and* if there exists at least $t+1$ players which follow the protocol, notwithstanding they could be passively corrupt [CDN15, §3], then \mathcal{L} outputs in **Open**. In flagship use-cases, **Open** is upgraded to reveal only the evaluation of a function on s (or on several secrets), typically for the purpose of threshold randomness generation [GJM+21], decryption [KJY+20] or signature [CGG+20]. Although this work is in the UC model, for completeness we detail in Appendix G the standalone specification of AVSS achieved, and compare it with the many existing ones. For many use cases recalled in Sections 7.1 and 7.2 and appendix F.2.2, \mathcal{D} can be emulated by a *distributed key generation* (DKG) algorithm. In this work we consider a *dynamic set of participants* [DJ97; SLL10; BGG+20; MZW+19; GKM+22; Gro21; GHK+21; GHL22; VAFB22; CDGK22; YXXM23; HZC+22; HKMR22]. In this model there is a counter, known as as *epoch* $e = 1, 2, \dots$ which very broadly models time. There is one separate set of players per epoch: $\mathcal{P}_{[n_e]}^e = (\mathcal{P}_1^e, \dots, \mathcal{P}_{n_e}^e)$, known as a *committee*. Starting from a VSS to $\mathcal{P}_{[n_1]}^1$, a (*dynamic*) *proactive secret sharing scheme* (PSS) provides a protocol, denoted as **Refresh**, between any two consecutive committees $\mathcal{P}_{[n_e]}^e$ and $\mathcal{P}_{[n_{e+1}]}^{e+1}$, dubbed old and new. They are parametrized by thresholds t_e and t_{e+1} . **Refresh** enables to extend the properties of the initial VSS, throughout the lifetime of the system, despite changes of participants and corruptions. Informally, the standalone specifications are:

[Secrecy] still holds after an arbitrary number of Refreshes up to some e_A , if at most t_e players in every $\mathcal{P}_{[n_e]}^e$, $e \leq e_A$, are corrupt by an adversary \mathcal{A} ;

[Liveness] if $t_e + 1$ players in every $\mathcal{P}_{[n_e]}^e$ up to some $\mathcal{P}_{[n_{e_o}]}^{e_o}$ are passively corrupt or honest, then all Refreshes up to $\mathcal{P}_{[n_{e_o}]}^{e_o}$ complete and $\mathcal{P}_{[n_{e_o}]}^{e_o}$ are able to **Open** to \mathcal{L} ;

[Correctness] after an arbitrary number of Refreshes up to some e_A , if at most t_e players in every $\mathcal{P}_{[n_e]}^e$, $e \leq e_A$ are corrupt by an adversary \mathcal{A} , then the only value which can be delivered to \mathcal{L} is the s which was committed in the initial **Share** to $\mathcal{P}_{[n_1]}^1$. //“robustness” sometimes refers to the combination of correctness and liveness.

As discussed at the end of Section 1.3 and in Appendix E, our results a fortiori apply to the model of *mobile-PSS*, with a static committee and a mobile adversary [HJKY95; CKLS02; GDK22].

1.1 The so-far curse of broadcast (BC) and synchrony in proactive refresh

BC requires an idealized synchronous network. We consider existing PSS *tolerating an honest majority*, i.e., which allow to set the thresholds to at least $t_e = \lfloor n_e/2 \rfloor - 1$, i.e., which can tolerate at least $f_e < n_e/2$ malicious corruptions per committee. All them so far [HJKY95; BGG+20; GKM+22; Gro21; GHK+21; HKMR22] rely on a primitive, known as (*Byzantine*) *broadcast* [FLL21, Definition 1]. We dub it as BC. It involves a *sender* S and a set of receivers \mathcal{R} . It requires that: (*Termination*) all receivers eventually output //even if S keeps silent; (*Consistency*): the same value; (*Validity*): which is furthermore equal to the input of S if it is honest. It is trivial that BC cannot be implemented without a *synchronous* network, whatever the setup. A *synchronous* network is the idealized model that there exists a public parameter Δ such that, in the whole execution, all messages sent are delivered within Δ .

Loss of security if synchrony fails at any point in time. In an implementation of a BC, if one message arrives after Δ , then the synchrony assumption fails so BC may lose consistency. In all aforementioned PSS under honest majority, if consistency of the BC is violated, then players output inconsistent new shares, which results in a *complete loss of the shared secret*. Worse, in some PSS, *the secret is leaked* if one single message arrives after Δ . This concerns those based on the accusation-response mechanism of Pedersen’s VSS: [MZW+19; GKM+22; HKMR22]. This mechanism is a convenient way to ensure that every player received a correct private message, without paying the price of public-key encryption appended with NIZK proofs of correctness of the plaintext. In [GKM+22, p8], when the expected private message (and/or the expected BC) from some honest \mathcal{P}_i is not received within Δ by an honest \mathcal{P}_j , then \mathcal{P}_j publicly *accuses* \mathcal{P}_i , which must then publicly expose the content of this message, which provides substantial information to the adversary on the stored secret. In Exp-CHURP-A [MZW+19, p. C.1.3], when the BC from an honest \mathcal{P}_i is not received

within a fixed delay Δ_{BC} , then honest players are forced to publicly expose what they sent to \mathcal{P}_i , which provides enough information to the adversary to reconstruct the stored secret.

Cost and security issues with practical implementations of BC. To minimize such scenarios, practical implementations set a delay between each interaction, which we denote also Δ . It is set to a very high conservative estimate of the worst-case message delivery delay (including the gaps between the local clocks of players). In Table 1 we give a first sight of the substantial latencies and costs of implementations of BC, complemented in Appendix I.4. The largest number n of players which we know to be used in practice is $n = 15$, in [JS20]. Nevertheless, our work is also addressing larger values of n . All academic works on PSS under honest majority targeting large values of n , such as [GKM+22; MZW+19], [Gro21] (Dfinity’s) or [BGG+20; HKMR22] (Algorand’s), suggest to emulate BC by publishing on a *public ledger*. Instantiating BC with Ethereum or Bitcoin brings about extra security, performance and cost issues, as illustrated in (3) of Table 1, then in Appendix I.4.3.

When It’s All Just Too Much: can BC be downgraded in existing PSS ? The main use-case of MPC in practice is the proactive refresh of shared keys and their use for threshold signing. The company “Fireblocks” (of market capitalization 8Bn\$) reports that Refreshes are performed every “minutes” [Fir22]. Hence, their main payload seems *not* to be the initial generation or sharing of keys, *nor* the punctual cost of using them for threshold-signing, but instead *the life-time long continuous Refresh of keys*. **This work addresses the latter**. A much used protocol for key-refresh (and threshold signature) is the one of [CGG+20, Figure 6] (long version of their merged CCS’20 paper). It is known under the name “CMP”. It is the one used by Fireblocks. It is also implemented by [Ami22] with a funding of Coinbase [Coi22] (of market capitalization 13Bn\$), and also by Taurus [Gro22]. At a high level, the Refresh of CMP proceeds as in the classical PSS of [HJKY95]. Namely, each players shares 0, then players adds these shares to their old share, to obtain their new share. Further rounds of accusation enable players to accuse the dealers which did not send consistent shares to them. The specification of CMP is lighter than [HJKY95], since they guarantee an output only if all players behave honestly. Thanks to this relaxed specification, they removed all the BC from their Refresh, making it much lighter. They observe [CGG+20, §1.2.8] that their signing algorithm can be generalized to lower thresholds $t < n - 1$, simply by switching to (n, t) Shamir sharing. However, we observe that this change alone would not lift the restriction of their current key-refresh, which is that one single deviating player can make the Refresh non-unanimously abort //simply by sending a rogue accusation in the 1. of Output. in [CGG+20, Figure 6]. The accusation message is denoted as $\langle \text{DecError} \rangle$. A rogue accusation can simply consist in claiming not to have received anything from the sender \mathcal{P}_j , or, received a wrong ciphertext from a colluding sender. As currently specified, it seems even feasible to send the rogue accusation to one isolated honest player \mathcal{P}_i , just before all other honest players believed that the Refresh went well and erase their old share. As a result, \mathcal{P}_i keeps its old share and does not store its new share. In such a scenario, honest players end up with both *less old and less new shares, than the threshold number needed* for reconstruction of the secret. Such a scenario was evidenced by the audit of Kudelski of the threshold wallet of ING [AS20, §3]. Their attack is called “Forget-and-Forgive”. We estimate that upgrading the key refresh of CMP to guarantee an output, despite a non-zero number t of misbehaving players, would require a total of *three* BC. The details of the estimation are given below. They illustrate why BC is too costly for real life industrial implementations, so that they remove it when possible. They also illustrate why removing BC, results in general in non-unanimous aborts.

- In Round 2, BC everything which is sent via P2P messages //otherwise, players could not reach a consistent view on a subset of senders which behaved well, and of which the sharings of 0 should be taken into account. In particular, this now makes Round 1 useless. In particular, one cannot use anymore their suggestion to downgrade the BC into “echo”, also known as *consistent broadcast* ([Rei94], [CKPS01, p. 3.3]). Despite the terminology “consistent”, echo allows that some players may output \perp , while others output an actual value. This suggestion is inherited from [GL02], and made its way into the aforementioned implementation of CMP [Ami22]. [One can further notice that this downgrade from BC to echo is also suggested in [DPSZ12, p4&25], while the unanimous abort in their functionality Fig. 15 holds only for an actual BC.]
- In (former) Round 3, BC the Paillier ciphertexts of shares of 0 //so that players can verify subsequent accusations not to have received a correct ciphertext from a sender, and unanimously decide if it should be discarded.
- In Output 1. BC the accusations $\langle \text{DecError} \rangle$ //otherwise players may end up with an inconsistent view on the players from which the sharings of 0 should be discarded.

1.2 Main contributions

Our first contribution is the specification of an ideal functionality for (asynchronous proactive) VSS, denoted as $\mathcal{F}_{\text{P-AVSS}}$. We outline it in Section 2.5, the formalism being in Fig. 5. It is parametrized by any thresholds $t_e < n_e$ per committee. It captures guarantees on privacy, correctness and liveness, depending on any actual number $f_e \leq n_e$ of corruptions. Let us just mention very roughly that it guarantees *privacy* and *correctness* up to $f_e \leq t_e$ corruptions per committee; and liveness if at least $t_e + 1$ parties per committee are passively corrupt or honest, and if *one* collector per $\mathcal{K}_{[\kappa_e]}^e$ is passively corrupt or honest. We refer to Appendix G.3 for a survey on existing functionalities for (A)VSS. Besides those which do not guarantee output delivery [CMP20], or, which assume that dummy players make synchronously their requests [HKMR22; YXXM23], there is the one of [AAPP22] which is not enough to enable guaranteed output delivery under honest majority, since it outputs raw Shamir shares.

Our main contribution is a proactive secret sharing protocol, called APSS0, which is the first tolerating both honest majority and asynchrony. More precisely, we consider a fully asynchronous communication network, with a bare bulletin board of public keys. Also, since AVSS is impossible under asynchrony beyond $t < n/3$, we make the extra assumption of a reliable broadcast (RB), denoted as $\text{RB}^{\mathcal{D} \rightarrow \mathcal{P}_{[n+1]}^1}$, *between the dealer \mathcal{D} and the first committee only* and **used only once**. Then:

Theorem 1. *Under the previous model, formalized in Section 2, there is a protocol, called APSS0 and described in Section 4, which UC emulates $\mathcal{F}_{\text{P-AVSS}}$ in the sense of universal composability (UC, [Can01]). Any Refresh started synchronously is expedited in 2δ , where δ is the actual message delivery delay.*

In particular, setting all thresholds to $t_e = \lceil n_e/2 \rceil - 1$, then APSS0 *has guaranteed output delivery (GOD) under honest majority*. In Section 7.4 we lift correctness to any number of corruptions, provided any commitment scheme supporting unlimited homomorphic additions.

Unprecedented security and speed. Since APSS0 does not rely on BC nor synchrony assumptions, it removes the aforementioned security and cost issues, which so far impacted all existing PSS under honest majority. Comparing the performances of APSS0 with existing PSS, which all have a weaker security due to these issues, is not apples-to-apples. Nevertheless in Table 1 we compare to previous PSS *tolerating an honest majority*. Other PSS tolerating a lower number $t < n/3$ of malicious corruptions are discussed at the end of Section 1.3.

Improved complexity APSS0 operates with intermediary committees for each epoch, called *collectors* and denoted as $\mathcal{K}_{[\kappa_e]}^e$. Liveness is guaranteed as soon as *one* collector per $\mathcal{K}_{[\kappa_e]}^e$ is passively corrupt or honest, the other guarantees brought by $\mathcal{F}_{\text{P-AVSS}}$ *still hold if all collectors are fully corrupt*. In the simplified $\mathcal{F}_{\text{P-AVSS}}$ above we considered that collectors $\mathcal{K}_{[\kappa_e]}^e$ consist of a $\kappa_e = (t_e + 1)$ -subset of $\mathcal{P}_{[n_e]}^e$, in addition to their role. Collectors can also be dynamically and non-interactively sampled at random among committees using a threshold coin. For instance, as shown in Section 4.3, the ones of [CKS05; GJM+21] can be used and refreshed inside APSS0. For instance the probability of failing to sample at least *one* collector honest among $\kappa = 12$ sampled, out of a committee of $n = 121$ players of which $t = 40$ are corrupt, is 0.00004%. //By comparison, in [GDK22, §8.2] (for $t < n/3$) they require that at least *half* of their collectors are honest. Considering the same numerical example, the probability of *failing* to match their requirement of 7 honest out-of 12 collectors sampled, is 16%. In Appendix B.1 we describe a division by $O(n)$ of the communication for n secrets in parallel, but with more latency.

Modularity & application to threshold signing and decryption APSS0 operates on any secret space supporting Shamir sharing. It enables the threshold opening of any linear map evaluated over shared secrets, without consensus on a set of vectors of shares. In Section 7.2 we exemplify how APSS can be used **without interaction** to generate a threshold BLS signature, Elgamal decryption, randomness generation and threshold (R)LWE decryption; and in Section 7.6: integration *without additional interaction* in the state of the art asynchronous interactive threshold Schnorr [RRJ+22].

Composability with any scheduling mechanism, YOSO. In APSS0, a player initiates a Refresh, or an Open, upon receiving input (refresh-sig), or (open-sig). In APSS0, a player **shuts-off**, i.e., erases its memory and quits the protocol, upon receiving the input (shutoff-sig). Since APSS0 is proven UC secure, it can be

Scheme	Network	Latency ⁽³⁾	Communication ⁽¹⁾	Setup
[HJKY95; HKMR22]	Synch	$\delta_{\text{PKI}} + 3\Delta_{\text{BC}}^n$	$n \text{BC}(n\gamma) $	PKI & URS ⁽⁶⁾
[BGG+20],[Gro21],[GHL22]	Synch	$\delta_{\text{PKI}} + \Delta_{\text{BC}}^n$	$n \text{BC}(n\gamma) $	PKI & NIZKs ⁽⁵⁾
[MZW+19, §C.1]	Synch	$\delta_{\text{PKI}} + 5\Delta_{\text{BC}}^n$	$n \text{BC}(n\gamma) $	PKI & NIZKs
[GKM+22]	Synch	$5\Delta_{\text{BC}}^n$	$ \text{BC}(n\gamma) $ ⁽⁴⁾	SRS ⁽⁶⁾
APSS	Asynch	$\delta_{\text{PKI}} + 2\delta$ ⁽²⁾	$\kappa \text{MC}((n\gamma)^2) $	PKI & NIZKs ⁽⁵⁾

- (1) The *communication complexity* of a Refresh in APSS0 is the total number of bits sent by the n honest players in the exiting committee $\mathcal{P}_{[n]}$ and by the κ collectors. We consider here $n'=n$ players of a new committee $\mathcal{P}'_{[n']}$, they do not speak. γ denotes the security parameter. For a given BC algorithm, $|\text{BC}(B)|$ denotes the total number of bits sent by honest players in an execution with input length $\Omega(B)$ and n receivers. Assuming honest majority $t < n/2$ and assuming a trusted setup and a synchronous network and a static adversary, then the BC of [ADD+19] has communication complexity $\text{BC}(B) = \Omega(B^2)$. Under dishonest majority, $t < n$, which is common for threshold wallets, and assuming a trusted VRF setup and a static adversary, the best-known constant-round BC [WXSD20] has $\text{BC}(\mathbf{1}) = \tilde{O}(n^4)$. The BC of [DS83] has $\text{BC}(\mathbf{1}) = O(\gamma n^2 + n^3)$. $|\text{MC}(B)|$ denotes the communication complexity of a *multicast* of $\Omega(B)$ bits to n receivers. A survey is given in Appendix I.4. A multicast is the authenticated sending of the same message to n players. The straightforward implementation, of sending it to each of the n players, costs $|\text{MC}(B)| = nB$. It can also be implemented by gossiping the signed message [CKMR22].
- (2) δ is the actual largest message delay in an execution. A typical intercontinental delay is $\delta = 100\text{ms}$, see Appendix I.6. Δ is a parameter which is set high enough so that no message ever should take more than Δ to be delivered. In the implementation reported in [AMN+20], they set $\Delta = 50\text{ms}$, which is equal to 50 times their measured actual message delay $\delta = 1\text{ms}$. If we apply the same $50\times$ conservative overhead to 100ms , we obtain $\Delta = 5\text{s}$. This is the order of magnitude of the $\Delta = 12\text{s}$ assumed by Ethereum.
- (3) Δ_{BC}^n denotes the expected latency of n BC in parallel and δ_{PKI} is the publication delay on the PKI. The BC of [ADD+19] has latency $\Delta_{\text{BC}} = 10\Delta$. Hence, n instances in parallel terminate after an expected $\Delta_{\text{BC}}^n = O(10\Delta \log(n))$, see Appendix I.7. Likewise, for the one of [WXSD20], we deduce an expected $\Delta_{\text{BC}}^n = O((\frac{n}{n-t})^2 \log(n))$. The BC of [DS83] has a fixed latency: $\Delta_{\text{BC}} = \Delta_{\text{BC}}^n = t+1$. A survey is given in Appendix I.4. When n is too large, then, e.g., Ethereum can be used as a BC, under the assumption that $2/3$ of stakeholders are honest, with $\Delta_{\text{BC}} = 7$ minutes. This costs transaction fees for n times n -sized broadcasts, of **15\$** each. Details are given in Appendix I.4.3.
- (4) The $|\text{BC}(n)|$ complexity displayed for [GKM+22] holds if amortized over $\geq n$ secrets.
- (5) In our implementation, as well as in [CMP20; BGG+20; Gro21; GHL22], NIZK-AoKs are instantiated from Fiat-Shamir transforms of special sound public coin HVZK-AoKs. So they require a priori a programmable random oracle (RO). URS stands for a public uniform random string, possibly known long before players publish their keys. Uniformity allows generation by nothing-up-my-sleeve sampling or distributed beacons [CD20]. It is needed for the setup of the Pedersen commitment scheme in [HJKY95, footnote 2][HKMR22]. A URS is also required in [Gro21; GHL22] as a *common parameter for the public keys* (the scheme being CCA in the former), enabling to amortize the sizes of ciphertexts. Since the last two assume the RO, they can simply set $\text{URS} := \text{RO}(1)$.
- (6) SRS designates a structured random string, needed for the polynomial commitment KZG, it is also needed in [MZW+19, §C.1]. SRS could be downgraded into URS in both [GKM+22; MZW+19], provided a polynomial commitment under URS.

Table 1: Comparison of Refreshes of PSS tolerating an *honest majority*, for committees of size n .

composed with *any* external scheduling mechanism sending these inputs. Some are surveyed in Appendix H.4. A global clock is considered in [CKLS02, p18], which, phrased in our terminology, sends to all (**refresh-sig**), then *just after*, (**shutoff-sig**). Liveness in [CKLS02, p18] is guaranteed under the synchronous assumption that this clock waits for all messages of a **Refresh** to be delivered, before sending its next tick to all. //No PSS can guarantee liveness with a clock which would urge players to speak-now-then-shut-off, before they received enough messages. Our $\mathcal{F}_{\text{P-AVSS}}$ coarsely captures this (see the very last comment in Fig. 5. *Instantiating APSS0* with this model of synchronous clock, which waits for the messages to the next committee being delivered before ticking, then *participants speak only once*. In that sense, this instantiation matches an even stronger requirement than YOSO [GHK+21], called *layered MPC* in [DKI+23].

We also provide contributions of possible independent interest:

A new computation model. A **Refresh** in APSS0 proceeds as follows. It uses as baseline the well-known method of *resharing* of Shamir shares. Consider an old and new committee: $\mathcal{P}_{[n]}$ and $\mathcal{P}'_{[n']}$, with thresholds t and t' . Each player \mathcal{P}_i of $\mathcal{P}_{[n]}$ starts not with one, but a *list* of n -sized vectors of ciphertexts. Each vector consists in n Shamir shares of s with threshold t , encrypted under the public keys of $\mathcal{P}_{[n]}$. For each vector $c_{[n]}$ in its list, \mathcal{P}_i multicasts to the collectors $\mathcal{K}'_{[k]}$ a re-sharing of its share, encrypted under the public keys of $\mathcal{P}'_{[n']}$. So this comes as an n' -sized vector of ciphertext shares: $c_{i \rightarrow [n]}$. It appends it with a NIZK argument of knowledge (AoK) proving the correct decryption-then-resharing-then-reencryption. Each collector, upon receiving a batch of $t+1$ consistent encrypted resharings, i.e., out of the *same* vector $c_{[n]}$, computes homomorphically their Lagrange linear combination, to obtain one vector of ciphertext new shares: $c'_{[n']}$. Then it multicasts $c'_{[n]}$ to $\mathcal{P}'_{[n']}$, appended with the received batch of resharings, and disappears. Each player of $\mathcal{P}'_{[n']}$ accepts at most one vector of encrypted new shares from each collector \mathcal{K}'_k . It includes it in its list only if the NIZKs appended prove that it is made of correct resharings and that the Lagrange linear combination is correct. *Liveness* is guaranteed by the fact that there exists at least one honest collector which sends to $\mathcal{P}'_{[n']}$ the *same* vector of ciphertext new shares.

Now, if **Refreshes** went on like this, since players do not have a public ledger, they would have to piggy-back to their re-resharings the entire history of previous encrypted resharings and of the NIZKs vouching for their correctness. To avoid this, we introduce a chain of correctness mechanism. Collectors collect signatures attesting the validity of vector of ciphertext shares. Once a vector of ciphertext shares has reached a quorum of $t+1$ signatures, it needs not anymore be appended with NIZKs. A novelty is that we “pipeline” this mechanism within each resharing, and thus do not degrade the latency.

50x speedup over [CDGK22, Asiacrypt’22]. In Section 8 we report on our implementation of NIZK-AoKs of encrypted resharing, instantiated with Elgamal encryption. We improve the ones of [CDGK22], which are also for Elgamal. We achieve a 10x speedup in verifying ($n = 1001$)-sized resharings, issued by a quorum of $t+1 = 501$ resharers-out-of- n , and a 50x speedup for $W = 1000$ secrets in parallel. A possibly new trick is the batch verification of proofs of resharing. We furthermore enable the verifier to precompute offline the scalar coefficients involved in the batch verification, thereby removing $2n \cdot W \cdot (t+1)$ online multiplications in \mathbb{F}_p , leaving only one online multi-scalar-multiplication of size $t+1$.

Simulatability of PVSS, without straight-line extraction. A by-product of our UC proof, is that it positively answers the question raised by Shrestha-Bhat-Kate-Nayak in [SBKN21, p5] (the v1), whether PVSS would be simulatable. Moreover, under honest majority we achieve it from any NIZK AoKs with simulation-soundness *after possibly rewinding*, i.e., *weak* simulation extractability [FKMV12], i.e., *not necessarily in straight-line*. This includes Bulletproofs and Spartan [BBB+18; Set20], as very recently shown under the DLOG by [DG23]. We were later informed of other works doing simulation proofs for PVSS [CD20; GHK+21; Div22], but all of them require NIZK-AoKs with *straight-line, a.k.a. online, extractability*. So this ruled-out Bulletproofs and Spartan.

A practical and rigorous formalization of resharing-friendly PKE. Recall that in APSS0, players homomorphically compactify a batch of $t+1$ resharings, into one vector of ciphertext new shares. We make the simple but possibly new observation, that, since players subsequently decrypt-then-reshare their new share, the public key encryption scheme (PKE) needs only supporting a *limited* number of linearly homomorphic operations mod p . //This somehow relates to the interactive bootstrapping of threshold ciphertexts [CLO+13,

Choudhury et al, Asiacrypt’13]. This brings about a new specification of public key linearly homomorphic encryption (LHE) scheme PKE, modulo a fixed prime p , made in Section 3. After completion of this work, some other works considered resharing with a LHE supporting an *unlimited number of LH operations*: [Div22; CDGK22]. This is only matched by the LHE described in [CCL+20, §3.2], and by Elgamal for plaintexts in DDH-hard groups. By contrast, our specification allows the use of much more existing LHE’s, including the two considered in [ISO19, ISO/IEC 18033 Part 6], i.e., Paillier and “Elgamal in-the-exponent” [CCN21]. It also specifies an encryptor-decryptor binding property, which is necessary for the robustness of any encrypted resharing scheme without extra commitments to subshares. It was apparently forgotten in existing such schemes [BGG+20; GHK+21] //and notified to them.

1.3 APSS: making APSS0 fully asynchronous

We now go beyond the composability of APSS0 with any scheduling mechanism, and introduce one, in Section 6, which is tailored to APSS0. It is a sub-protocol, called the *Refreshing squad* //in honor of [CDDS85], which delivers to players the inputs (*refresh-sig*) and (*shutoff-sig*). The former triggers a player \mathcal{P}_i^e to wait a delay Δ_{wait}^e , which is a tunable parameter depending on e , then to initiate a Refresh. Optionally, we enable Refreshing squad to also deliver a new kind of output, called (*keys-sig*). When this option is activated, players in the new committee $\mathcal{P}_{[n_{e+1}]}^{e+1}$ do not generate and publish their encryption key, until they have received (*keys-sig*) and waited the further tunable delay Δ_{wait}^e . We call APSS the whole protocol obtained. Refreshing squad is designed so as to shrink to a minimum the critical timeframe during which an old committee is still online, i.e., not shut-off, while players have generated secret material related to the next epoch, i.e.: their keys for new players, and their resharnings for old players. Late generation of this secret material is guaranteed by the two properties called *last minute* in Theorem 2. Minimizing this critical timeframe is useful when APSS is compiled into the related *mobile-PSS* setting (see below and in Appendix E). Indeed, corruptions during this timeframe count in both budgets of the old and the new epoch. Another practical advantage is that, since future committees need only be created when they need to generate their keys, our *last-minute key generation* is useful for security and cost reasons. This is discussed in [GHK+21] (“Future Horizon”).

As stated in Theorem 2 this mechanism guarantees liveness, and openability of the secret during a *window of opening*, of which the duration depends on the parameters Δ_{wait}^e . In conclusion, APSS is a fully asynchronous PSS under honest majority, **of which the liveness does not depend on any timing assumption** //in particular in Theorem 2, if we set all Δ_{wait}^e ’s finite until some $\Delta_{\text{wait}}^e = \infty$, then $\mathcal{P}_{[n_{e_0}]}^{e_0}$ is guaranteed to Open the secret to \mathcal{L} . As detailed in Appendix H.4, all existing termination mechanism, for PSS under asynchrony, relied on $t < n/3$ malicious corruptions.

Theorem 2 (APSS). *Protocol APSS implements functionality $\mathcal{F}_{\text{P-AVSS}}$, except that liveness is further conditioned to all committees of collectors $\mathcal{K}_{[r_e]}^e$ having an honest majority. Furthermore, consider an execution in which both committees and collectors have an honest majority for all $e \geq 1$, and in which the RB ($\text{RB}^{\mathcal{D} \rightarrow \mathcal{P}_{[n_1]}^1}$) to $\mathcal{P}_{[n_1]}^1$ terminates, e.g., if \mathcal{D} honest. For each e , consider the time T^e at which the first player of $\mathcal{P}_{[n_e]}^e$ receives (*refresh-sig*). Then the T^e ’s are all finite, monotonically increasing, and such that:*

Last-minute key generation $\forall e$: no player of $\mathcal{P}_{[n_{e+1}]}^{e+1}$ generates its key pair before $T^e - \delta + \Delta_{\text{wait}}^e$;

Fast shutoff all players of $\mathcal{P}_{[n_e]}^e$ have shut-off before $T^e + \Delta_{\text{wait}}^e + 4\delta$.

Window of opening for any e such that: $\Delta_{\text{wait}}^e > \delta$ and all honest players in $\mathcal{P}_{[n_e]}^e$ have received (*open-sig*) before $T^e + \Delta_{\text{wait}}^e$, then \mathcal{L} outputs before $T^e + \Delta_{\text{wait}}^e + \delta$ //(open-sig) formalizes any external instruction to Open;

The most general (and correct) “mobile” corruption model in a static committee. In Appendix E we describe a generic compilation from any PSS with dynamic committees, into a PSS for a *fixed* committee, $\mathcal{P}_{[n]}$, in which a *mobile* adversary \mathcal{A} changes its corruptions over time. In the tradition of [OY91; HJKY95], we call this model *mobile-PSS*. However, we found no model of mobile-PSS under asynchrony which would capture all existing protocols. The one of [CKLS02] assumes a global clock which ticks epochs, (*refresh-sig*)

and (shutoff-sig). Their liveness is conditioned to ticks happening after all messages of the previous Refresh were delivered. The one of [ABKL22] was proven by themselves to be unimplementable. The one of [SLL10] forgets to count in the corruption budget of *both* adjacent epochs, a player which would be corrupt during a Refresh//this does not affect the correctness of their PSS, which is instead in the *dynamic* setting. The problem in the model of [SLL10] impacted recent mobile-PSS which borrowed it: [GDK22], and the mobile-PSS model considered in [YXXM23, §A]. //We notified to them a passive corruption attack on their mobile-PSS, which they quickly acknowledged to have already identified, and fixed the model in their new version. In Appendix E we go beyond this observation, and introduce a corruption model which captures all existing PSS in the static setting, *including* [CKLS02]. It furthermore captures differences in the security levels of existing PSS schemes, depending on fine-grained delays in their Refreshes.

Faster than all existing asynchronous PSS. For completeness we survey, in Appendix H, further PSS which do not guarantee simultaneously liveness, correctness and privacy beyond $t < n/3$ malicious corruptions [ZSV05; CKLS02; SLL10; DM15; YXD22; VAFB22; YXXM23; HZC+22; GDK22]. APSS guarantees all three up to $t < n/2$. As APSS, most of them tolerate asynchrony. Nearly all of them use a primitive known as *consensus*, in order to agree on a set of new shares. An exception is [ZSV05], but which has exponential complexity. But it is trivial ([DLS88, Thm 4.4]) that consensus is not implementable above $t < n/3$ corruptions, even under partial synchrony and *whatever the setup*. Due to their use of consensus, the Refresh of these PSS take at least 16δ (and $O((t+1)\Delta)$ in the worst-case for [SLL10; VAFB22]), vs 2δ for APSS.

2 Preliminaries and Model for APSS0

In Sections 2.3 and 2.4 we define the participants, adversary and corruptions. In Section 2.5 we define the ideal functionality which APSS0 aims at implementing. Then in Sections 2.6 to 2.9 we specify the resources at hand: asynchronous communication channels, bulletin board of keys etc. In Appendix A we give the full details of the formalization in the UC framework of [Can01]. In Appendices E and G we compare with other related models and impossibilities.

2.1 General Notation and Parameters

General (arbitrary) parameters Let p be any prime number, denote as $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$ the finite field of order p . Let $(\mathbb{S}, 0, +)$ be any \mathbb{F}_p -vector space, denoted as the *space of secrets and shares*. For instance, the reader may consider $\mathbb{S} = \mathbb{F}_p$. Another useful example is $\mathbb{S} = \mathbb{F}_p[Y]/Q$ a finite polynomial ring. In our ElGamal-based implementation, we will set \mathbb{S} an abelian group of order p , in additive notation, in which DDH is hard. The security parameter is denoted as γ .

General notation For F a finite set, we denote $|F|$ its cardinality, and $f \stackrel{\mathbb{S}}{\leftarrow} F$ the sampling of an element in F uniformly at random. The empty string is denoted as \perp . For m an integer, we denote $[m] := \{1, \dots, m\}$. Vectors with coordinates indexed by some set are denoted with this set as subscript, e.g., $c_{i \rightarrow [n']} := (c_{i \rightarrow j})_{j \in [n']}$. Their coordinates are mostly denoted as subscript, and also sometimes in brackets, e.g., $\mathbb{L}_i[k]$ for the k -th entry of \mathbb{L}_i . Unless specified otherwise, $\|\cdot\|$ denotes the sup norm $\|\cdot\|_\infty$.

Eventual Delivery When we say that an output v of an ideal functionality, typically a message sent over an asynchronous channel, is *eventually-delivered* to P , we informally mean the following. The functionality informs the adversary that an output is ready for P , and possibly leaks v , if it is public. The adversary can delay the delivery of v up to a *finite polynomial delay*, which it *adaptively adjusts*, after which v is delivered to P . We further formalize this in the UC framework in Appendix A.1.1, following existing works.

2.2 Polynomials, Shamir secret (re-)sharing, and interpolation We refer to [CDN15, §3.2]. Let $t < n$ be any integer such that $n < p$. $\mathbb{S}[X]_{\leq t}$ denotes the $(t+1)$ -vector space of polynomials of degree at most t with coefficients in \mathbb{S} . We abuse notation and denote them as *degree- t polynomials*. We denote as $\mathbb{F}_p[\mathbf{X}]_t^{(0)} \subset \mathbb{F}_p[X]_{\leq t}$ the t -vector subspace of degree t polynomials evaluating to 0 at 0, i.e., of the form $\sum_{\ell=1}^t a_\ell X^\ell$. To generate a Shamir secret sharing of a secret $S \in \mathbb{S}$: sample a random degree- t polynomial $h \in \mathbb{S}[X]_{\leq t}$ such that $h(0) = S$, then output the *shares*: $S_i \leftarrow h(i), \forall i \in [n]$.

- Let us concretely describe the case where \mathbb{S} is abelian of order p ([ACR21, §6]). Consider G any generator of \mathbb{S} , e.g., $G = 1$ if $\mathbb{S} = \mathbb{F}_p$. Sample $\sum_{\ell=1}^t a_\ell X^\ell \leftarrow_{\mathbb{S}} \mathbb{F}_p[X]_t^{(0)}$, set $h \leftarrow S + (\sum_{\ell=1}^t a_\ell X^\ell) \cdot G$, then output $S_i = h(i) = S + (\sum_{\ell=1}^t i^\ell \cdot a_\ell) G$, $\forall i \in [n]$. The same goes coordinate-per-coordinate for any \mathbb{S} .

Conversely, we denote as a *vector of (n, t) -shares of S* any vector of the form $(h(i))_{i \in [n]}$, for any degree- t polynomial h s.t. $h(0) = S$. Any degree- t polynomial h is *uniquely* determined by its evaluations at any distinct $t+1$ points, by a mere *linear combination*. Consider any $(t+1)$ -sized subset $\mathcal{U} \subset \mathbb{F}_p$. Define the *Lagrange polynomials* as $\lambda_i^{\mathcal{U}}(\mathbf{X}) \leftarrow \prod_{j \in \mathcal{U} \setminus \{i\}} \frac{\mathbf{X} - j}{i - j} \in \mathbb{F}_p[\mathbf{X}]_{\leq t}$, $\forall i \in \mathcal{U}$. Then, for any $h \in \mathbb{S}[X]_{\leq t}$ we have the *Lagrange interpolation formula*: $h = \sum_{i \in \mathcal{U}} \lambda_i^{\mathcal{U}}(X) \cdot S_i$. Hence, h is referred to as *the sharing polynomial* of the $(S_i)_{i \in [n]}$. In particular, denoting as $\lambda_i^{\mathcal{U}} \leftarrow \lambda_i^{\mathcal{U}}(\mathbf{0})$ the “Lagrange coefficients”, we have *linear reconstruction* of the secret from any $t+1$ shares: $S = h(0) = \sum_{i \in \mathcal{U}} \lambda_i^{\mathcal{U}} \cdot h(i)$.

Finally, we have the *t -privacy* property that, for a fixed s , any t -sized subset $\mathcal{I} \subset [n]$ of coordinates, then when h varies uniformly s.t. $h(0) = s$, we have that $(S_i = h(i))_{i \in \mathcal{I}}$ vary uniformly independently in \mathbb{S}^t . //This follows from the invertibility of any $t+1$ -sized minor of the $(t+1) \times n$ Vandermonde matrix.

We now recall the method of re-sharing of any vector of (old) shares $(S_i)_{i \in [n]}$ of some S , into a new (n', t') -vector of shares, for any $t' < n'$. Consider any $t+1$ -sized subset $\mathcal{U} \subset [n]$ and, for each $i \in \mathcal{U}$, any vector of shares $S_{i \rightarrow [n']} = (S_{i \rightarrow j})_{j \in [n']}$ of S_i , denoted as *sub-shares*. We dub the latter as a *re-sharing of S_i* , or also as *sub-shares of S_i* . Define the *new shares* $S'_{[n']} = (S'_j)_{j \in [n']}$ as:

$$(1) \quad S'_j \leftarrow \sum_{i \in \mathcal{U}} \lambda_i^{\mathcal{U}} \cdot S_{i \rightarrow j}, \quad \forall j \in [n'] . \textit{Fact: they form a vector of shares of } S.$$

Let us recall the proof of the fact. For each vector of sub-shares $S_{i \rightarrow [n']}$, denote as H_i the sharing polynomial, i.e., $S_{i \rightarrow j} = H_i(j) \quad \forall j \in [n']$. Define the *new polynomial* as $H' \leftarrow \sum_{i \in \mathcal{U}} \lambda_i^{\mathcal{U}} \cdot H_i$. Then, by construction, we have that $S'_j = H'(j) \quad \forall j \in [n']$. On the other hand, also by construction, we have that $H'(0) = \sum_{i \in \mathcal{U}} \lambda_i^{\mathcal{U}} S_i = s$, which concludes the proof. An illustration is provided in Fig. 15, and a historical account in Appendix I.1.

2.3 Participants: Dealer, Learner and (Dynamic) Committees of players We consider one PPT machine called *dealer* and denoted as \mathcal{D} and one called *learner* and denoted as \mathcal{L} . \mathcal{D} initially starts with an input $s \in \mathbb{S}$, denoted as its *secret* //we will revert to the upper-case notation S in our Elgamal-based implementation. \mathcal{L} may output some value in \mathbb{S} at some point. We then show in Section 4 how to open linear combinations of several secrets from several \mathcal{D} 's. We consider an arbitrarily long sequence of integers $e = 1, 2, \dots$, referred to as “epoch numbers”. For each $e \in \mathbb{N}^*$, we consider a set of $n_e < p$ distinct probabilistic polynomial time (PPT) machines $\mathcal{P}_{[n_e]}^e = (P_1^e, \dots, P_{n_e}^e)$, called *committee of share-holders*, or simply *committee*. Their members are denoted as *players*. When a player receives the input (shutoff-sig), then it *shuts-off*, i.e., erases all its memory and quits the protocol. We say that a player is *online* if it is not *shut-off*. For each $e \in \{2, 3, \dots\}$, we also consider a set of κ_e distinct PPT machines $\mathcal{K}_{[\kappa_e]}^e$, denoted as a *committee of collectors*. Their sizes $n_e < p$, κ_e are tunable parameters. For ease of notation we set $\mathcal{K}_{[0]}^1 := \emptyset$ //notice that, if we had made the alternative convention: $\mathcal{K}_{[1]}^1 := \{\mathcal{D}\}$, then this would have shortened our Proposition 5 and Section 6. We did not, for sake of clarity. All players $\{\mathcal{P}_i^e\}_{i,e}$ and collectors are disjoint threads. In the sampling method suggested in the introduction, we provided probabilities under the specific model where a collector thread running on the same computer as a corrupt player thread, would automatically be corrupt. Our results hold in general, without imposing such correlations between corruptions. At any point, any player \mathcal{P}_i^e , $\forall e, i$, may receive two inputs: (refresh-sig) and (open-sig). Anticipating on APSS, and roughly speaking: the former instructs \mathcal{P}_i^e to start resharing its secret shares to $\mathcal{P}_{[n_{e+1}]}^{e+1}$, and the latter to start sending its secret shares to \mathcal{L} .

2.4 Adversary \mathcal{A} , Static Corruptions We consider a PPT machine called the *adversary* and denoted \mathcal{A} . It can (*maliciously*) *corrupt* any participant, i.e., \mathcal{D} , \mathcal{L} and any member of committees $\mathcal{P}_{[n_e]}^e$, $\mathcal{K}_{[\kappa_e]}^e$, $\forall e$, before they start the protocol. This means the following. A corrupt entity reveals all its internal state to \mathcal{A} [which boils down to, for \mathcal{D} : it input; for the others: nothing], then becomes forever a proxy of \mathcal{A} . The corrupt players in each committee $\mathcal{P}_{[n_e]}^e$ are indexed as $\mathcal{I}^e \subset [n_e]$. We denote their number as $f_e \leftarrow |\mathcal{I}^e|$. The

non-corrupt participants as denoted as *honest*, and indexed as $\mathcal{H}^e = [n_e] \setminus \mathcal{I}^e$. In Appendix B.2 we discuss how early adaptive corruptions are supported with less than $\binom{n}{t}$ loss.

2.5 Specification of the functionality to implement What we are aiming at, is a protocol, APSS0, which *UC-emulates* the following dummy protocol. The meaning of *to UC emulate* is reminded in Section 5.1, with further helpful diagrams in Figs. 10 and 11. The non-trivial actions of the dummy protocol are performed by an ideal functionality which we introduce. We call it the functionality of *proactive asynchronous verifiable secret sharing*, and denote it as $\mathcal{F}_{\text{P-AVSS}}$. Its full specification is given in Fig. 5, we now convey the main ideas.

The dummy \mathcal{D} gives its input s to $\mathcal{F}_{\text{P-AVSS}}$, which stores it. From this point, \mathcal{D} is *committed* to s . This means that, unless a number of corruptions above t_e in some committee, $\mathcal{F}_{\text{P-AVSS}}$ cannot deliver to \mathcal{L} another value than s . $\mathcal{F}_{\text{P-AVSS}}$ eventually notifies (*committed*) to all dummy players of the first committee $\mathcal{P}_{[n_1]}^1$. When all honest dummy players of $\mathcal{P}_{[n_1]}^1$ have received *committed* // [which will correspond, in APSS0, to: terminated the initial AVSS], then $\mathcal{F}_{\text{P-AVSS}}$ memorizes that $\mathcal{P}_{[n_1]}^1$ can realize an opening. To this end, it initiates the counter $e_o \leftarrow 1$. $\mathcal{P}_{[n_{e_o}]}^{e_o}$ is the highest committee allowed to make an opening of the secret. Upon receiving input (*refresh-sig*) or (*open-sig*), an honest dummy player forwards it to $\mathcal{F}_{\text{P-AVSS}}$ in the form of a request: (*refresh-req*) or opening: (*open-req*). The only not totally dummy rule, is that dummy players of $\mathcal{P}_{[n_1]}^1$ must *wait* to have been notified (*committed*), before they request (*refresh-req*) or (*open-req*). Upon receiving (*shutoff-sig*), a dummy player notifies it to $\mathcal{F}_{\text{P-AVSS}}$ then *shuts-off*. Upon receiving (*refresh-req*) from $t_{e_o} + 1$ dummy players of committee $\mathcal{P}_{[n_{e_o}]}^{e_o}$ and if at least *one* collector in $\mathcal{K}_{[\kappa_{e_o+1}]}^{e_o+1}$ is honest, then $\mathcal{F}_{\text{P-AVSS}}$ eventually updates $e_o \leftarrow e_o + 1$ // unless too much shut-offs happened in-between. Upon receiving (*open-req*) from $t_e + 1$ dummy players of some committee $\mathcal{P}_{[n_e]}^e$, $e \leq e_o$, $\mathcal{F}_{\text{P-AVSS}}$ eventually delivers [unless too much shut-offs happened in-between] the stored value to \mathcal{L} . We denote this event as a *collective opening*. If more than $t_e + 1$ players in some committee are corrupt, then $\mathcal{F}_{\text{P-AVSS}}$ leaks the stored value, after letting \mathcal{A} change its value. This latter power will be thwarted in APSS2. We now stress some subtleties. Consider *any* protocol implementing $\mathcal{F}_{\text{P-AVSS}}$ as described above. Consider the scenario where *one isolated* honest player would start to reveal its shares to \mathcal{L} , i.e., starts the *Open* protocol. Since it is hard to prevent t_e corrupt players from also sending compatible consistent shares to \mathcal{L} , this results in the secret being opened to \mathcal{L} . We qualify such event as an *early opening*. // We did not find formalized elsewhere this unavoidable power, e.g., not in the related F_{COM} of [CDN15, p. 105]. We formalize this adversarial power by having $\mathcal{F}_{\text{P-AVSS}}$ maintain another counter, denoted as $e_A \geq e_o$. It updates it to $e_A + 1$ as soon as $t_{e_A} + 1 - f_{e_A}$ honest dummy players of $\mathcal{P}_{[n]}^{e_A}$ request (*refresh-req*) // roughly this translates in the protocol by: send their subshares to $\mathcal{K}_{[\kappa_{e_A+1}]}^{e_A+1}$. We denote such event as an *early refreshing*. Then, we give to \mathcal{A} the power to send an (*open-order*) to $\mathcal{F}_{\text{P-AVSS}}$, which triggers an immediate delivery of the secret to \mathcal{L} , as soon as $t_e + 1 - f_e$ honest dummy players of some $\mathcal{P}_{[n]}^{e \leq e_A}$ request (*open-req*).

2.6 Asynchronous Message Transmitting, (bare) Multicast, and Actual Network Delay δ All players in some committee $\mathcal{P}_{[n_e]}^e$ are connected with all collectors in $\mathcal{K}_{[\kappa_{e+1}]}^{e+1}$, themselves connected with the next committee $\mathcal{P}_{[n_{e+1}]}^{e+1}$, by *public authenticated message transmitting with eventual delivery*. It is captured by the following functionality \mathcal{F}_{AT} , which is further formalized in Section 2.6. It is parametrized by a sender S and a receiver R . On input (*input, ssid, v*) from S , then $\mathcal{F}_{\text{AT}}^{S,R}$ leaks v to \mathcal{A} and eventually delivers (*ssid, v*) to R . To *multicast* a message to a committee barely means to send it over \mathcal{F}_{AT} to all its members. All players are furthermore connected to \mathcal{L} by *secure message transmitting*. It is the same as \mathcal{F}_{AT} , excepted that only the bitlength $|v|$ is leaked. For a given execution, we define as δ the time taken by the longest message delivery. // δ is measured *a posteriori*. \mathcal{A} needs not commit on an upper bound on δ . In Appendix H.1 we discuss weaker models which assume so.

2.7 Bulletin board PKI: $\mathcal{F}_{\text{bPKI}}$, and Signatures We consider the ideal functionality of a bulletin board of public keys, denoted as $\mathcal{F}_{\text{bPKI}}$ and presented in Appendix A.1.4. Upon receiving a key ek_i^e from any

player $\mathcal{P}_i^e \in \mathcal{P}_{[n_e]}^e$, it stores $(\mathcal{P}_i^e, \text{ek}_i)$ and leaks this information to \mathcal{A} . For each e it: -waits until it received a public key from every honest player $\mathcal{P}_i^e \in \mathcal{P}_{[n_e]}^e$ -sets a timeout -then after it elapsed, sets to \perp the keys of the (necessarily corrupt) players of $\mathcal{P}_{[n_e]}^e$ which did not give a key. Then it sets as $\text{ek}_{[n_e]} \leftarrow (\text{ek}_i)_{i \in [n_e]}$ the vector of all keys, [where those from some *corrupt* players may be \perp], and eventually delivers it to all the system. We also require any black box standard digital signature scheme satisfying unforgeability in the sense of EUF-CMA.

2.8 Single-Shot Reliable Broadcast $\text{RB}^{\mathcal{D} \rightarrow \mathcal{P}_{[n_1]}^1}$ from \mathcal{D} to $\mathcal{P}_{[n_1]}^1$. To share its input to $\mathcal{P}_{[n_1]}^1$, \mathcal{D} will perform *only once* a *reliable broadcast* (RB). RB is a weakening of BC, in which receivers $\mathcal{R} = \mathcal{P}_{[n_1]}^1$ *do not need to output* if the sender $S = \mathcal{D}$ is corrupt. We abstract it out as the following functionality denoted as $\text{RB}^{\mathcal{D} \rightarrow \mathcal{P}_{[n_1]}^1}$, and formalized in Appendix A.1.7. *Upon receiving* a value, say s , for the first time from \mathcal{D} , then: it leaks s to \mathcal{A} , and eventually-delivers s to each receiver $\mathcal{P}_i^1 \in \mathcal{P}_{[n_1]}^1$. $\text{RB}^{\mathcal{D} \rightarrow \mathcal{P}_{[n_1]}^1}$ can be implemented assuming *any* of the following assumptions. We refer to Appendix I.5 for a detailed survey.

- \mathcal{D} is always honest. //It can be implemented by a mere insecure multicast.
- $t_1 < n_1/3$. //in 3δ and $O(Bn^2)$ bits, or 4δ and $O(Bn + \gamma n^2)$ bits
- $t_1 < n_1/2$ AND the network provides *two initial rounds of synchrony then becomes asynchronous*: [GPS19, §5], see also Appendix I.2.
- $t_1 < n_1$ AND the network provides $t+1$ synchronous rounds.

2.9 NIZKs We will use *non-interactive zero knowledge arguments of knowledge*, which we dub *NIZK AoKs*, or simply *NIZKs*. We capture them by the ideal functionality $\mathcal{F}_{\text{NIZK}}$, recalled in Appendix A.1.5. Moreover, under honest majority, i.e., if $t_e < n_e/2 \forall e$, we relax $\mathcal{F}_{\text{NIZK}}$ into any NIZK AoK with *weak simulation extractability* [FKMV12], i.e., which *possibly requires rewinding*.

3 LHE with Limited Evaluations and Bilateral Binding

We introduce a new specification of linearly homomorphic public key encryption (LHE). It comes as the set of algorithms $\text{PKE} = (\text{EKGen}, \text{Enc}, \text{Dec}, \boxplus, \boxdot)$ described below. //Roughly speaking, we require that a ciphertext c “correctly decrypts mod p ”, even if it was produced by colluding encryptor(s) and decrypter, even as the result of homomorphic operations, as long as the “sizes of the plaintext and noise of c ” are below some fixed bounds M and R . Aiming at using PKE in APSS0, we set the (front-end) plaintext space equal to our \mathbb{F}_p -vector-space of secrets: \mathbb{S} . Now, for the (back-end) purpose of analyzing the properties of PKE, it is convenient to consider a larger *extended plaintext space*. We consider $(\mathbb{M}, 0, +)$ a *fixed* abelian group containing \mathbb{S} as a subset, endowed with a norm $\|\cdot\|$ and a linear map denoted mod p , such that $\mathbb{S} \hookrightarrow \mathbb{M} \xrightarrow{\text{mod } p} \mathbb{S} = \text{id}_{\mathbb{S}}$. //Informally, “Linear” means that $\{+ \text{ then mod } p\} = \{\text{mod } p \text{ then } +\}$. For instance, although Paillier has *one distinct plaintext space* $\mathbb{Z}/N_i\mathbb{Z}$ for each public key ek_i , we set the *extended plaintext space* as $\mathbb{M} := \mathbb{Z}$ with $\|\cdot\|$ the absolute value. Then, the common (front-end) plaintext space in APSS0 is $\mathbb{S} := \mathbb{F}_p \subset \mathbb{Z}$. Anticipating, the correctness bounds below impose $(t+1)p^2 < (N_i - 1)/2 \forall i$.

- To generate a key pair: sample a secret $\text{dk} \in d\mathcal{K}$, output $\text{ek} := \text{EKGen}(\text{dk}) \in e\mathcal{K}$, where EKGen is a *public deterministic* function. //Determinism does not restrict generality, up to incorporating in dk the randomness necessary to generate (dk, ek) ([HPW15, §2.1]).
- There is a (back-end) *deterministic function* which takes any $m \in \mathbb{M}$, public key $\text{ek} \in e\mathcal{K}$, randomness $\rho \in \mathbb{Z}^d$, where d is a parameter, then returns $\text{Enc}_{\text{ek}}(m; \rho)$ in the *ciphertext space* \mathcal{C} .
- There is a (front-end) *deterministic decryption function* which takes any ciphertext $c \in \mathcal{C}$ and key dk , and returns $\text{Dec}(\text{dk}, c) \in \mathbb{S} \sqcup \perp$.
- There are deterministic (front-end) functions: the *homomorphic addition* \boxplus and the *scalar multiplication* \boxdot , such that $\forall (\text{ek}, m, m', \rho, \rho')$, we have $\text{Enc}_{\text{ek}}(m; \rho) \boxplus_{\text{ek}} \text{Enc}_{\text{ek}}(m'; \rho') = \text{Enc}_{\text{ek}}(m + m'; \rho + \rho')$, and, $\forall \lambda \in \mathbb{Z}$, we have $\lambda \boxdot_{\text{ek}} \text{Enc}_{\text{ek}}(m; \rho) = \text{Enc}_{\text{ek}}(\lambda m; \lambda \rho)$. //In particular, $(-1) \boxdot_{\text{ek}}$ cannot just be built from \boxplus .

//By convention, any $\text{ek} \notin \mathcal{eK}$ is denoted as \perp . By convention, $\text{Enc}_\perp(m; *) := m$. For this reason, we consider that \mathcal{C} contains a copy of \mathbb{S} , on which \boxplus and $\lambda\boxtimes$ simply operate as $+$ and $\lambda\times$.

We require IND-CPA over $\mathbb{S} \subset \mathbb{M}$, as follows. Define the (front-end) *randomized encryption* algorithm under key ek as: on input $s \in \mathbb{S}$, sample $\rho \xleftarrow{\$} [-R_{\text{enc}}, R_{\text{enc}}]^d$, where R_{enc} is a public bound, then return $\text{Enc}_{\text{ek}}(s; \rho)$ //imposing the sampling to be uniform in such a subset of \mathbb{Z}^d does not restrict generality. Then any PPT machine \mathcal{A} has negligible advantage in the following game: $\{b \leftarrow \mathcal{A}(c) : \text{ek} = \text{EKGen}(\text{dk} \xleftarrow{\$} \mathcal{dK})\}$; $\mathbb{S}^2 \ni (s_0, s_1) \leftarrow \mathcal{A}(\text{ek})$; $b \xleftarrow{\$} \{0, 1\}$; $c \leftarrow \text{Enc}_{\text{ek}}(s_b)$.

Definition 3 (LHE). We say that PKE *supports linearly homomorphic operations mod p up to (M, R) , with bilateral binding*, if any PPT machine \mathcal{A} has negligible probability to produce a $(\text{dk} \in \mathcal{dK}, m \in \mathbb{M}, \rho)$ such that $\|m\| \leq M$, $\|\rho\| \leq R$ and $\text{Dec}(\text{dk}, \text{Enc}_{\text{ek}}(m, \rho)) \neq m \pmod p$, where $\text{ek} := \text{EKGen}(\text{dk})$.

For any $0 \leq t < n < p$, and keys $(\text{ek}_i)_{i \in [n]}$ of which at least $t+1$ are correctly generated, we say that a n -uple of ciphertexts: $(\text{Enc}_{\text{ek}_i}(s_i; \rho_i))_{i \in [n]}$ is a *vector of ciphertext shares* if $\|s_i\| \leq M$ and $\|\rho_i\| \leq R$, $\forall i \in [n]$ and the $(s_i \bmod p)_{i \in [n]}$ form a vector of (n, t) -Shamir shares. //Definition 3 guarantees that they do decrypt to $(s_i \bmod p)_{i \in [n]}$. We specify any (M, R) s.t.: $M \geq p^2(t+1)$ and $R \geq p(t+1)R_{\text{enc}}$. //Anticipating on APSS0 (Algorithm 2) the consequence is as follows. Consider any $t+1$ n' -sized vectors of *fresh* encryptions of Shamir (sub)shares, indexed by some set \mathcal{U} of size $t+1$: $\{(\text{Enc}_{\text{ek}_j}(s_{i \rightarrow j}; \rho_j))_{j \in [n']}$: $i \in \mathcal{U}\}$. Each of their n' coordinates is encrypted under the public key ek'_j of some *decrypter* \mathcal{P}'_j . We can even assume that each vector was generated by a malicious encrypter \mathcal{P}_i . The bottomline is that each $(s_{i \rightarrow j})_{j \in [n']}$ forms a vector of share and that the encryptions are *fresh*, i.e., $\|\rho_j\| \leq R_{\text{enc}}$, $\forall j \in [n']$, $\forall i \in \mathcal{U}$. Then consider the homomorphic evaluation of the Lagrange combination (Eq.1) applied coordinate-by-coordinate to these $t+1$ vectors. Denote it as $c'_{[n']}$. This is described in Equation (2) and illustrated in Figs. 15 & 12. Denote as $(s'_j)_{j \in [n']}$ the linear combination of the vectors of plaintext (sub)shares. Then, the specifications $M \geq p^2(t+1)$ and $R \geq p^2(t+1)R_{\text{enc}}$ imply that $c'_{[n]}$ is a vector of ciphertext shares of $(s'_j)_{j \in [n']}$. In particular any decrypter \mathcal{P}'_j , even if colluding with the $t+1$ encrypters, *cannot possibly* exhibit a secret key dk'_j explaining ek'_j , under which the decryption of its new share would be different from s'_j .

//The case where both $M = R_{\text{enc}} = \infty$ and the probability is zero, is known as *LHE with perfect correctness* mod p . There exists *only two such schemes*. The one of [CCL+20, §3.2] is used in the PVSS of [Div22; KMM+23]. The one of Elgamal [$\mathbb{M} := \mathbb{S}$ abelian of order p , $\|\cdot\|$ nil and mod $p = \text{id}_{\mathbb{S}}$] is used in the PVSS of [CDGK22]. Definition 3 is also matched by Paillier, with (front end) plaintexts in $\mathbb{S} := \mathbb{F}_p$. The parameters are $M \leq (N-1)/2$, for N a lower bound on the $\text{ek}_i = N_i$, and $R = \infty$: see [BDOZ11, §2.1]. We make the observation that it is also matched by Elgamal with plaintext in-the-exponent, up to M a size for which solving DLP is efficient. It is also matched by lattice-based schemes, e.g. Regev mod p ([BDOZ11, §2.1]), provided a cap on the noise in encryption (R_{enc}) and key generation (\mathcal{dK}). Further details and efficiency discussions are in Appendix F.1.

//A major difference compared to the specification of “SHE” in [BDOZ11, §2] is that their probability is “taken over the random choices” used to *honestly* sample dk (and, less clearly, ρ). So this allows the probability to be non-negligible when the decrypter *chooses* its secret key, and, a possibly colluding encryptor chooses the encryption noise. Another minor difference is that they specify \mathbb{Z} as the fixed plaintext space. Our greater generality enables to capture schemes such as [BFV/BGV21], for which $\lambda\boxtimes$ multiplies the plaintext by a *polynomial* λ . A last addition is our introduction of a (front end) plaintext space, for easier usability in protocols. In Appendix I.3 we compare to the related notions of committing/robust/undeniable encryption. We finally explain why some existing specifications of PKE’s used for encrypted resharing, actually do not prevent *undetected loss of the secret*. The PKE in [GHK+21] is specified so that EKGen is injective ⁶. This does not prevent corrupt encryptor(s) \mathcal{P}_i , possibly colluding with decrypter \mathcal{P}'_j , to create a decryption different from the plaintext. The same attack is not either prevented by the specification of the PKE in [BGG+20]. In fact, even if they suggest that their ciphertexts of sub-shares play the role of “commitments” to subshares, this actually seems incompatible with their PKE being deniable. In more detail they specify the use of the RIND-SO scheme of [HPW15], as discussed in Appendix B.2, which has some form of deniability.

⁶p30: “so, given a ciphertext, an adversary should not be able to come up with an alternative decryption key which can decrypt this ciphertext to an incorrect message.”.

4 Protocol APSS0

For simplicity, the following description of APSS0, takes as public parameters: a list of public encryption keys $\text{ek}_{[n_e]}^e$ for every committee $\mathcal{P}_{[n_e]}^e$, as well as, implicitly, a list of public signature verification keys for every committee $\mathcal{P}_{[n_e]}^e$ and for \mathcal{D} . It is straightforward to compile the description, into the actual APSS0 in the $\mathcal{F}_{\text{bPKI}}$ model: see Appendix B.2. Our model of Section 2 assumes that all committees are initialized since the beginning of the execution. This limitation is straightforward to lift, see Appendix B.4. The description is for one dealer \mathcal{D} and one learner \mathcal{L} , then in Section 7.1 we explain how to open linear combinations of secrets from several dealers. APSS0 consists of the following subprotocols:

- Share** (Section 4.1) Is between \mathcal{D} and the first committee $\mathcal{P}_{[n_1]}^1$. They start it as soon as they are initialized.
- Refresh**($\mathcal{P}_{[n_e]}^e, \mathcal{P}_{[n_{e+1}]}^{e+1}$) $\forall e$: (Section 4.2) takes place between committees $\mathcal{P}_{[n_e]}^e$ and $\mathcal{P}_{[n_{e+1}]}^{e+1}$, denoted as *old* and *new*, it also involves the committee $\mathcal{K}_{[\kappa_{e+1}]}^{e+1}$ of collectors. Any player $\mathcal{P}_i^e \in \mathcal{P}_{[n_e]}^e$ starts **Refresh**($\mathcal{P}_{[n_e]}^e, \mathcal{P}_{[n_{e+1}]}^{e+1}$) upon receiving the input (**refresh-sig**). On the other hand, $\mathcal{P}_{[n_{e+1}]}^{e+1}$ start **Refresh**($\mathcal{P}_{[n_e]}^e, \mathcal{P}_{[n_{e+1}]}^{e+1}$) as soon as they are initialized.
- Open** (Section 4.3) is between all committees and \mathcal{L} . \mathcal{L} starts **Open** since it is initialized. Any player $\mathcal{P}_i^e, \forall i, e$, starts **Open** upon receiving the input (**open-sig**). //Anticipating, players of $\mathcal{P}_{[n_1]}^1$ do not take any action in **Open** before they output (**committed**), since they did not receive their share.

4.1 Share The dealer \mathcal{D} , on input a secret s , chooses a random polynomial $h(\cdot) \in \mathbb{S}[X]_{\leq t_1}$ of degree at most t_1 , such that $h(\cdot) = s$, and samples encryption randomnesses $(\rho_j)_{j \in [n_1]}$ with norms $\leq R_{\text{enc}}$. It then generates the n_1 -sized vector of ciphertext shares $c_{[n_1]} := (\text{Enc}_{\text{ek}_i^1}(h(i); \rho_i))_{i \in [n_1]}$ under the encryption keys $\mathcal{P}_{[n_1]}^1$, as well as a NIZK AoK, π_{sh} , of: s , a degree- t_1 polynomial h , and encryption randomnesses $(\rho_i)_{i \in [n_1]}$ s.t. $c_{[n_1]}[i] = \text{Enc}_{\text{ek}_i^1}(h(i); \rho_i)$ and $\|\rho_i\| \leq R_{\text{enc}} \quad \forall i \in [n_1]$. //Such $(c_{[n_1]}, \pi_{\text{sh}})$ is widely known as a *publicly verifiable secret sharing* (PVSS) [Sta96]. It Reliably-broadcasts ($\text{RB}^{\mathcal{D} \rightarrow \mathcal{P}_{[n_1]}^1}$) $(c_{[n_1]}, \pi_{\text{sh}})$ to $\mathcal{P}_{[n_1]}^1$ //this is the *only* $\text{RB}^{\mathcal{D} \rightarrow \mathcal{P}_{[n_1]}^1}$ in the whole protocol, then **shuts-off** itself. Upon receiving an output from $\text{RB}^{\mathcal{D} \rightarrow \mathcal{P}_{[n_1]}^1}$, a player $\mathcal{P}_i^1 \in \mathcal{P}_{[n_1]}^1$:

- checks if it is of the form $(c_{[n_1]}, \pi_{\text{sh}})$ with π_{sh} a NIZK AoK as above;
- if the check fails //which can happen if \mathcal{D} is corrupt, then sets $c_{[n_1]}$ equal to a pre-defined default vector of ciphertext shares of 0, e.g., $c_{[n_1]} \leftarrow (\text{Enc}_{\text{ek}_i^1}(0; 0))_{i \in [n_1]}$;
- wraps $c_{[n_1]}$ it in a list of size one $\mathbb{L}_i^1 \leftarrow \{c_{[n_1]}\}$ and outputs (**committed**). //Let s be the secret of which the plaintexts of $c_{[n_1]}$ are the shares, we say that “*the secret s was committed by \mathcal{D}* ”. In particular, s must be the input of \mathcal{D} if it is honest.

In what follows, we dub the unique common $c_{[n_1]}$ as “*the vector of ciphertext shares received from \mathcal{D}* ”. //This terminology is a slight abuse, in the case above where $c_{[n_1]}$ is set to the default vector.

4.2 Refresh ($\mathcal{P}_{[n]}, \mathcal{P}'_{[n']}$) In Algorithm 2 we describe a **Refresh** between an old committee $\mathcal{P}_{[n]} := \mathcal{P}_{[n_e]}^e$ and a new committee $\mathcal{P}'_{[n]} := \mathcal{P}_{[n_{e+1}]}^{e+1}$. Each player in $\mathcal{P}_{[n]}$ has a *list of inputs* $\mathbb{L}_i \in \{\mathcal{C}^n, \perp\}^\kappa$ of fixed size κ . *Some entries may be initially \perp , they may later receive an element of \mathcal{C}^n , during the Refresh*. The reason is that the list \mathbb{L}_i keeps receiving values from the ongoing previous **Refresh**($\mathcal{P}_{[n_{e-1}]}^{e-1}, \mathcal{P}_{[n_e]}^e$). //This is a common point with PSS/MPC based on “agreement on a common subset”, see Appendix H.3.3. There, the input of each player is a n -sized binary list, of which it switches the i -th entry to 1 upon terminating the AVSS from the i -th dealer. The first committee have their lists of size 1, which they fill with the vector of ciphertext shares received from \mathcal{D} . To be included in the list of an honest player, any vector $c_{[n_e]} \in \mathcal{C}^n$ must furthermore come appended with enough data proving that it has been correctly formed, as we are going to detail. In particular, the vector of ciphertext shares $c_{[n_{e-1}]}$ from which it was computed, must be endorsed by $t_{e-1} + 1$ signatures issued by players in $\mathcal{P}_{[n_{e-1}]}^{e-1}$ (or received from \mathcal{D} , for $\mathcal{P}_{[n_1]}^1$). We call these signatures as a *quorum verification certificate*, denoted as **qvc**. Existence of a **qvc** guarantees that at least one honest signer, in $\mathcal{P}_{[n_{e-1}]}^{e-1}$, validated

that $c_{[n_{e-1}]}$ had been correctly formed. The goal of a Refresh is to achieve that, for each of the following predicates, if it holds at some point for $\mathcal{P}_{[n]}$, then it holds for $\mathcal{P}'_{[n']}$ //Lemma 6 shows the first, provided $\leq t$ corruptions in $\mathcal{P}_{[n]}$; and Lemma 4 shows the second to hold eventually, if at least $t+1$ players of $\mathcal{P}_{[n]}$ are honest and one collector in $\mathcal{K}'_{[\kappa']}$ is honest.

- We say that the *correctness invariant* holds for $\mathcal{P}_{[n]}$ if, if there exists a vector of ciphertext shares $c_{[n]}$ included in the list of an honest player of $\mathcal{P}_{[n]}$, then [A] there exists a secret, s , which was shared by \mathcal{D} , and [B] $c_{[n]}$ is a vector of ciphertext shares of s .
- We say that the *common set of shares invariant* holds for $\mathcal{P}_{[n]}$, if there is an index $k \in [\kappa]$ and a $c_{[n]} \in \mathcal{C}^n$, such that all the k -th entries of the lists \mathbb{L}_i of honest players \mathcal{P}_i of $\mathcal{P}_{[n]}$, are all equal to $c_{[n]}$. //players do not reach consensus on k , this is how we break the $t < n/3$ bound.

Refresh($\mathcal{P}_{[n]}$, $\mathcal{P}'_{[n']}$)

Participants: $\mathcal{P}_{[n]}$, $\mathcal{P}'_{[n']}$ and collectors $\mathcal{K}'_{[\kappa']}$.

Public parameters: any \mathbb{S} and LHE as in Section 3, public encryption keys $\text{ek}_{[n]}$ and $\text{ek}_{[n']}$ of $\mathcal{P}_{[n]}$ and $\mathcal{P}'_{[n']}$.

Inputs of each $\mathcal{P}_i \in \mathcal{P}_{[n]}$: a private decryption key dk_i , and a (non-private) κ -sized list $\mathbb{L}_i \in \{\mathcal{C}^n, \perp\}^\kappa$. The \perp entries may possibly be filled later during the Refresh.

Outputs of each $\mathcal{P}'_j \in \mathcal{P}'_{[n']}$: a (non-private) κ' -sized list \mathbb{L}'_j , initialized empty: $\{\perp\}^{\kappa'}$ // \mathcal{P}'_j keeps filling \mathbb{L}'_j , until possibly receiving input (shutoff).

Encrypted Resharing. For every $c_{[n]} \in \mathcal{C}^n$ which is at some point in its list \mathbb{L}_i , each $\mathcal{P}_i \in \mathcal{P}_{[n]}$ does:

1. Decrypt c_i into $s_i \in \mathbb{S}$ and generate a Shamir sharing $(s_{i \rightarrow j})_{j \in [n']}$ of s_i , i.e. sample a random degree t' polynomial $h_i \in \mathbb{S}[X]_{\leq t}$ s.t. $h_i(0) = s_i$ and set $s_{i \rightarrow j} = h_i(j)$ for all $j \in [n']$.
2. Sample encryption randomnesses $(\rho_j)_{j \in [n']}$, and generate a ciphertext of each coordinate $s_{i \rightarrow j}$ under the public key of $\mathcal{P}'_j \in \mathcal{P}'_{[n']}$: $c_{i \rightarrow j} = \text{Enc}_{\text{ek}'_j}(s_{i \rightarrow j}; \rho_j)$, $\forall j \in [n']$. Denote $c_{i \rightarrow [n']}$ = $(c_{i \rightarrow j})_{j \in [n']}$, the n' -sized vector of ciphertext shares obtained.
3. Generate a NIZK AoK, denoted $\pi_{\text{res},i}$, of: $\text{dk}_i \in \mathcal{dK}$, a degree- t' polynomial h_i and $(\rho_j)_{j \in [n']}$ s.t. $s_i = \text{Dec}(\text{dk}_i, c_i)$, $h_i(0) = s_i$, $\text{ek}_i = \text{EKGen}(\text{dk}_i)$ and $c_{i \rightarrow j} = \text{Enc}_{\text{ek}'_j}(h_i(j); \rho_j)$ and $\|\rho_j\| \leq R_{\text{enc}}$, $\forall j \in [n']$.
4. Generate a signature σ_i on $c_{[n]}$.
5. Multicast $(c_{[n]}, c_{i \rightarrow [n]}, \pi_{\text{res},i}, \sigma_i)$ to $\mathcal{K}'_{[\kappa']}$.

Selection & Combination. Each $\mathcal{K}'_{k'} \in \mathcal{K}'_{[\kappa']}$:

1. Waits until it receives, for some $(t+1)$ -sized subset $\mathcal{U} \subset [n]$ of $\mathcal{P}_{[n]}$, tuples of the form:

$$(c_{[n]}, c_{i \rightarrow [n]}, \pi_{\text{res},i}, \sigma_i), \quad \forall i \in \mathcal{U}.$$

all with the *same* $c_{[n]}$, and such that: (a) all $t+1$ NIZK proofs $\pi_{\text{res},i}$ are verified and (b) all $t+1$ signatures σ_i are valid.

2. Combines the $t+1$ signatures on $c_{[n]}$ into $\text{qvc} = \{\sigma_i\}_{i \in \mathcal{U}}$, thereby obtaining a verified old sharing $(c_{[n]}, \text{qvc})$.

3. Computes homomorphically the Lagrange linear combination of Equation (1):

$$(2) \quad c'_j := \bigoplus_{i \in \mathcal{U}} (\lambda_i^{\mathcal{U}} \boxtimes c_{i \rightarrow j}) \quad \forall j \in [n'],$$

set $c'_{[n']} \leftarrow (c'_j)_{j \in [n']}$. //see also Figures 15 & 12.

4. Multicasts to $\mathcal{P}'_{[n']}$ the *proven new sharing*: $(c'_{[n]}, \{c_{i \rightarrow [n]}, \pi_{\text{res},i}\}_{i \in \mathcal{U}}, (c_{[n]}, \text{qvc}))$ then terminates.

Continuously growing outputs of $\mathcal{P}'_{[n']}$. Any player $\mathcal{P}'_j \in \mathcal{P}'_{[n']}$, upon receiving a tuple $(c'_{[n]}, \{c_{i \rightarrow [n]}, \pi_{\text{res},i}\}_{i \in \mathcal{U}}, (c_{[n]}, \text{qvc}))$ from a collector $\mathcal{K}'_{k'}$, and if $\mathbb{L}'_i[k] = \perp$, then does the following:

1. Check (a') all the proofs $\{\pi_{\text{res},i}\}_{i \in \mathcal{U}}$, (b') all the signatures $\text{qvc} = \{\sigma_i\}_{i \in \mathcal{U}}$ and (c') correctness of the computation of Equation (2). If all checks pass, **accept** $c'_{[n']}$ and set $\mathbb{L}'_j[k] \leftarrow c'_{[n]}$.

Algorithm 2: $\mathcal{P}_{[n]} := \mathcal{P}_{[n_e]}^e$ is the old committee, $\mathcal{P}'_{[n']} := \mathcal{P}_{[n_{e+1}]}^{e+1}$ the new one.

4.3 Open We describe the instructions for any player \mathcal{P}_i^e in any arbitrary committee $\mathcal{P}_{[n_e]}^e$. For every non- \perp entry $(c_{[n_e]})$ which is included at some point in one's $(\kappa_e$ -sized) list \mathbb{L}_i^e //possibly included *later* than reception of open-sig, do:

- generate a decryption s_i of one's encrypted share c_i of $c_{[n_e]}$, which we dub as an *opening share*. Generate a NIZK of correct decryption, denoted $\pi_{\text{dec},i}$. Precisely, it is a NIZK AoK of $\text{dk}_i^{(e)}$ such that: $\text{EKGen}(\text{dk}_i^{(e)}) = \text{ek}_i^{(e)}$ and $\text{Dec}(\text{dk}_i^{(e)}, c_i) = s_i$;
- send to \mathcal{L} , over private channel, the triple $(c_{[n_e]}, s_i, \pi_{\text{dec},i})$.

Upon receiving any $t_e + 1$ such triples for some same $c_{[n_e]}$ from a $t_e + 1$ -subset of some $\mathcal{P}_{[n_e]}^e$, indexed by $\mathcal{U} \subset [n_e]$, \mathcal{L} outputs the Lagrange linear combination $s = \sum_{i \in \mathcal{U}} \lambda_i^{\mathcal{U}} s_i$ of the $(s_i)_i$.

4.4 Latency and Correctness. In Lemma 4 we prove that the latency of a Refresh is 2δ , as claimed in Theorem 1. From it we deduce Proposition 5, which states that every collective opening eventually delivers an output to \mathcal{L} . In Lemma 6 we prove the *correctness invariant* //which will enable the simulation of opening shares in the UC proof. From it we deduce Proposition 7: \mathcal{L} outputs the secret which was committed. Both Propositions 5 and 7 will be useful in the UC proof //to guarantee that \mathcal{L} outputs the same value, and within the same *finite* delay, in both the real and ideal executions.

Lemma 4 (Refresh in 2δ). *Consider two consecutive committees $\mathcal{P}_{[n]}$ and $\mathcal{P}'_{[n']}$, such that at least $t+1$ players in $\mathcal{P}_{[n]}$ are honest and at least one collector $\mathcal{K}'_{k' \in [k']}$ is honest and, at some point in time, the common set of shares invariant holds for $\mathcal{P}_{[n]}$ //and they all received (refresh-sig) and none received shutoff-sig. Then, the invariant holds for $\mathcal{P}'_{[n']}$ at most 2δ later //unless some $\mathcal{P}'_{i \in [n']}$ received (shutoff-sig) in-between.*

Proof. At the considered point in time, there exists at least one index $k \in [k]$ and a common $c_{[n]} \in \mathcal{C}^n$ such that all honest players of $\mathcal{P}_{[n]}$ have their $\mathbb{L}_i[k] = c_{[n]}$ //without knowing which k (s)! and have multicast a resharing of $c_{[n]}$. So, before δ , the honest $\mathcal{K}'_{k'}$ receives $t+1$ resharings of $c_{[n]}$ such that checks (a) and (b) pass. Then it forms a *proven new sharing*, i.e., a tuple such that checks (a'), (b') and (c') pass; then multicasts it. Every honest player $\mathcal{P}'_j \in \mathcal{P}'_{[n']}$ receives it before δ then immediately accepts and includes in its list the $c'_{[n']}$ enclosed: $\mathbb{L}'_j[k'] \leftarrow c'_{[n']}$. \square

Proposition 5 (Liveness). *Consider any execution in which \mathcal{D} is honest and any epoch e_o such that, $\forall 1 \leq e \leq e_o$: there are at least $t_e + 1$ honest players in $\mathcal{P}_{[n_e]}^e$ and at least one honest collector in \mathcal{K}_k^e and $\forall e < e_o$, all honest players of $\mathcal{P}_{[n_e]}^e$ received (refresh-sig) and all honest players of $\mathcal{P}_{[n_{e_o}]}^{e_o}$ received (open-sig) //and no player received (shutoff-sig) . Then \mathcal{L} eventually outputs.*

Proof. Since \mathcal{D} is honest, $\text{RB}^{\mathcal{D} \rightarrow \mathcal{P}_{[n_1]}^1}$ guarantees that all honest players of $\mathcal{P}_{[n_1]}^1$ will receive its vector of ciphertext shares in finite time. So the *common set of shares invariant* holds for $\mathcal{P}_{[n_1]}^1$. By induction on $1 \leq e \leq e_o - 1$, applying Lemma 4, the invariant holds for e_o . So at some finite point in time, there exists an index k^{e_o} and a vector of ciphertext shares $c_{[n]}^{e_o}$ such that every honest $\mathcal{P}_i^{(e_o)} \in \mathcal{P}_{[n_{e_o}]}^{e_o}$ has its $\mathbb{L}_i^{e_o} = c_{[n]}^{e_o}$ //and received (open-sig). From there, within δ , \mathcal{L} receives $t_{e_o} + 1$ proven decryption shares from honest players then outputs. \square

Lemma 6 (Correctness invariant). *Assume that, for some e , there are at most t_e corrupt players in $\mathcal{P}_{[n_{e'}]}^{e'} \forall 1 \leq e' \leq e$. Consider any honest player \mathcal{P}_i^e (if any) of which the list \mathbb{L}_i^e has a non-empty entry. Consider any such entry: $c_{[n_e]}$. Then:*

[A] *There is a secret, s , which was committed by \mathcal{D} ;*

[B] *Denote as $\tilde{s} \in \mathbb{F}_p$ the value of which the plaintexts of $c_{[n_e]}$ are the shares (mod p), then $\tilde{s} = s$.*

Proof. We prove the statement of the lemma by induction on e . It trivially holds for $e=1$. Let us assume that it holds up to some committee $\mathcal{P}_{[n_e]}^e = \mathcal{P}_{[n]}$, and let it show it for committee $\mathcal{P}_{[n_{e+1}]}^{e+1} = \mathcal{P}'_{[n']}$. Either all lists of honest players of $\mathcal{P}'_{[n']}$ are empty, then nothing is to be proven. Or, consider any non-empty entry $c'_{[n']}$ in the list of some honest player $\mathcal{P}'_j \in \mathcal{P}'_{[n']}$. Since it followed the protocol, it must have received a tuple: $(c'_{[n]}, \{c_i \rightarrow [n'], \pi_{\text{res},i}\}_{i \in \mathcal{U}}, (c_{[n]}, \text{qvc}))$ such that all checks (a',b',c') pass.

From the $t+1$ signatures (b'), we deduce that $c_{[n]}$ was in the list of at least one honest player in $\mathcal{P}_{[n]}$. We deduce, from the recursion assumption, that [A] \mathcal{D} shared a secret, s , and that [B] $c_{[n]}$ is a vector of ciphertext shares of s . Now, denote as $(s_i)_{i \in [n]}$ the plaintext shares of $c_{[n]}$. (a') guarantees that, for all $i \in \mathcal{U}$, there exists resharing: $s_{i \rightarrow [n]}$ of s_i , of which $c_{i \rightarrow [n']}$ is the encryption. (c') guarantees that $c'_{[n']}$ is their correct homomorphic Lagrange linear combination. Since all encryptions are proven with bounded randomnesses, i.e., $\|\rho_j\| \leq R_{\text{enc}} \forall i, j$, we conclude that $c'_{[n]}$ is a vector of ciphertext shares of the same secret, s as $c_{[n]}$. //For this conclusion, we refer to Section 2.2, then below Definition 3. Figures 15 & 12 may further help. \square

Proposition 7. Assume that \mathcal{L} outputs, and denote $\mathcal{P}_{[n e_A]}^{e_A}$ the committee from which it receives the $(t_{e_A}+1)$ triples triggering the output. Assume that there are at most t_e corrupt players in $\mathcal{P}_{[n e]}^e \forall 1 \leq e \leq e_A$. Then there is a secret s which was committed by \mathcal{D} , and the output of \mathcal{L} is equal to s .

Proof. Since \mathcal{L} receives $t_{e_A}+1$ triples $(c_{n e_A}, s_i, \pi_{\text{dec}, i})$ for the same $c_{n e_A}$, then, since one of the senders is honest, it must be that $c_{n e_A}$ was in its list. So the conclusions of Lemma 6 apply. \square

4.5 Communication Complexity We consider a $\text{Refresh}(\mathcal{P}_{[n]}, \mathcal{P}'_{[n']})$. The worst-case complexity of the Resharing step is at most $n \cdot \kappa' \cdot O(\kappa \cdot n' \gamma)$ [Each $(\mathcal{P}_i)_{i \in [n]}$ sends to each $(\mathcal{K}'_{k'})_{k \in [n']}$ at most κ encrypted resharing, each of size $O(n' \gamma)$]. Let us take as starting point the moment (known to none) when all honest players of $\mathcal{P}_{[n]}$ received a common vector of ciphertext shares in their lists \mathbb{L}_i [with respect to their previous resharing as entering], and all received **refresh-sig**. Assuming they are at least $t+1$, then it takes no more than δ until each honest collector $\mathcal{K}'_{k'} \in \mathcal{K}'_{[n']}$ receives $t+1$ resharing messages. It is then able to form its message and multicast it to $\mathcal{P}'_{[n']}$. The size of this message is dominated by the batch of $t+1$ resharing attached. Such a batch has size $O(\gamma(t+1)n')$, where, for simplicity we assumed that ciphertexts and signatures [and commitments, in the generalizations of Sections 7.3 and 7.4] have size $O(\gamma)$. Hence, we recover the dominating term of the communication complexity: $\kappa' \cdot \text{MC}_{n'}(O(\gamma(t+1)n'))$ claimed below Theorem 1. As soon as an honest collector multicasts its message, it takes no more than δ until honest players in $\mathcal{P}'_{[n']}$ receive one vector of ciphertext shares in common. In conclusion, we arrived at the starting point for the next Refresh.

5 Proof of Theorem 1

5.1 Roadmap Following [Can01], we say that a protocol APSS0 *UC-emulates* the dummy protocol of Section 2.5, involving $\mathcal{F}_{\text{P-AVSS}}$, if there exists a PPT machine \mathcal{S} denoted *simulator*, also known as “ideal adversary”, as follows. Further helpful diagrams are provided in Figs. 10 and 11. It must be that for every PPT *environment* \mathcal{Z} , that fully controls a “dummy” adversary \mathcal{A} [i.e. \mathcal{A} is a proxy of \mathcal{Z}] and which may send inputs to honest participants [to the dealer \mathcal{D} : a secret s ; to players: signals (**refresh-sig**), (**open-sig**) and (**shut-off**)] and which is forwarded their outputs in real time by honest participants [for the learner \mathcal{L} : $\tilde{s} \in \mathcal{S}$, for players: (**committed**)] has negligible advantage in distinguishing between the following two executions:

- $\text{REAL}_{\mathcal{A}}$: an actual execution of the protocol APSS, with dummy adversary \mathcal{A} fully controlled by \mathcal{Z} , and functionalities $\mathcal{F}_{\text{bPKI}}$, $\text{RB}^{\mathcal{D} \rightarrow \mathcal{P}'_{[n]}}$, \mathcal{F}_{ST} , \mathcal{F}_{AT} , $\mathcal{F}_{\text{NIZK}}$, as depicted in Fig. 10;
- $\text{IDEAL}_{\mathcal{F}_{\text{P-AVSS}}, \mathcal{S}}$: an execution denoted as *ideal*, where \mathcal{S} interacts with \mathcal{Z} on behalf of \mathcal{A} . On the other side, \mathcal{S} interacts with $\mathcal{F}_{\text{P-AVSS}}$ (Section 2.5 then Fig. 5) on behalf of the corrupt participants, and also of \mathcal{A} , as depicted in Fig. 11. The honest dummy participants //i.e., the honest ones among: \mathcal{D} , \mathcal{L} and players $\mathcal{P}_i^e \forall i, e$ are connected to \mathcal{Z} as in a real execution. But on the other side, they only interact with $\mathcal{F}_{\text{P-AVSS}}$, with which they perform the dummy protocol, as described in Section 2.5.

5.2 Simulators \mathcal{S} for a corrupt \mathcal{D} . The simulator \mathcal{S} for a corrupt \mathcal{D} and a corrupt \mathcal{L} simply simulates honest players following the protocol, and ideal functionalities behaving as specified. It instructs $\mathcal{F}_{\text{P-AVSS}}$ to deliver (**committed**) when simulated honest players output so. It makes them follow **Refresh** or **Open** when being relayed from $\mathcal{F}_{\text{P-AVSS}}$ the **refresh-sig**'s and **open-sig**'s. By definition the view generated is identical to the one of a real execution.

In the case of an honest \mathcal{L} , the simulator \mathcal{S} simulates in addition an honest \mathcal{L} and takes the following additional steps. First, it computes the plaintext secret s committed by \mathcal{D} //under honest majority in $\mathcal{P}_{[n_1]}^1$: \mathcal{S} simply decrypts the shares of honest players then interpolates s . Else, \mathcal{S} extracts in straight line the shares of corrupt players out of the NIZK AoK's of plaintext knowledge received from \mathcal{D} . It gives s as input to $\mathcal{F}_{\text{P-AVSS}}$. Later in the execution, suppose that there a committee $\mathcal{P}_{[n_{e_0}]}^{e_0}$ for which the conditions of Proposition 5 are matched, in particular, in which a quorum of $t_{e_0} + 1$ honest dummy player request open-req. Then, \mathcal{S} sends delay request to $\mathcal{F}_{\text{P-AVSS}}$, in order to delay the delivery of the output, until the simulated \mathcal{L} outputs. //By Proposition 5, \mathcal{S} needs only sending a polynomial number of delay requests, so this strategy is tractable. When the simulated \mathcal{L} outputs, \mathcal{S} instructs $\mathcal{F}_{\text{P-AVSS}}$ to immediately release the secret to the dummy \mathcal{L} //via (open-order). In conclusion, both the dummy honest \mathcal{L} and the simulated \mathcal{L} output at the same time. //Notice that we will use again this strategy in the case of an honest \mathcal{D} , as described below then further in Appendix C.4.

Moreover, both output values are the same //if all committees so far had $< t_e$ corruptions, then both are equal to s by Proposition 7. Else, \mathcal{S} uses its power to tamper with the value stored by $\mathcal{F}_{\text{P-AVSS}}$, in order to adjust it with the one opened to the simulated \mathcal{L} . In conclusion, the view of \mathcal{Z} in the ideal execution is the same as in a real one.

5.3 Simulators \mathcal{S} for an honest \mathcal{D} . We describe the simulator for an honest \mathcal{D} and a corrupt \mathcal{L} . It is the hardest one, since it has to simulate decryption shares. Inline we sketch the adaptations for the easier case of an honest \mathcal{L} We describe the simulator \mathcal{S} for one unique honest dealer \mathcal{D} , and a unique \mathcal{L} . It can be easily generalized to the several openings of several linear combinations of several secrets [the technique, given in [CDN15, p. 127], is that simulated decryption shares are picked uniformly at random in the subspace consistent with all previous openings.]. To ease the notation, we focus on the difficult case of executions with $f_e = t_e$ corruptions in each committee $\mathcal{P}_{[n_e]}^e$ //From $t_e + 1$ corruptions, $\mathcal{F}_{\text{P-AVSS}}$ leaks the secret to \mathcal{S} and authorizes \mathcal{S} to modify its value, so the simulation becomes easy.

The simulator \mathcal{S} is formally described in Algorithm 13 in Appendix C.2. We now convey the main ideas of \mathcal{S} by describing it via a sequence of incremental changes, starting from a real execution. In the last hybrid obtained, the view of \mathcal{Z} is generated solely by interaction with $\mathcal{F}_{\text{P-AVSS}}$, hence what we are describing is a simulator. The full details of the hybrids and the proofs of indistinguishability are in Appendix C.2. We first make the change that $\mathcal{F}_{\text{NIZK}}$ does not verify any witness from honest players.

//In the case of an honest \mathcal{L} , we make the change that \mathcal{L} does not anymore forwards to \mathcal{Z} the actual output in Open. Instead, we make \mathcal{D} give to $\mathcal{F}_{\text{P-AVSS}}$ its input. Then we delay the output from $\mathcal{F}_{\text{P-AVSS}}$ to \mathcal{L} as in Section 5.2. Then, \mathcal{L} notifies to \mathcal{Z} the output received from $\mathcal{F}_{\text{P-AVSS}}$. So the same conclusion as in Section 5.2 applies: by Propositions 5 and 7, both the *value* notified by \mathcal{L} and the *time* at which it is notified, is the same as in the real execution. These change and conclusion are further formalized in Appendix C.4.

We then simulate opening shares as follows, and further formalized in $\text{Hyb}^{\text{ShSim}}$. //These changes can be skipped for an honest \mathcal{L} , until the $\text{Hyb}^{\text{Refresh}}[e', i]$'s below. For any vector of ciphertext shares $c_{[n_e]}$ which is in the list \mathbb{L}_j^e of an honest player \mathcal{P}_j^e which performs Open, we replace the decrypted share s_j which it sends to \mathcal{L} , by the following value \tilde{s}_j . \tilde{s}_j is the evaluation at j of the degree t_e polynomial interpolated at:

- the t_e plaintext shares of corrupt players in $c_{[n_e]}$ //They are computed as follows. $c_{[n_e]}$ was formed as the Lagrange linear combination of $t_{e-1} + 1$ encrypted resharings: $c_{i \rightarrow [n_e]}$. So it remains to compute the plaintext shares, intended to corrupt players, in each $c_{i \rightarrow [n_e]}$. For those generated by honest resharers ($i \in [n_{e-1}] \setminus \mathcal{I}^{e-1}$), we already know the plaintext shares. For those generated by corrupt resharers ($i \in \mathcal{I}^{e-1}$), we extract the plaintext shares from the NIZKs of resharing (Algorithm 2 3). *Under honest majority*, i.e., if $t_{e-1} < n_{e-1}/2$, then we extract them by rewinding the environment. We can do so because what we are describing is an intermediate distribution, not a simulator. *Else*, we extract them in straight-line from $\mathcal{F}_{\text{NIZK}}$;
- and a last interpolation point, equal to the *secret shared in* $c_{[n_e]}$: $s_{c_{[n_e]}}$. //Concretely, we can interpolate $s_{c_{[n_e]}}$ from the t_e previous points, plus one opening share of any honest player.

In conclusion, since the plaintexts of $c_{[n_e]}$ all lie on a polynomial of degree t , the method which we have just described returns the same honest opening shares as in the real execution.

We then change the last interpolation point. Namely, we replace $s_{c_{[n_e]}}$ by the actual secret s leaked by $\mathcal{F}_{\text{P-AVSS}}$. Both values are equal, by correctness of APSS0 (Proposition 7). So the view is unchanged.

Then, in the (cascade of) $\text{Hyb}^{\text{Refresh}}[e', i]$ for each $e' \in [e-1, \dots, 1]$ (in this backwards order) then each $i \in [0, \dots, n]$ we replace the way resharing is carried out. As further formalized in Appendix C.5, for any e and index $i \in [1, \dots, n]$ for which \mathcal{P}_i^e is honest, we replace the share s_i reshared by \mathcal{P}_i^e , by $\tilde{s}_i \leftarrow 0$. This is indistinguishable from the real execution by IND-CPA of encrypted sharing, which is proven in Proposition 8. The intuition of this proposition is simple: assume for a moment that ciphertexts perfectly hide the content of plaintexts. Then, for any two chosen secrets: s_i and 0, any t_e shares of both vary uniformly at random, so the view of \mathcal{Z} has the same distribution. We then make the reduction to the $(n_e - t_e)$ -message-IND-CPA security of the actual encryption scheme PKE. The reason for making these changes in backwards order on e , is that decryption keys of $\mathcal{P}_{[n_{e+1}]}^{e+1}$ are not used anymore when we change the plaintext shared by a \mathcal{P}_i^e , so that we can apply IND-CPA.

Finally, in $\text{Hyb}^{\text{Share}}$ formalized in Appendix C.6, we replace the input of a simulated honest dealer by 0. This is the core of the proof. Thanks to the modifications so far, honest players of $\mathcal{P}_{[n_1]}^1$ do not use their decryption keys anymore. So we can apply IND-CPA of encrypted sharing, with respect to their keys, exactly like in the previous cascade of games. This enables to conclude that the distributions are indistinguishable.

In conclusion, we arrived at a view produced by a machine which interacts only with \mathcal{Z} and $\mathcal{F}_{\text{P-AVSS}}$.

- If the extractions of NIZKs are done in *straight-line*, as described in $\text{Hyb}^{\text{ShSim}}$, then we can conclude that we have now arrived at a simulator for APSS in the $\mathcal{F}_{\text{NIZK}}$ hybrid model. So this concludes the proof
- Otherwise, we then make the following final change, called as $\text{Hyb}_{\text{Open}}^{\text{ShInfer}}$ (inference of corrupt shares), aiming at removing extraction by rewinding \mathcal{Z} . Recall that, if we are in this case, then it must be that we are under honest majority.

Referring to the notation of $\text{Hyb}^{\text{ShSim}}$, we do not anymore extract from NIZK AoKs the shares of corrupt players, in each $c_{i \rightarrow [n_e]}$ issued by a corrupt \mathcal{P}_i^{e-1} . Instead, we decrypt the $t_e + 1$ the shares intended to honest players $c_{i \rightarrow [n_e]}$, then interpolate from them the t_e ones intended to corrupt players. Since $c_{i \rightarrow [n_e]}$ is a vector of ciphertext shares, as ensured by the NIZK AoKs (Algorithm 2 3), we conclude that both methods to compute the t_e shares intended to corrupt players return the same values.

6 APSS = APSS0 + δ -synchronized start & termination

We now introduce a mechanism called Refreshing squad which delivers to players the instructions (*refresh-sig*) and (*shutoff-sig*), that they can subsequently use as inputs in APSS0. Recall that (*shutoff-sig*) instructs a player to *shut-off*, i.e., erase its memory and leave the protocol. Recall, from Section 1.3, that we introduce an additional tuning in APSS0, which is a public parameter $\Delta_{\text{wait}}^e \geq 0$. Upon receiving (*refresh-sig*), a player of some committee $\mathcal{P}_{[n_e]}^e$ *waits* Δ_{wait}^e , then starts a Refresh with the new committee $\mathcal{P}_{[n_{e+1}]}^{e+1}$. The reason for this waiting is that, upon receiving (*refresh-sig*) from Refreshing squad, a player has the guarantee that, *if* the delay $\Delta_{\text{wait}}^e - \delta$ is positive, then within delay δ , all players of the same committee will not be *shut-off* and will have at least one consistent set of shares in common. Therefore, they will be able to *Open*. This is formally captured as the *window of opening* guarantee in Theorem 2.

Optionally, we enable Refreshing squad to also deliver a new kind of output, called (*keys-sig*). When this option is activated, players in each new committee $\mathcal{P}_{[n_{e+1}]}^{e+1}$ in APSS wait for (*keys-sig*), then a further Δ_{wait}^e , before they generate their keys and publish their public key. This enables to *postpone until the last minute* the moment when they generate sensitive information, i.e., their decryption key. //In this respect, Theorem 2 is stated assuming the following two simplifications. First, the publication delay of keys is 0. Second, we set to 0 the delay after which, if a corrupt player has not sent its key to $\mathcal{F}_{\text{bPKI}}$, then $\mathcal{F}_{\text{bPKI}}$ publishes it as \perp . In practice, implementations may leverage as follows the guarantee that Refreshing squad delivers (*keys-sig*) to honest players at most δ from each other. When an honest player of $\mathcal{P}_{[n_e]}^e$ retrieves $t_{e+1} + 1$ keys from the bulletin board, then it sets a time-out estimated larger than δ , then sends its resharing without the shares of unpublished key indices. When new keys show up, it releases the missing encrypted shares. Then, after some further conservative timeout, it considers as \perp the still-unpublished keys. Then, it releases in the clear the missing shares.

6.1 Description of Refreshing squad We now convey the main ideas of Refreshing squad. The details are formalized in Algorithm 14 in Appendix D. Its guarantees are captured by Theorem 2. We give here the intuition of them and why they hold, while the full details can be found in Appendix D.

As stated in Theorem 2, the liveness of Refreshing squad, hence of APSS, holds if all committees of collectors have an honest majority, and, as in APSS0, if all committees $\mathcal{P}_{[n_e]}^e$ of shareholders contain at least $t_e + 1$ honest players. So we now assume $\kappa_e = 2\ell_e + 1$ collectors for each $e \geq 1$. Once a collector in $\mathcal{K}_{[\kappa_e]}^e$ has finished its task, it multicasts a signed message (**done**) to $\mathcal{P}_{[n_{e-1}]}^{e-1}$. A *quorum of collectors certificate* (qkc^e), consists of a set of signatures on (**done**) issued by a quorum of $\ell_e + 1$ collectors in $\mathcal{K}_{[\kappa_e]}^e$. Existence of an honest collector in the quorum guarantees that, within δ , all players of $\mathcal{P}_{[n_e]}^e$ will receive a consistent set of encrypted new shares. //This is where we use the honest majority of collectors assumption. If $\ell_e + 1$ collectors were corrupt, then they could form a rogue qkc^e , thereby triggering too early shut-offs, as we will see. Upon receiving or forming a qkc^e for the first time, a player of $\mathcal{P}_{[n_e]}^e$: forwards it to all $\mathcal{P}_{[n_e]}^e$, and multicasts a signed message (**Ack-qkc**) to $\mathcal{P}_{[n_{e-1}]}^{e-1}$. Then it outputs (**refresh-sig**). We dub this as: the player *receives* (**refresh-sig**) (from Refreshing squad). The forwarding guarantees that honest players of $\mathcal{P}_{[n_e]}^e$ output **refresh-sig** at most δ from each other, thereby guaranteeing the claimed *window of opening*.

A *quorum of refreshers certificate* (qrc^e), consists of a set of signatures on (**Ack-qkc**) issued by a quorum of $t_e + 1$ players of $\mathcal{P}_{[n_e]}^e$. A qrc^e guarantees that, in no longer than δ : all honest players of $\mathcal{P}_{[n_e]}^e$ will have received (**refresh-sig**) and will have received a consistent set of encrypted new shares. Hence, a qrc^e guarantees that the protocol can continue without the help of $\mathcal{P}_{[n_{e-1}]}^{e-1}$. Hence, upon receiving or forming a qrc^e , a player of $\mathcal{P}_{[n_{e-1}]}^{e-1}$ can safely **shut-off**, i.e., outputs (**shutoff-sig**) //we must actually refine this mechanism in two ways, in Refreshing squad. First, any $\text{qrc}^{e'} \geq e-1$ triggers (**shutoff-sig**), not only a qrc^e . Second, \mathcal{P}^{e-1} must forward the $\text{qrc}^{e'}$ to all $\mathcal{P}_{[n_{e-2}]}^{e-2}$ before it outputs **shutoff-sig**. Without both refinements there would be pathological executions, e.g., if $e-1 = 2$, in which all $\mathcal{P}_{[n_2]}^2$ shut-off, while $\mathcal{P}_{[n_1]}^1$ are never delivered any $\text{qrc}^{e'} > 1$, so never shut-off. Optionally, we also specify that a player about to shut-off also forwards the qrc to all its committee. As explained in Case II] of the proof of *Fast shutoff*, not doing so would increase by δ the delay to shut-off in some corner cases.

We finally address the optional last-minute-delivery of (**keys-sig**). Players of $\mathcal{P}_{[n_e]}^e$ forward their qrc^e to $\mathcal{P}_{[n_{e+1}]}^{e+1}$, enabling them to receive it at most δ later. A player of $\mathcal{P}_{[n_{e+1}]}^{e+1}$, upon forming or receiving a qrc^e , forwards it to all $\mathcal{P}_{[n_e]}^e$ and $\mathcal{P}_{[n_{e+1}]}^{e+1}$ then outputs (**keys-sig**).

7 Generalisations and Applications

7.1 Enabling to open linear maps **non-interactively over several secrets.** We consider any number W of dealers: $(\mathcal{D}_w)_{w \in [W]}$, w.l.o.g. with one secret each: $(s^{(w)})_{w \in [W]} \in \mathbb{S}^W$. We consider *any* \mathbb{F}_p -linear map $\Lambda : \mathbb{S}^W \rightarrow \Gamma$, where (Γ, \oplus) is any \mathbb{F}_p -vector space. We now describe how to securely open the evaluation of Λ over shared secrets, then give examples and use-cases in Section 7.2. The baseline method has been well-known since [Ben86]. Namely, each player sends to \mathcal{L} the evaluation of Λ on its shares, then \mathcal{L} computes the Lagrange interpolation of these evaluations, in the target space of Λ .

We now describe how to adapt this baseline to APSS0. The task is not completely trivial, since each secret comes with several sets of consistent shares. Roughly, the main idea is that players treat as a *batch* all shares of all secrets which they received from the same collector. Existence of at least one honest collector \mathcal{K}_k guarantees that all honest players in $\mathcal{P}_{[n]}$ have at least, in common, one *batch* of vector of ciphertext shares of all secrets. It is the batch which they received from \mathcal{K}_k . In more details, dealers **Share** as in Section 4.1, except that they use a BC instead of the RB. This is to prevent players of $\mathcal{P}_{[n_1]}^1$ from waiting forever for possibly inactive corrupt dealers. For each player \mathcal{P}_i^e , the entry $\mathbb{L}_i^e[k]$ in its list contains all the W vectors of ciphertext shares (one for each distributed secret $s^{(w)}$) which it received from the collector \mathcal{K}_k^e in the previous **Refresh**. To open the image under Λ of shared values $(s^{(w)})_{w \in [W]}$, each player \mathcal{P}_i^e , for each entry $\mathbb{L}_i^e[k]$ of its list $(c_{[n_e]}^w)_{w \in [W]}$: decrypts its plaintext new shares into $(s_i^w)_{w \in [W]}$, computes their image: $s_i \leftarrow \Lambda((s_i^w)_{w \in [W]})$ then sends $((c_{[n_e]}^w)_{w \in [W]}, s_i, \text{NIZK})$ to \mathcal{L} [the NIZK proves decryption of the $(c_i^w)_{w \in [W]}$ -then- Λ .] Upon receiving

any set of $t_e + 1$ correct triples with the same $(c_{[n_e]}^w)_{w \in W}$, from some $t_e + 1$ -subset of $\mathcal{P}_{[n_e]}^e$, then \mathcal{L} outputs the Lagrange linear combination of the s_i . //So it does not process more messages than in [BGG+20; GHL22], when opening players behave honestly.

7.2 Integration in existing threshold schemes for signing, VRF, decryption. We give some linear maps $\Lambda : \mathbb{S}^W \rightarrow \Gamma$, and use cases using in black-box their non-interactive opening, enabled by Section 7.1.

Scalar multiplication $s \in \mathbb{F}_p \mapsto s.G$, for threshold coin tossing, decryption and signing. We consider $\Gamma := \mathbb{G}$ a group of order p , in additive notation, and $G \in \mathbb{G}$ a *public* element. The first two applications require DDH in \mathbb{G} . The threshold opening of this map implements by itself: a threshold PRF ([CKS05]), threshold Elgamal decryption ([DF90]) and threshold signing (BLS: [Bol03; TCZ+] and SPS [CKP+22]). //For BLS, spelled out in detail in Appendix F.2.3, careful discussions on the CDH-like assumptions required are provided in [BLS04; Dar10], depending on the type of the bilinear group of which \mathbb{G} is one side.

Mixed addition / scalar multiplications $(S \in \mathbb{G}, (\alpha, \beta) \in \mathbb{F}_p^2) \mapsto S + \alpha.H + \beta.H'$ for VUF signing. \mathbb{G} , of order p and in additive notation, is one side of a type III bilinear group with SXDH. This map is used in [GJM+21, Figure 4, VUF.Sign]. A threshold opening of it generates a signature which publicly proves correctness of an evaluation of their threshold verifiable unpredictable function (VUF).

$S \in \mathbb{G} \mapsto \text{pairing}(Z, S)$ for VUF evaluation. Z is a public group element. Threshold opening is this map ([GJM+21, Figure 4, VUF.Eval]) implements threshold evaluation of a VUF.

$\Lambda : (s, e) \in (\mathbb{F}_p[X]/(X^N + 1))^2 \mapsto c_1.s + c_2 + e$ for RLWE decryption. c_1 and c_2 are public polynomials in a RLWE ring $\mathbb{F}_p[X]/(X^N + 1)$ //The multiplication $s \mapsto c_1.s$ is a modular multiplication by a constant polynomial c_1 , so is indeed \mathbb{F}_p -linear. The threshold opening of this map (followed by a public rescaling-then-rounding) implements threshold decryption of a RLWE-based ciphertext (c_1, c_2) , such as from [BFV/BGV21]. We explain it in more detail in Appendix F.2.4 //Noticeably; we incorporated in Λ a *common secret-shared smudging noise* e . This is an adaptation of the overlooked technique of [GLS15], which enables to keep constant the size of the smudging noise, vs exponential in n when naively smudging the opening shares.

Let us briefly conclude that the above black-box tool for threshold decryption, is essentially the only ingredient needed for YOSO MPC [GHK+21; BDO22]. When the threshold encryption scheme is a FHE, such as [BFV/BGV21], then the above black box tool enables practical constant round MPC [CLO+13; GLS15; Coh16; HHPV21; KJY+20].

7.3 Resharing based on homomorphic commitments to shares, and relaxing PKE. A simple variation enables APSS to also deliver commitments to shares, under a specified commitment scheme Com . The most general case is when we maximally relax the requirement on PKE. Namely, we specify PKE to be only *committing* ([TZ21]), i.e., if an encryptor proves knowledge of a plaintext m , then an honest decrypter should obtain m . Now, what a player \mathcal{P}_i has in its list are pairs of: a *batch* of commitments to subshares $(\text{com}_{v \rightarrow [n]})_{v \in \mathcal{V}}$, and the openings: $(s_{v \rightarrow i}, \widehat{s}_{v \rightarrow i})_{v \in \mathcal{V}}$ of the $(\text{com}_{v \rightarrow i})_{v \in \mathcal{V}}$. The player \mathcal{P}_i reconstructs its share s_i by Lagrange interpolation from the subshares $(s_{v \rightarrow i})_{v \in \mathcal{V}}$. To reshare s_i into $\mathcal{P}'_{[n']}$, \mathcal{P}_i multicasts to the collectors: the *batch* with its signature; a commitment to each subshare: $\text{com}_{i \rightarrow j} = \text{Com}(s_{i \rightarrow j}, \widehat{s}_{i \rightarrow j})$ for $j \in [n']$; and an encryption $c_{i \rightarrow j}$ of the decommitment $(s_{i \rightarrow j}, \widehat{s}_{i \rightarrow j})$ for each $j \in [n']$; appended with the expected NIZK AoK, spelled-out in Appendix F.2.1. //As a side-benefit, removing the LHE condition enables PKE's with packed ciphertexts [GHL22], and removing bilateral binding enables PKE's with selective-opening resilience [HPW15], enabling resilience against early adaptive corruptions (Appendix B.2). The property needed to have simulatability of decryption shares needed in our proof of APSS0, is that the previous commitment scheme: (i) either is Feldman (then $\mathcal{F}_{\text{P-AVSS}}$ should leak the commitments), or, (ii) is preimage-sampleable, as detailed in Appendix F.2.5. Instantiated with a linearly-homomorphic commitment such as Feldman or Pedersen, then \mathcal{P}_i needs not appending the batch $(\text{com}_{v \rightarrow [n]})_{v \in \mathcal{V}}$ to its resharing, instead, it can append the short Lagrange-combination of them $\text{com}_{\mathcal{V} \rightarrow [n]}$ (similarly to APSS).

7.4 APSS2: unbounded correctness and no signatures Consider an instantiation, of the previous variation, with a commitment scheme supporting an unlimited number of homomorphic additions, such as Feldman’s or Pedersen’s. Then we have the invariant that, in every epoch, commitments to the shares interpolate to the initial commitment S to the secret. *Assume* there is a public ledger on which \mathcal{D} could post this commitment before disappearing. Then upon receiving $t+1$ openings of a purported set $\text{com}_{i \in \mathcal{U}}$ of commitments to shares, \mathcal{L} can directly check whether the purported commitments lie on the same degree t polynomial as S . Hence, this upgrade of APSS has correctness even if all players are corrupt. We denote it as APSS2. By the same interpolation-of-commitments mechanism, we see that players need not anymore generate signatures. Indeed, any purported set of commitments to shares can be directly checked against S .

7.5 Initial sharing from DKG. Protocols denoted as *distributed key generation (DKG)*, somehow emulate the initial sharing, from a single dealer to $\mathcal{P}_{[n_1]}^1$, of a random secret. We refer to Appendix F.2.2 for a survey of the guarantees provided by DKGs, including with a biased secret. Each $\mathcal{P}_i^1 \in \mathcal{P}_{[n_1]}^1$ receives a private share s_i of s , denoted as its key share. In addition the DKG publishes a commitment to this share, which is denoted as \mathcal{P}_i ’s *verification key*. A verification key is what enables \mathcal{L} to verify each decryption share. Verification keys can either take the form of ciphertexts of (sub-)shares ([GHK+21]), in which case they can be refreshed along with shares as in APSS0. Or, for concrete applications [HKMR22; CMP20; Gro21], they take the form of Feldman commitments to (sub-)shares, in which case the variants of Sections 7.3 and 7.4 enable their refresh along with shares.

7.6 Integration in the asynchronous threshold Schnorr signing “ROAST” (CCS’22) The threshold Schnorr signature of [RRJ+22, §4] is centralized by a semi-honest *coordinator* C . Its first task is to wait until it receives empty messages from a quorum of $t+1$ (their “ t ”) volunteer signers. Then, it replies by requesting to the volunteers to generate a pre-signature, using their key shares. We make the observation that, in the situation where the signing key is shared within a committee of APSS, *the first empty message to C can be used to decide which common set of shares to use*. Instead of an empty message, players send to C their local lists \mathbb{L}_i ’s. The coordinator C waits until it obtains messages from $t+1$ players with a *common* $c_{[n]}$ in their lists (as granted by Lemma 4), then requests them to generate a pre-signature, *using their key shares of this $c_{[n]}$* .

8 Faster NIZKs of Encrypted Resharing, Implementation

8.1 Basic proof of correct resharing Let us recall the PKE of Elgamal [Elg84]. Let G be any generator of \mathbb{S} . We consider a decryption key $\text{dk} \in \mathbb{F}_p$, and the corresponding encryption key $\text{Ek} := \text{dk}.G$. To encrypt a message $S \in \mathbb{S}$, sample $\rho \xleftarrow{\$} \mathbb{F}_p$ and return $C = (C_1, C_2) = (S + \rho.\text{Ek}, \rho.G) \in \mathcal{C} = \mathbb{S}^2$. To decrypt (C_1, C_2) under key $\text{Ek} = \text{dk}.G$, return $C_1 - \text{dk}.C_2$.

Now we consider a resharer issuing a NIZK AoK as specified in Algorithm 2. We subsequently dub it as a *prover*. We also consider any entity verifying the proof, denoted as the *verifier*. For simplicity we set as $n' = n$ the number of members in the new committee $\mathcal{P}'_{[n']}$. Our main concern is a verifier which is a member \mathcal{P}'_i of the new committee, since it must verify, in Item 1 (a’) of 2, potentially up to κ' batches of $t+1$ proofs of resharing. We consider the instantiation with the Elgamal PKE, which is well-known to be an LHE as specified in Section 3. We use the notation of Section 2.2. The prover has input a ciphertext of its share $C = (C_1, C_2)$ under its public key Ek . Following steps Items 1 and 2, it decrypts this ciphertext, then generates a vector of ciphertext shares $\{(C'_{j,1}, C'_{j,2})\}_{j \in [n]}$ of it under the public keys $\{\text{Ek}'_j\}_{j \in [n]}$ of the new committee. Concretely, it samples $q(\cdot) \xleftarrow{\$} \mathbb{F}_p[X]_t^{(0)}$, i.e. a random polynomial s.t. $q(0) = 0$, and encryption randomnesses $\rho'_1, \dots, \rho'_n \xleftarrow{\$} \mathbb{F}_p^n$. It then returns

$$(3) \quad (C'_{j,1}, C'_{j,2}) = (C_1 - \text{dk}.C_2 + q(j).G + \rho'_j.\text{Ek}'_j, \rho'_j.G), \forall j \in [n].$$

Then it generates the NIZK AoK specified in Item 3. Recall that its purpose is to prove that $(C'_{j,1}, C'_{j,2})_{j \in [n]}$ is formed as in Equation (3). Up to moving the constant C_1 on the right-hand side, this is equivalent to

proving knowledge of $\mathbf{dk} \in \mathbb{F}_p$, encryption randomnesses $\rho'_1, \dots, \rho'_n \in \mathbb{F}_p$ and polynomial $q(\cdot) \in \mathbb{F}_p[X]_t^{(0)}$, such that, denoting as $Y = [(Y_{1,j}, Y_{2,j}) \in (\mathbb{S}^2)^n, Y_0 := \mathbf{Ek}] \in \mathbb{S}^{2n+1}$ the public vector defined by:

$$(4) \quad Y_0 := \mathbf{Ek} \text{ and for } j \in [n], Y_{1,j} := -C_1 + C'_{j,1}; Y_{2,j} := C'_{j,2}.$$

The vector Y is the image of $(\mathbf{dk}, q(\cdot), \rho'_1, \dots, \rho'_n)$ by the following linear map $f : \mathbb{F}_p^{n+t+1} \rightarrow \mathbb{S}^{2n+1}$:

$$(5) \quad f(\mathbf{dk}, q(\cdot), \rho'_1, \dots, \rho'_n) := (\{-\mathbf{dk} \cdot C_2 + q(j) \cdot G + \rho'_j \cdot \mathbf{Ek}'_j, \rho'_j \cdot G\}_{j \in [n]}, \mathbf{dk} \cdot G).$$

So the resharer is brought back to proving knowledge of a preimage of Y by a public linear map. For this task there is a generic Σ protocol, i.e., 3-moves and public coin, which is provided in Fig. 3. It dates back to [Cra96; Iva98], in the context of opening one-way q homomorphisms. As observed in [ACR21, Theorem 1], it actually works for any linear map f . There, it is stated to be perfectly complete, special honest-verifier zero-knowledge and unconditionally special-sound. Hence, as proven in [FKMV12, Th. 3], its transform under Fiat Shamir is (not straight-line) simulation-sound extractable.

In Fig. 3 we also state two possible ways to transform the Σ -protocol by Fiat-Shamir into an NIZK AoK. The left one is the one considered in [CDGK22]. It produces a proof which is, in our context, in $\mathbb{S}^{2n+1} \times \mathbb{F}_p$. This short format is the same as the one of a Schnorr signature (where the LHS space is instead just \mathbb{S}). However, it is incompatible with batch verification techniques. The right one is the one considered in this work. It produces a proof which, in our context, is in $\mathbb{S}^{2n+1} \times \mathbb{F}_p^{n+t+1}$.

Σ -protocol for ZK AoK of a preimage by a homomorphism

Non-interactive argument of knowledge of witness for Y for the relation $R = \{(Y; w) \in \mathcal{Y} \times \mathcal{W} : Y = f(w)\}$

Public parameters: Finite field \mathbb{F}_p , \mathbb{F}_p -vector spaces \mathcal{W}, \mathcal{Y} , linear map $f : \mathcal{W} \rightarrow \mathcal{Y}$, vector $Y \in \mathcal{Y}$.

Public Input: Vector Y , of which the prover claims knowledge of a preimage by f .

Prover's private input: Witness $w \in \mathcal{W}$, s.t. $Y = f(w)$.

Σ -protocol: Prover samples $w^* \xleftarrow{\$} \mathcal{W}$ “the random witness” and sends $Y^* = f(w^*)$ to the verifier. The verifier samples $c \xleftarrow{\$} \mathbb{F}_p$ [the challenge] which it sends to the prover. The prover replies with $z := w^* + cw$. The verifier accepts the proof if $f(z) = Y^* + c \cdot Y$.

Two variants for its Fiat-Shamir transform : both use as public input a (programmable) random oracle $H : \{0, 1\}^* \rightarrow \mathbb{F}_p$. Denote $pp := (\mathcal{W}, \mathcal{Y}, f, Y, H)$.

The FS transform considered in [CDGK22]:

NIZK.Prove_{yoLo}(w, pp, Y, f):

$w^* \xleftarrow{\$} \mathcal{W}, Y^* \leftarrow f(w^*), c \leftarrow H(Y, Y^*), z \leftarrow w^* + c \cdot w$
return $\pi \leftarrow (c, z)$

NIZK.Verify_{yoLo}(pp, Y, f, π)

Parse $\pi = (c, z)$
return accept if $z \in \mathcal{W}$ and $c \leftarrow H(Y, f(z) - c \cdot Y)$.

The FS transform considered in this work:

NIZK.Prove(w, pp, Y, f)

$w^* \xleftarrow{\$} \mathcal{W}, X^* \leftarrow f(w^*), c \leftarrow H(Y, Y^*), z \leftarrow w^* + c \cdot w$
return $\pi \leftarrow (Y^*, z)$

NIZK.Verify(pp, Y, f, π)

Parse $\pi = (Y^*, z)$
return accept if $f(z) = Y^* + c \cdot Y$ for $c \leftarrow H(Y, Y^*)$.

Figure 3: The Σ -protocol is the one considered in [ACR21, Theorem 1], and also in [CDGK22, Fig. 1]. We also compare two variants of its Fiat-Shamir transform into an NIZK AoK. On the left is the version used in [CDGK22, Figure 2] to derive their NIZK of resharing in [CDGK22, Figure 9]. On the right is the version used in this work.

8.2 Optimization for faster verification

Many preimages of a fixed linear map. The proofs and verifications of a number W of resharnings by a single prover, are amortized using the following technique described in [ACF21, Thm. 4]. Instead of

providing W proofs of the form $w_i^* + cw_i$, the resharer provides a single proof $w^* + \sum_{j=1}^W c^j w_i$ with image $Y^* + \sum_{i=1}^W c^i Y_i$, which allows to replace W verifications with a single one.

Adaptation of the technique of batch Schnorr signatures verification. Since each proof opens a linear map, the verification of one such proof consists in checking a set of $2n + 1$ linear combinations. Instead of checking $2n + 1$ equalities, the verifier chooses a random element $r \in \mathbb{F}_p$ and checks at once whether:

$$(6) \quad \sum_{j=0}^{2n+1} r^j (Y_j^* + c.Y_j) = \sum_{j=0}^{2n+1} r^j f_j(z)$$

where Y_j and $f_j(z)$ denote the j -th coordinates of Y and $f(z)$. Recall also that we constructed Y as $Y = [Y_{1,[n]}, Y_{2,[n]}, \text{dk}.G] \in \mathbb{S}^n \times \mathbb{S}^n \times \mathbb{S}$ (Equation (4)), and that f is detailed in Equation (5). This strategy, first described in [BGR98], allows to drastically reduce the complexity of operations in \mathbb{S} : instead of n multi-exponentiations of 4 elements, the only operation involving elements of \mathbb{S} is one multi-exponentiation of $3(2n + 1)$ elements. The same strategy brings a further amortization factor, when applied to $t+1$ proofs of resharing issued by a quorum of $t+1$ resharers, each of which having a specific challenge c . The verifier still checks all the claims at once using one multi-exponentiation with powers of a chosen random element $r \in \mathbb{F}_p$. There is actually a larger amortization surface in APSS0, since a verifier in $\mathcal{P}'_{[n]}$ is forwarded $[k]$ proofs of resharing from each single resharer, each proof with a different challenge. We did not capture this optimization in Table 4 //The reason is that, there, we preferred to display the most common use-case of verifying a quorum of $t+1$ publicly verifiable resharings.

Verifier precomputation The checking made in Equation (6) involves the computation of the scalar $\sum_j r^j q(j)$ for a polynomial $q \in \mathbb{F}_p[X]_{\leq t}$ s.t. $q(0) = 0$. To further speed up the verification process, the computation of this scalar in the latter multi-exponentiation is done in the following way. The verifier precomputes the vector $\sum_j r^j (j^1, \dots, j^t)$; during the verification process, it remains to compute the inner product of this vector with the vector of coefficients of q . This allows to compute only t multiplications in \mathbb{S} during the verification process, instead of nt .

Our code⁷ uses version 0.9.1 of the gnark-crypto [BPH+22] library in Go, with operations in the 254-bit “pairing-friendly” BN254 Barreto-Naehrig curve [BN05] over a 254-bit base field. We use SHA256 for hashing. The timings below were measured on one AMD EPYC 7F72 CPU with 24 cores at 3.2GHz, and 724GB RAM.

In Table 4 the substantial speedup of the verification process using the improvements is shown (under “amortized”), compared with the naive method consisting in using the protocol described in [CDGK22, Figure 2] (under “naive”). Precisely, the latter uses the variant of Fiat-Shamir on the left of Fig. 3, without any of the three amortization techniques of Section 8.2. The number $n = 2t + 1$ still stands for the number of players in the new committee, i.e. the number of public keys. W stands for the number of proofs per prover, i.e., of reshared secrets in parallel. The timings in the table below are given for verification of batches of $t + 1$ resharings.

	$n = 11$		$n = 51$		$n = 101$		$n = 1001$	
	Amortized	Naive	Amortized	Naive	Amortized	Naive	Amortized	Naive
$W = 1$	0.009 ms	0.05 ms	0.046 ms	0.462 ms	0.151 ms	1.79 ms	10.8 ms	178 ms
$W = 50$	0.079 ms	1.53 ms	0.718 ms	23.1 ms	2.64 ms	89.5 ms	179 ms	8.90 s
$W = 1000$	0.83 ms	24.3 ms	10.7 ms	464 ms	37.1 ms	1.79 s	3.35 s	178 s

Table 4: Verification time, given a new committee of n parties, for $t + 1$ proofs of W claims each, using amortizations described above (compared to straightforward separate verification using [CDGK22])

We also implemented APSS0 using the alternative PKE and NIZK of resharing denoted “DHPVSS” in [CDGK22, p. 5.2] (provided $\deg(m^*) \leq n - t - 1$). DHPVSS has a proof of size of only 3 group elements and turned out to be 2.5 times faster. But we do not further report on it, since “DHPVSS” requires a one-time-pad-sized PKI.

⁷available at https://gitlab.com/levrat.christophe/apss_proof_verification

References

- [ACF21] T. Attema, R. Cramer, and S. Fehr. “Compressing Proofs of k-Out-Of-n Partial Knowledge”. In: *CRYPTO*. 2021.
- [ACR21] T. Attema, R. Cramer, and M. Rambaud. “Compressed Σ -Protocols for Bilinear Group Arithmetic Circuits and Application to Logarithmic Transparent Threshold Signatures”. In: *ASIACRYPT (Eprint 2020/1447 v4)*. 2021.
- [ADD+19] I. Abraham, S. Devadas, D. Dolev, K. Nayak, and L. Ren. “Synchronous Byzantine Agreement with Expected $O(1)$ Rounds, Expected $O(n^2)$ Communication, and Optimal Resilience”. In: *Financial Cryptography*. 2019.
- [Ami22] Amis. *implementation of CMP MPC with echo broadcast*. <https://github.com/getamis/alice/tree/master/crypto/tss/ecdsa/cggmp>. 2022.
- [AS20] J.-P. Aumasson and O. Shlomovits. *Attacking Threshold Wallets*. Eprint 2020/1052, presentation at RWC’21 and at BlackHat’20. Also summarized on Kudelski’s blog. 2020.
- [BBB+18] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. “Bulletproofs: Short proofs for confidential transactions and more”. In: *IEEE S&P*. 2018.
- [Ben86] J. C. Benaloh. “Secret Sharing Homomorphisms: Keeping Shares of a Secret Secret (Extended Abstract)”. In: *CRYPTO*. 1986.
- [BFV/BGV21] A. Kim, Y. Polyakov, and V. Zucca. “Revisiting Homomorphic Encryption Schemes for Finite Fields”. In: *ASIACRYPT*. 2021.
- [BGG+20] F. Benhamouda, C. Gentry, S. Gorbunov, S. Halevi, H. Krawczyk, C. Lin, T. Rabin, and L. Reyzin. “Can a Public Blockchain Keep a Secret?” In: *TCC*. 2020.
- [BGR98] M. Bellare, J. A. Garay, and T. Rabin. “Fast batch verification for modular exponentiation and digital signatures”. In: *EUROCRYPT*. 1998.
- [BLL+21] F. Benhamouda, T. Lepoint, J. Loss, M. Orrù, and M. Raykova. “On the (in)security of ROS”. In: *EUROCRYPT*. 2021.
- [BLS04] D. Boneh, B. Lynn, and H. Shacham. “Short Signatures from the Weil Pairing”. In: *J. Cryptol.* (2004).
- [BN05] P. S. Barreto and M. Naehrig. “Pairing-friendly elliptic curves of prime order”. In: *SAC*. 2005.
- [Bol03] A. Boldyreva. “Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme”. In: *PKC*. 2003.
- [BPH+22] G. Botrel, T. Piellard, Y. E. Housni, A. Tabaie, and I. Kubjas. *ConsensSys/gnark-crypto: v0.6.1*. 2022.
- [BTA+19] S. Basu, A. Tomescu, I. Abraham, D. Malkhi, M. K. Reiter, and E. G. Sirer. “Efficient Verifiable Secret Sharing with Share Recovery in BFT Protocols”. In: *CCS*. 2019.
- [Can01] R. Canetti. “Universally composable security: A New Paradigm for Cryptographic Protocols”. In: *FOCS*. 2001.
- [CCL+20] G. Castagnos, D. Catalano, F. Laguillaumie, F. Savasta, and I. Tucker. “Bandwidth-efficient threshold EC-DSA”. In: *PKC*. 2020.
- [CD20] I. Cascudo and B. David. “ALBATROSS: publicly Attestable BATched Randomness based On Secret Sharing”. In: *ASIACRYPT*. 2020.
- [CDD+99] R. Cramer, I. Damgård, S. Dziembowski, M. Hirt, and T. Rabin. “Efficient Multiparty Computations Secure Against an Adaptive Adversary”. In: *EUROCRYPT*. 1999.
- [CDDS85] B. A. Coan, D. Dolev, C. Dwork, and L. J. Stockmeyer. “The Distributed Firing Squad Problem (Preliminary Version)”. In: *STOC*. 1985.
- [CDGK22] I. Cascudo, B. David, L. Garms, and A. Konring. “YOLO YOSO: Fast and Simple Encryption and Secret Sharing”. In: *ASIACRYPT*. 2022.
- [CDN15] R. Cramer, I. B. Damgård, and J. B. Nielsen. *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, 2015.
- [CGG+20] R. Canetti, R. Gennaro, S. Goldfeder, N. Makriyannis, and U. Peled. “UC Non-Interactive, Proactive, Threshold ECDSA with Identifiable Aborts”. In: *CCS*. 2020.
- [CGM19] Y. Chen, N. Genise, and P. Mukherjee. “Approximate Trapdoors for Lattices and Smaller Hash-and-Sign Signatures”. In: *ASIACRYPT*. 2019.

- [CH20] A. Choudhury and A. Hegde. “High Throughput Secure MPC over Small Population in Hybrid Networks (Extended Abstract)”. In: *INDOCRYPT*. 2020.
- [Cho20] A. Choudhury. “Improving the Efficiency of Optimally-Resilient Statistically-Secure Asynchronous Multi-party Computation”. In: *INDOCRYPT*. 2020.
- [CKLS02] C. Cachin, K. Kursawe, A. Lysyanskaya, and R. Strobl. “Asynchronous Verifiable Secret Sharing and Proactive Cryptosystems”. In: *ACM CCS*. 2002.
- [CKMR22] S. Coretti, A. Kiayias, C. Moore, and A. Russell. “The Generals’ Scuttlebutt: Byzantine-Resilient Gossip Protocols”. In: *CCS*. 2022.
- [CKP+22] E. Crites, M. Kohlweiss, B. Preneel, M. Sedaghat, and D. Slamanig. *Threshold Structure-Preserving Signatures*. ePrint 2022/839. 2022.
- [CKS05] C. Cachin, K. Kursawe, and V. Shoup. “Random Oracles in Constantinople: Practical Asynchronous Byzantine Agreement Using Cryptography”. In: *J. Cryptol.* (2005).
- [CMP20] R. Canetti, N. Makriyannis, and U. Peled. *UC Non-Interactive, Proactive, Threshold ECDSA*. ePrint 2020/492, merged into CCS’20. 2020.
- [Coh16] R. Cohen. “Asynchronous Secure Multiparty Computation in Constant Time”. In: *PKC*. 2016.
- [Coi22] Coinbase. *announcing CMP MPC with echo broadcast*. 2022.
- [Cra96] R. Cramer. “Modular Design of Secure yet Practical Cryptographic Protocols”. PhD thesis. CWI and University of Amsterdam, 1996.
- [Dar10] S. C. and Darrel Hankerson and Edward Knapp and Alfred Menezes. “Comparing two pairing-based aggregate signature schemes”. In: *Des. Codes, Cryptogr.* (2010).
- [DF90] Y. Desmedt and Y. Frankel. “Threshold cryptosystems”. In: *CRYPTO*. 1990.
- [DG23] Q. Dao and P. Grubbs. “Spartan and Bulletproofs are simulation-extractable (for free!)” In: *EUROCRYPT*. 2023.
- [Div22] S. K. and Divya Ravi and Sophia Yakoubov. *Towards Efficient YOSO MPC Without Setup*. Eprint 2022/187. 2022.
- [DKI+23] B. David, A. Konring, Y. Ishai, E. Kushilevitz, and V. Narayanan. *Perfect MPC over Layered Graphs*. ePrint 2023/330. 2023.
- [DLS88] C. Dwork, N. Lynch, and L. Stockmeyer. “Consensus in the Presence of Partial Synchrony”. In: *J. ACM* (1988).
- [DPSZ12] I. Damgård, V. Pastro, N. Smart, and S. Zakarias. “Multiparty Computation from Somewhat Homomorphic Encryption”. In: *CRYPTO*. 2012.
- [Elg84] T. Elgamal. “A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms”. In: *CRYPTO*. 1984.
- [Fir22] Fireblocks. *”MPC key shares are automatically refreshed in minutes-long intervals”*. 2022.
- [FKMV12] S. Faust, M. Kohlweiss, G. A. Marson, and D. Venturi. “On the Non-malleability of the Fiat-Shamir Transform”. In: *INDOCRYPT*. 2012.
- [GDK22] C. U. Günther, S. Das, and L. Kokoris-Kogias. *Practical Asynchronous Proactive Secret Sharing and Key Refresh*. Cryptology ePrint Archive, Paper 2022/1586. 2022.
- [GGOR13] J. A. Garay, C. Givens, R. Ostrovsky, and P. Raykov. “Broadcast (and Round) Efficient Verifiable Secret Sharing”. In: *ICITS*. 2013.
- [GHK+21] C. Gentry, S. Halevi, H. Krawczyk, B. Magri, J. B. Nielsen, T. Rabin, and S. Yakoubov. “YOSO: You Only Speak Once: Secure MPC with Stateless Ephemeral Roles”. In: *CRYPTO*. 2021.
- [GHL22] C. Gentry, S. Halevi, and V. Lyubashevsky. “Practical Non-interactive PVSS with Thousands of Parties”. In: *EUROCRYPT*. 2022.
- [GJKR07] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. “Secure Distributed Key Generation for Discrete-Log Based Cryptosystems”. In: *J. Cryptol.* (2007).
- [GKM+22] V. Goyal, A. Kothapalli, E. Masserova, B. Parno, and Y. Song. “Storing and Retrieving Secrets on a Blockchain”. In: *PKC*. 2022.
- [GL02] S. Goldwasser and Y. Lindell. “Secure Computation without Agreement”. In: *DISC*. 2002.
- [GP21] C. Ganesh and A. Patra. “Optimal Extension Protocols for Byzantine Broadcast and Agreement”. In: *Distrib. Comput.* (2021).

- [GPS19] Y. Guo, R. Pass, and E. Shi. “Synchronous, with a Chance of Partition Tolerance”. In: *CRYPTO*. 2019.
- [Gro22] T. Group. *Implementation of CMP*. 2022.
- [HJKY95] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung. “Proactive Secret Sharing”. In: *CRYPTO*. 1995.
- [HKMR22] S. Halevi, H. Krawczyk, A. Miao, and T. Rabin. “Threshold Cryptography as a Service (in the Multiserver and YOSO Models)”. In: *CCS*. 2022.
- [HPW15] C. Hazay, A. Patra, and B. Warinschi. “Selective Opening Security for Receivers”. In: *ASIACRYPT*. 2015.
- [HZC+22] B. Hu, Z. Zhang, H. Chen, Y. Zhou, H. Jiang, and J. Liu. “DyCAPS: Asynchronous Proactive Secret Sharing for Dynamic Committees”. In: *eprint* (2022).
- [ISO19] ISO/IEC. *ISO/IEC 18033-6:2019 IT Security techniques — Encryption algorithms — Part 6: Homomorphic encryption*. 2019. URL: <https://www.iso.org/obp/ui/#iso:std:iso-iec:18033:-6:ed-1:v1:en%7D>.
- [Iva98] R. C. and Ivan Damgård. “Zero-Knowledge Proofs for Finite Field Arithmetic; or: Can Zero-Knowledge be for Free?” In: *CRYPTO*. 1998.
- [JS20] A. P. Jonas Nick and G. Sanders. *Liquid: A Bitcoin Sidechain*. <https://blockstream.com/assets/downloads/pdf/liquid-whitepaper.pdf>. 2020.
- [KG20] C. Komlo and I. Goldberg. “FROST: Flexible Round-Optimized Schnorr Threshold Signatures”. In: *SAC*. 2020.
- [KJY+20] E. Kim, J. Jeong, H. Yoon, Y. Kim, J. Cho, and J. H. Cheon. “How to Securely Collaborate on Data: Decentralized Threshold HE and Secure Key Update”. In: *IEEE Access* (2020).
- [KKK08] J. Katz, C. Koo, and R. Kumaresan. “Improving the Round Complexity of VSS in Point-to-Point Networks”. In: *ICALP*. 2008.
- [KMM+23] A. Kate, E. V. Mangipudi, P. Mukherjee, H. Saleem, and S. A. K. Thyagarajan. *Non-interactive VSS using Class Groups and Application to DKG*. eprint 2023/451. 2023.
- [KMS20] E. Kokoris Kogias, D. Malkhi, and A. Spiegelman. “Asynchronous Distributed Key Generation for Computationally-Secure Randomness, Consensus, and Threshold Signatures.” In: *CCS*. 2020.
- [LNR18] Y. Lindell, A. Nof, and S. Ranellucci. “Fast Secure Multiparty ECDSA with Practical Distributed Key Generation”. In: *CCS*. 2018.
- [MCK20] A. Momose, J. P. Cruz, and Y. Kaji. *v3 2020/04/19 of Hybrid-BFT: Optimistically Responsive Synchronous Consensus with Optimal Latency or Resilience*. ePrint 2020/406. 2020.
- [MZW+19] S. K. D. Maram, F. Zhang, L. Wang, A. Low, Y. Zhang, A. Juels, and D. Song. “Churp: Dynamic-committee proactive secret sharing”. In: *CCS*. 2019.
- [OY91] R. Ostrovsky and M. Yung. “How to Withstand Mobile Virus Attacks”. In: *PODC*. 1991.
- [PCR13] A. Patra, A. Choudhury, and C. P. Rangan. “Efficient Asynchronous Verifiable Secret Sharing and Multiparty Computation”. In: *J Cryptol* (2013).
- [PCR14] A. Patra, A. Choudhury, and C. P. Rangan. “Asynchronous Byzantine Agreement with optimal resilience”. In: *Distributed Computing* (2014).
- [PCRR09] A. Patra, A. Choudhary, T. Rabin, and C. P. Rangan. “The Round Complexity of Verifiable Secret Sharing Revisited”. In: *CRYPTO*. 2009.
- [Rei94] M. K. Reiter. “Secure Agreement Protocols: Reliable and Atomic Group Multicast in Rampart”. In: *CCS*. 1994.
- [RRJ+22] T. Ruffing, V. Ronge, E. Jin, J. Schneider-Bensch, and D. Schröder. “ROAST: Robust Asynchronous Schnorr Threshold Signatures”. In: *CCS*. 2022.
- [SBKN21] 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).
- [Set20] S. T. V. Setty. “Spartan: Efficient and General-Purpose zkSNARKs Without Trusted Setup”. In: *CRYPTO*. 2020.
- [Sta96] M. Stadler. “Publicly Verifiable Secret Sharing”. In: *EUROCRYPT*. 1996.
- [TCZ+] A. Tomescu, R. Chen, Y. Zheng, I. Abraham, B. Pinkas, G. Golan-Gueta, and S. Devadas. “Towards Scalable Threshold Cryptosystems”. In: *IEEE S&P*.

- [TZ21] A. Takahashi and G. Zaverucha. *Verifiable Encryption from MPC-in-the-Head*. eprint 2021/1704. 2021.
- [VAFB22] R. Vassantlal, E. Alchieri, B. Ferreira, and A. Bessani. “COBRA: Dynamic Proactive Secret Sharing for Confidential BFT Services”. In: *IEEE S&P*. 2022.
- [WXSD20] J. Wan, H. Xiao, E. Shi, and S. Devadas. “Expected Constant Round Byzantine Broadcast Under Dishonest Majority”. In: *TCC*. 2020.
- [YXD22] Y. Yan, Y. Xia, and S. Devadas. *Shanrang: Fully Asynchronous Proactive Secret Sharing with Dynamic Committees*. ePrint 2022/164. 2022.
- [YXXM23] T. Yurek, Z. Xiang, Y. Xia, and A. Miller. “Long Live The Honey Badger: Robust Asynchronous DPSS and its Applications”. In: *Usenix*. 2023.
- [ZSV05] L. Zhou, F. B. Schneider, and R. Van Renesse. “APSS: Proactive secret sharing in asynchronous systems”. In: *ACM TISSEC* (2005).

A Complements on the model

A.1 Ideal functionalities

A.1.1 Formalizing eventual delivery We now explain the high level idea of the mechanism which we use to formalize eventual delivery, following [KMTZ11; CGHZ16]. Every ideal functionality \mathcal{F} , when it needs to eventually deliver (ssid, v) to some entity P , engages in the following interaction. It notifies \mathcal{A} of the output id (ssid) , initializes a counter $D_{\text{ssid}} \leftarrow 1$, which captures the delivery delay. Upon receiving (delay) from \mathcal{A} , it sets $D_{\text{ssid}} \leftarrow D_{\text{ssid}} + 1$. Upon receiving (fetch) from P , it sets $D_{\text{ssid}} \leftarrow D_{\text{ssid}} - 1$, as well as for all other counters related to pending outputs for P . In addition, we specify that it leaks (fetch) to \mathcal{A} . This precision is not present in previous works [KMTZ11; CGHZ16; LLM+20]. We added it, since otherwise we did not see how \mathcal{A} could accurately schedule the delivery of outputs. Precisely, it would not know how many (fetch) were done so far, so would not know how many (delay) it must make to delay the output sufficiently long, and would also not know when a message is delivered. It is left implicit that entities fetch as much as they can all. Since \mathcal{A} is PPT, at some point it gets exhausted of pressing the button “ delay ”. So, after sufficiently many fetch es more, the counter drops down to 0. Then \mathcal{F} can deliver (ssid, v) to P .

In $\mathcal{F}_{\text{P-AVSS}}$, we formalized that it is instead the players requesting an opening which have the power to request the decrease of the delivery delay D , 1 by 1 each time. We could have also, instead, followed the formalism of all previous works, and have instead the learner \mathcal{L} which would have made the delay to drop by -1 upon requesting fetch . Likewise, we formalized that the players of an old committee $\mathcal{P}_{[n_e]}^e$, requesting a Refresh , have the power to request the decrease of the refresh delay D_e , 1 by 1 each time. We could have well have, instead, given this power to the players of the new committee $\mathcal{P}_{[n_{e+1}]}^{e+1}$.

This mechanism was first introduced in [KMTZ11, §3.2] for secure channels. Then it was extended by [CGHZ16] for secure function evaluation. We kept the more explicit formalism of [KMTZ11; LLM+20], in which \mathcal{A} is required to request delay each time it wants to augment by one the delay, instead of entering a new delay in unary notation, as specified in [CGHZ16].

Finally, we made the following addition to these models. We specified that the functionality notifies \mathcal{A} each time an honest recipient calls (fetch) . That way, the adversary knows at any moment in time when an output will be delivered. In particular, it knows when an output is delivered. This addition is mere formalism, since all previous works not in UC explicitly assume this power.

A.1.2 Bookkeeping of corruptions (and of shut-off) by ideal functionalities. Functionalities keep track of the corrupt entities: \mathcal{D} and/or \mathcal{L} when the case, as well as both \mathcal{I}^e and the corrupt \mathcal{K}_k^e , for all e . This is enabled by slight additions to the UC model made in [CDN15, §4] [KMTZ11, footnote 8]. In addition, we assume that $\mathcal{F}_{\text{P-AVSS}}$ is informed in real time when a dummy honest player receives (shut-off) . This can simply be implemented by specifying, in the dummy protocol, that a dummy honest player notifies to $\mathcal{F}_{\text{P-AVSS}}$ the reception of (shut-off) , just before it does shut-off .

A.1.3 Ideal functionality of (asynchronous proactive) verifiable secret sharing. It is formalized in Fig. 5. It maintains a counter, denoted as e_o , which is roughly the highest committee such that in all previous committees, there is a quorum of dummy honest players which queried a Refresh, in the form of a (refresh-req) request. In more detail, the adversary has the power to *delay* the increment of the counter e_o , with the same kind of fetch-and-delay mechanism as described in Appendix A.1.1. More precisely, what plays the role of fetch here are the (refresh-req). To this enable this role, we add the formal precision to the dummy protocol, that upon receiving refresh-sig or open-sig, a dummy honest player sends refresh-sig or open-sig to $\mathcal{F}_{\text{P-AVSS}}$ as much as it can. This has for effect to decrease by one, each time, the delays of Refresh or of delivery of the output.

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

Sharing On input (“share”, $s \in \mathbb{S}$) from \mathcal{D} for the first time, or possibly from \mathcal{A} if \mathcal{D} is corrupt: stores $\tilde{s} \leftarrow s$, and eventually-delivers (committed) to each player $\mathcal{P}_i^1 \in \mathcal{P}_{[n_1]}^1$. // “eventually-delivers” consists of the same fetch-and-delay mechanism as explained in Appendix A.1.1.

Then, initialize two counters: $e_{\mathcal{A}} \leftarrow 1$. // any $t_e + 1 - f_e$ honest players of any $\mathcal{P}_{[n]}^{e \leq e_{\mathcal{A}}}$ can open \tilde{s} , if the adversary allows it.

and $e_o \leftarrow 0$. // any $t_e + 1$ honest players of any $\mathcal{P}_{[n]}^{e \leq e_o}$ can open \tilde{s} . If $e_o = 0$ then no committee has this power.

\mathcal{A} Delaying eventual resharing and delivery

- Initialize $D \leftarrow 1$ //delivery delay
- Initialize $D_e \leftarrow 1, \forall e$. //resharing delay
- Upon receiving delay or delay_e from \mathcal{A} , set $D \leftarrow D+1$ or $D_e \leftarrow D_e + 1$

Bookkeeping requests from honest players

- Initialize $\text{HResharers}_e \leftarrow \{\}, \text{HOpeners}_e \leftarrow \{\}, \forall e$.
- For any e , upon receiving (open-req) from any honest player $\mathcal{P}_i^e \in \mathcal{P}_{[n_e]}^e$: set $\text{HOpeners}_e \leftarrow \text{HOpeners}_e \cup \{\mathcal{P}_i^e\}$, set $D \leftarrow D - 1$ and leak (open-req, \mathcal{P}_i^e) to \mathcal{A} .
- For any e , upon receiving (refresh-req) from any honest player $\mathcal{P}_i^e \in \mathcal{P}_{[n_e]}^e$: set $\text{HResharers}_e \leftarrow \text{HResharers}_e \cup \{\mathcal{P}_i^e\}$, set $D_e \leftarrow D_e - 1$ and leak (refresh-req, \mathcal{P}_i^e) to \mathcal{A} .
- Upon being notified that a player \mathcal{P}_i^e is shut-off, remove \mathcal{P}_i^e from HOpeners_e and HResharers_e .

Opening

- [Early opening] **For any $e \leq e_{\mathcal{A}}$** , if $|\text{HOpeners}_e| \geq t_e + 1 - f_e$ and if some s is stored, then:
 - (i) if \mathcal{L} is corrupt, leak \tilde{s} to \mathcal{A} ;
 - (ii) if \mathcal{L} is honest, upon receiving (open-order) from \mathcal{A} , if no output was delivered yet to \mathcal{L} , deliver \tilde{s} to \mathcal{L} .
- [Collective opening] If $|\text{HOpeners}_e| \geq t_e + 1$ for some $e \leq e_o$ and $D_e \leq 0$ and no output was delivered yet to \mathcal{L} , then deliver \tilde{s} to \mathcal{L} .

Allowing new committees to open

- [Initial AVSS] When $t_1 + 1$ honest players of $\mathcal{P}_{[n_1]}^1$ have received (committed), set $e_o \leftarrow \max(e_o, 1)$.
- [Early refreshing] If $|\text{HResharers}_{e_{\mathcal{A}}}| \geq t_{e_{\mathcal{A}}} + 1 - f_{e_{\mathcal{A}}}$ and if some s is stored, then, upon subsequently receiving (refresh-order) from \mathcal{A} : set $e_{\mathcal{A}} \leftarrow e_{\mathcal{A}} + 1$.
- [Collective refreshing] If $|\text{HResharers}_{e_o}| \geq t_{e_o} + 1$ and $D_{e_o} \leq 0$, set $e_o \leftarrow e_o + 1$.

Corruptions above thresholds

- [Privacy break-down] If there exists $e \leq e_{\mathcal{A}}$, such that the number of corruptions in $\mathcal{P}_{[n_e]}^e$ is $f_e \geq t_e + 1$, then leak \tilde{s} to \mathcal{A}
- [Correctness break-down] If there exists $e \geq 1$, such that the number of corruptions in $\mathcal{P}_{[n_e]}^e$ is $f_e \geq t_e + 1$, then allow \mathcal{A} to modify the value of the stored \tilde{s} and to open it to \mathcal{L} at any time. [Correctness break-down is thwarted by APSS2.]
- [Liveness break-down] Let e the minimum epoch number for which all collectors $\mathcal{K}_{[n_e]}^e$ are corrupt, then never allow e_o to go above $e - 1$. [Notice also that, for some $e \leq e_o$, if $t_e \geq n_e/2$, then the specifications above enable collective opening or re-sharing from $\mathcal{P}_{[n_e]}^e$ only if \mathcal{A} {corrupted or shut-off} $< t_e$ players in $\mathcal{P}_{[n_e]}^e$]

Figure 5: Ideal functionality of proactive AVSS.

A.1.4 Bulletin board of public keys: $\mathcal{F}_{\text{bPKI}}$ We present in Fig. 6 the ideal functionality of a bulletin board of public keys, denoted as $\mathcal{F}_{\text{bPKI}}$. Upon receiving a key ek_i^e from any player $\mathcal{P}_i^e \in \mathcal{P}_{[n_e]}^e$, it stores

$(\mathcal{P}_i^e, \text{ek}_i)$ and leaks this information to \mathcal{A} . For each e it: -waits until it received a public key from every honest player $\mathcal{P}_i^e \in \mathcal{P}_{[n_e]}^e$ -sets a timeout -then after it elapsed, sets to \perp the keys of the (necessarily corrupt) players of $\mathcal{P}_{[n_e]}^e$ which did not give a key. Then it sets as $\text{ek}_{[n_e]} \leftarrow (\text{ek}_i)_{i \in [n_e]}$ the vector of all keys, [where those from some *corrupt* players may be \perp], and eventually delivers it to all the system.

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

Output format Initialize an empty vector $\text{ek}_{[n]} = \{\top\}^n$.
When all $\text{ek}_i \neq \top, \forall i \in [n]$, set `output-available = true`.

Formalizing eventual delivery For every $R \in \mathcal{R}$, initialize a counter $D_R \leftarrow 0$ //the delivery delay.
Initialize `output-available` \leftarrow false
Upon receiving `fetch` from any $R \in \mathcal{R}$, $D_R \leftarrow D_R - 1$. When $D_R = 0$ for the first time, if no output was delivered yet to R , wait until `output-available = true` then deliver $\text{ek}_{[n]}$ to R .

Formalizing timeout for keys of corrupt players Initialize a counter $T_{\mathcal{A}} \leftarrow 1$ //the timeout.
Upon receiving `delay-keys` from \mathcal{A} , $T_{\mathcal{A}} \leftarrow T_{\mathcal{A}} + 1$.
Upon receiving `fetch` from any $R \in \mathcal{R}$, $T_{\mathcal{A}} \leftarrow T_{\mathcal{A}} - 1$.
If $T_{\mathcal{A}} = 0$, freeze forever $T_{\mathcal{A}} = 0$. Then, for all $i \in \mathcal{I}$: if $\text{ek}_i = \top$, then set $\text{ek}_i \leftarrow \perp$.

Honest keys registration Upon receiving the first message $(\text{Register}, \widetilde{\text{ek}}_i)$ from an honest key-holder P_i , send $(\text{Registered}, P_i, \widetilde{\text{ek}}_i)$ to \mathcal{A} and set $\text{ek}_i \leftarrow \widetilde{\text{ek}}_i$.

Corrupt keys registration Upon receiving a message $(\text{Register}, (\widetilde{\text{ek}}_i \neq \top)_{i \in \mathcal{I}})$ from \mathcal{A} and if $T_{\mathcal{A}} > 0$, then: set $\text{ek}_i \leftarrow \widetilde{\text{ek}}_i \forall i \in \mathcal{I}$.

Figure 6: The bulletin board of public keys functionality $\mathcal{F}_{\text{bPKI}}$, parametrized by a set of n key-holders, denoted as $\mathcal{P}_{[n]}$, of which the corrupt ones are indexed by $\mathcal{I} \subset [n]$, and by a set of receivers \mathcal{R} . It does not perform any check on the keys received. In APSS, a published key which is not in $\epsilon\mathcal{K}$ is automatically considered as \perp by honest players.

A.1.5 Non-interactive zero-knowledge arguments of knowledge (NIZK-AoK) We present in Algorithm 7 the ideal functionality of *non-interactive zero-knowledge arguments of knowledge* (NIZK-AoK), denoted as $\mathcal{F}_{\text{NIZK}}$. It is mainly borrowed from [GOS06], with differences which we highlight. It is parametrized by a NP relation \mathcal{R} . Upon request of a prover P exhibiting some public input x and knowledge of some secret witness w , it verifies if $(x, w) \in R$ then deletes w from its memory. If the verification passes, then $\mathcal{F}_{\text{NIZK}}$ eventually outputs to P a string π . We denote Π the space of such strings π . During the delay of output, \mathcal{A} has the power to set the value of π . If it does not use this power, then $\mathcal{F}_{\text{NIZK}}$ sets π to a default value π_0 . //This is a difference with [GOS06], in which \mathcal{A} could delay forever the delivery of π . Sticking to this model would have prevented us from specifying a functionality $\mathcal{F}_{\text{P-AVSS}}$ with guaranteed output delivery (GOD) in case of honest majority. Another way around this problem is proposed in [CEK+16], in which \mathcal{A} must answer in priority to requests from functionalities, such as $\mathcal{F}_{\text{NIZK}}$, which model purely local computations. However, it was not clear to us to what extend their adversary could choose not to answer any further request at all, thereby preventing GOD in APSS0 under honest majority. Notice that we neglect the delay of $\mathcal{F}_{\text{NIZK}}$ in the latency bound of Lemma 4.

Upon subsequent input the same string π and x from any verifier V , $\mathcal{F}_{\text{NIZK}}$ then confirms to the verifier that P knows some witness for x . //Again, there is a potentially infinite delay in [GOS06], in the case where no (x, π) is recorded, during which $\mathcal{F}_{\text{NIZK}}$ expects to receive from \mathcal{A} a possible witness for x . We again transformed this into a time-out, based on the same fetch-and-delay mechanism.

Under honest majority, it is possible to instantiate $\mathcal{F}_{\text{NIZK}}$ under standard assumptions, from the sole setup of non-interactive publications on a bulletin board $\mathcal{F}_{\text{bPKI}}$, using so-called multi-string NIZKs [GO07].

A.1.6 Asynchronous message transmitting We formalize in Fig. 8 the ideal functionality of asynchronous *public authenticated* message transmitting with eventual delivery delay, denoted as \mathcal{F}_{AT} . It is

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

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

Proof: On input $(\text{prove}, \text{sid}, \text{ssid}, x, w)$ from P , ignore if $(x, w) \notin R$. Request (proof, x) to \mathcal{A} then go to the next step.

Reception of the NIZK Initialize a counter $T_{\mathcal{A}} \leftarrow 1$ //the timeout.

Upon receiving delay from \mathcal{A} , $T_{\mathcal{A}} \leftarrow T_{\mathcal{A}} + 1$.

Upon receiving fetch from P , $T_{\mathcal{A}} \leftarrow T_{\mathcal{A}} - 1$.

Upon receiving (π) from \mathcal{A} and if $T_{\mathcal{A}} > 0$, then: freeze forever store (x, π) and deliver $(\text{proof}, \text{sid}, \text{ssid}, \pi)$ to P .

If $T_{\mathcal{A}} = 0$, then: freeze forever $T_{\mathcal{A}} = 0$, store (x, π) and deliver $(\text{proof}, \text{sid}, \text{ssid}, \pi_0)$ to P .

Verification: On input $(\text{verify}, \text{sid}, \text{ssid}, x, \pi)$ from any verifier V , check whether (x, π) is stored. If not, then do the following instructions:

- request (verify, x, π) to \mathcal{A} ;
- initiate a counter D_{verif} which \mathcal{A} can increase by $+1$ steps, and V by -1 steps;
- upon receiving an answer $(\text{witness}, w)$ from \mathcal{A} and if $D_{\text{verif}} > 0$ and if $(x, w) \in R$, then: store (x, π) ;
- when $D_{\text{verif}} = 0$, halt those instructions and go to the next (and last) step.

If (x, π) is stored, return $(\text{verification}, \text{sid}, \text{ssid}, 1)$ to V , else return $(\text{verification}, \text{sid}, \text{ssid}, 0)$.

Algorithm 7: Non-interactive zero-knowledge functionality

parametrized by a sender S and a receiver R , hence the terminology “authenticated”. It delivers every message sent within a finite delay D , hence the terminology *eventual delivery*, although D can be adaptively increased by \mathcal{A} . It leaks the content of every message to \mathcal{A} , hence the terminology “public”. We also define a stronger variant, denoted \mathcal{F}_{ST} for *secure transmitting*, which leaks only the length of the messages. Only messages intended to \mathcal{L} will go through this stronger variant.

 $\mathcal{F}_{\text{AT}}/\mathcal{F}_{\text{ST}}$

- Upon receiving a message (send, m) from S , initialize $D_{\text{mid}} \leftarrow 1$, where mid is a unique message ID, store $(\text{mid}, D_{\text{mid}}, m)$ and leak $(\text{mid}, D_{\text{mid}}, m)$ to \mathcal{A} . \mathcal{F}_{ST} leaks only $(\text{mid}, D_{\text{mid}}, |m|)$.
- Upon receiving a message (fetch) from R :
 1. Set $D_{\text{mid}} \leftarrow D_{\text{mid}} - 1$ for all mid stored, and leak (fetch) to \mathcal{A} .
 2. If $D_{\text{mid}} = 0$ for some stored $(\text{mid}, D_{\text{mid}}, m)$, deliver the message m to R and delete (mid, m) from the memory.
- Upon receiving a message $(\text{delay}, \text{mid})$ from \mathcal{A} , for some stored mid , set $D_{\text{mid}} \leftarrow D_{\text{mid}} + 1$.
- (Adaptive message replacement) Upon receiving a message $((\text{mid}, m) \rightarrow m')$ from \mathcal{A} , if S is corrupt and the tuple $(\text{mid}, D_{\text{mid}} > 0, m)$ is stored, then replace the stored tuple by $(\text{mid}, D_{\text{mid}}, m')$.

Figure 8: Ideal functionality of asynchronous *public authenticated* message transmitting with eventual delivery delay, parametrized by sender S and receiver R . The straightforward upgrade to obtain asynchronous *secure* message transmitting \mathcal{F}_{ST} is described inline.

Our baseline for \mathcal{F}_{ST} is the functionality denoted $\mathcal{F}_{\text{ed-smt}}$ [KMTZ11]. For \mathcal{F}_{AT} , we made the addition to leak the contents of the messages to \mathcal{A} . We also incorporated two other additions, borrowed from the \mathcal{F}_{NET} in [LLM+20]. The first consists in attaching a unique identifier to each message and counter, in order to give to \mathcal{A} a control on the delay of each message individually. Notice that [CGHZ16] model this individual control by, instead, given the power to \mathcal{A} to re-order messages not delivered yet. The second addition consists in forcing explicitly \mathcal{A} to press (delay) to augment the delay by $+1$, instead of the (equivalent) formalization in which \mathcal{A} enters the additional delay in unary notation.

A.1.7 Reliable broadcast (RB) ideal functionality The following functionality for RB, denoted as $\text{RB}^{S \rightarrow \mathcal{R}}$, proceeds simply as follows. On receiving a message s from the sender S , it sends s to each receiver $R \in \mathcal{R}$ by using the same procedure as \mathcal{F}_{AT} . Notice that in protocol APSS0, the sender is \mathcal{D} and will use $\text{RB}^{\mathcal{D} \rightarrow \mathcal{P}_{[n_1]}^1}$ only once. There exists another functionality for RB in [CP22; AAPP22], which is denoted as $\mathcal{F}_{\text{ACast}}$. The difference is that for every given $R \in \mathcal{R}$, $\mathcal{F}_{\text{ACast}}$ may *never* deliver s to R if the adversary does not allow to, notwithstanding other honest players could have been delivered s . The reason is that they use the classical UC framework of *delayed output* of Canetti, in which the delivery of every single output from a functionality needs to be allowed by the adversary.

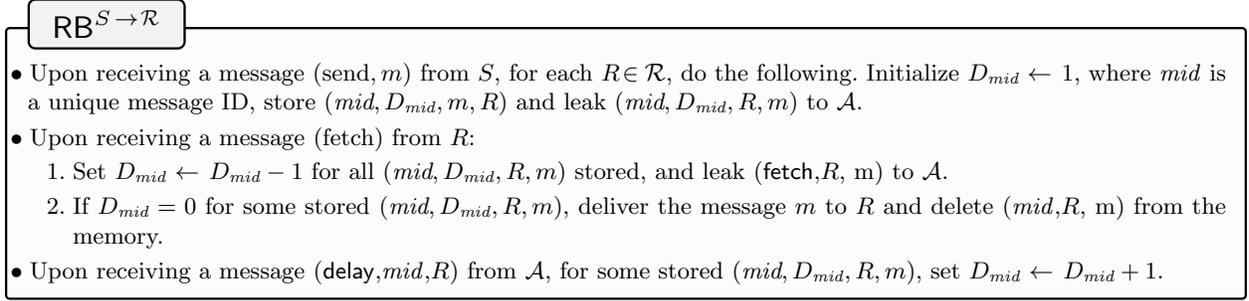


Figure 9: Ideal functionality of reliable broadcast. It is parametrized by a sender S and a set of receivers \mathcal{R} . The straightforward upgrade to obtain asynchronous *secure* message transmitting \mathcal{F}_{ST} is described inline.

B Complements on APSS0: Q & A

B.1 Illustration of a resharing in APSS0 Fig. 12 illustrates $\text{Refresh}(\mathcal{P}_{[n]}, \mathcal{P}'_{[n']})$ in APSS0 between an old committee $\mathcal{P}_{[n]} := \mathcal{P}_{[n_e]}^e$ and a new one $\mathcal{P}'_{[n']} := \mathcal{P}_{[n_{e+1}]}^{e+1}$.

B.2 How to compile protocol APSS0 of Section 6 into the $\mathcal{F}_{\text{bPKI}}$ model of Section 2, in which public keys are not public parameters In the description of APSS0 made in Section 6, we considered as public parameters encryption keys $\text{ek}_{[n]}$ and $\text{ek}_{[n']}$ of $\mathcal{P}_{[n]}$ and $\mathcal{P}'_{[n']}$. To compile APSS0 in the $\mathcal{F}_{\text{bPKI}}$ model, we let players of each committee $\mathcal{P}_{[n]}$ generate key pairs $(\text{dk}_i, \text{ek}_i)$ and publish ek_i to $\mathcal{F}_{\text{bPKI}}$. Then, each player of an old committee waits to receive from $\mathcal{F}_{\text{bPKI}}$ the list of all the keys of the new committee (or of committee $\mathcal{P}'_{[n]}$ for \mathcal{D}) before starting a Refresh with the new committee. Likewise, each player of a new committee waits to receive from $\mathcal{F}_{\text{bPKI}}$ the list of all the keys of the old committee before taking any action. Recall that $\mathcal{F}_{\text{bPKI}}$ always delivers a list of keys in finite time. Indeed, it is specified in Appendix A.1.4 to assign a \perp key to a corrupt player if it did not receive any key from it before a timeout.

B.3 What happens if some honest player does not publish its keys “in time” ? By definition, $\mathcal{F}_{\text{bPKI}}$ does not return a vector of keys until it has received all those of *honest* players. This model of $\mathcal{F}_{\text{bPKI}}$ is implicit in the mainstream so-called “PKI model”, such as considered in [DMR+21]. The definition of $\mathcal{F}_{\text{bPKI}}$ which we make, is essentially the minimum one with which it is possible to instantiate MPC protocols in the PKI model. //Notice that since our $\mathcal{F}_{\text{bPKI}}$ does not deliver keys simultaneously, it does not even enable players to start synchronously [CDN15, p89].

Let us illustrate on an example why this model of $\mathcal{F}_{\text{bPKI}}$ is indeed minimal. Consider, the following protocol, which is the simplest MPC protocol which actually uses the PKI model. A dealer reliably broadcasts a publicly verifiably encrypted secret sharing (PVSS) of its input s , with threshold t out of n players. Any $t+1$ out of n honest players can reconstruct s to any entitled \mathcal{L} , by sending verifiable decryptions of their

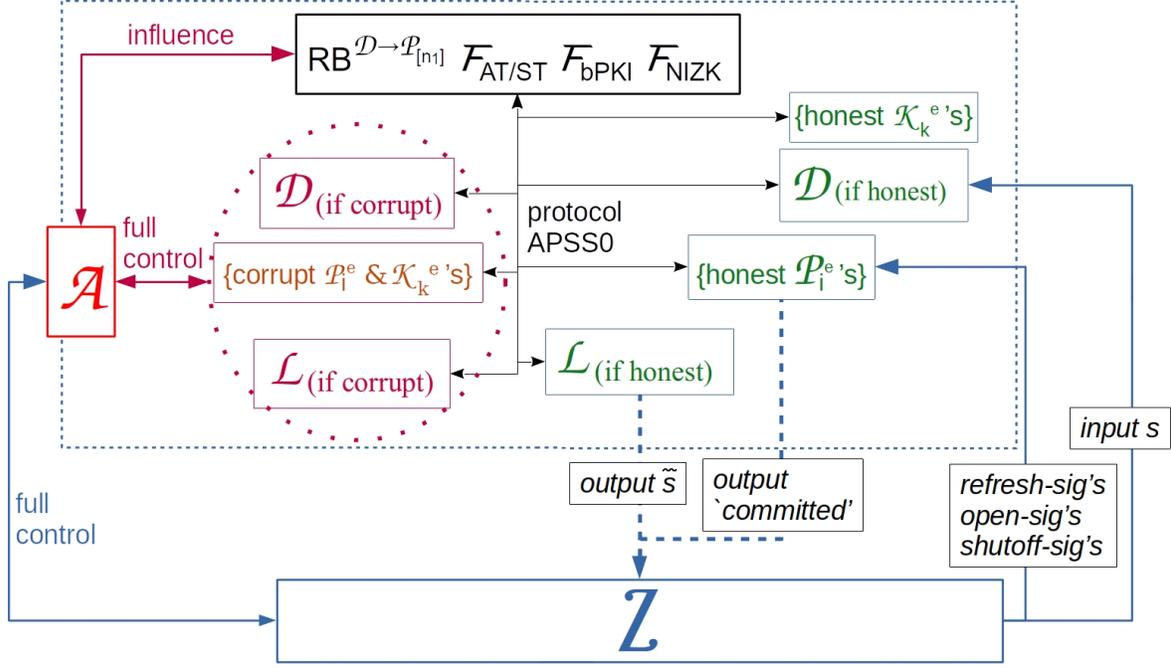


Fig. 10: $REAL_{\mathcal{A}}$ execution of APSS0 with dummy adversary \mathcal{A} . The Environment \mathcal{Z} interacts with the system (the big rectangle in dotted blue) by: its full control on \mathcal{A} , its power to give an input to honest participants: (s to \mathcal{D} if it is honest, and signals to players), and its power to learn the outputs (\tilde{s} from \mathcal{L} if it is honest, and (committed) from honest players of $\mathcal{P}_{[n_1]}^1$).

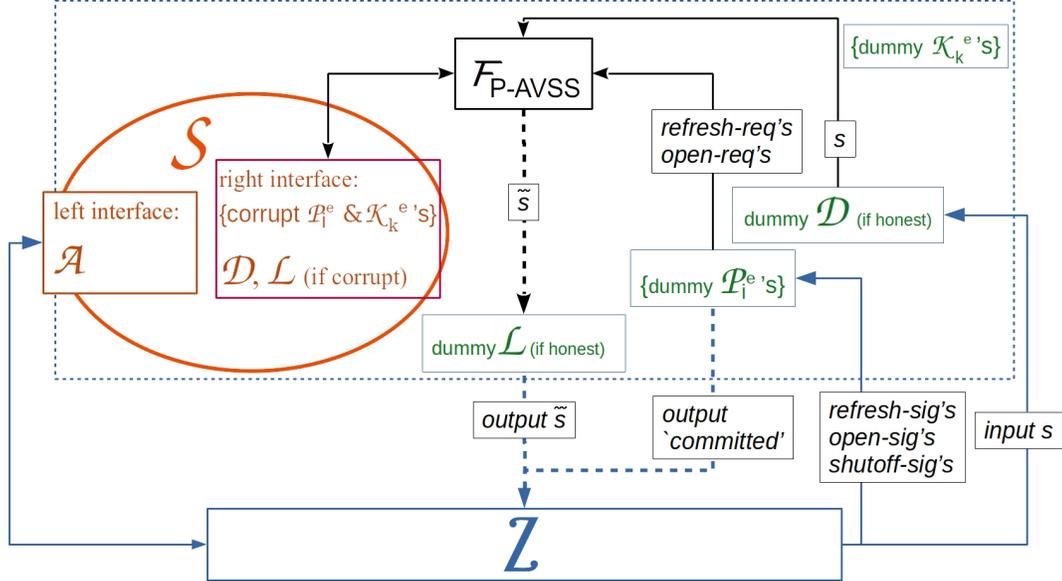


Fig. 11: $IDEAL_{\mathcal{F}_{P-AVSS}, \mathcal{S}}$ execution: dummy honest participants perform the dummy protocol with \mathcal{F}_{P-AVSS} ; the simulator \mathcal{S} interacts on the right with \mathcal{F}_{P-AVSS} with the same interface as \mathcal{A} and corrupt entities in the dummy protocol, and on the left, interacts with \mathcal{Z} with the same interface as the dummy adversary in the real protocol.

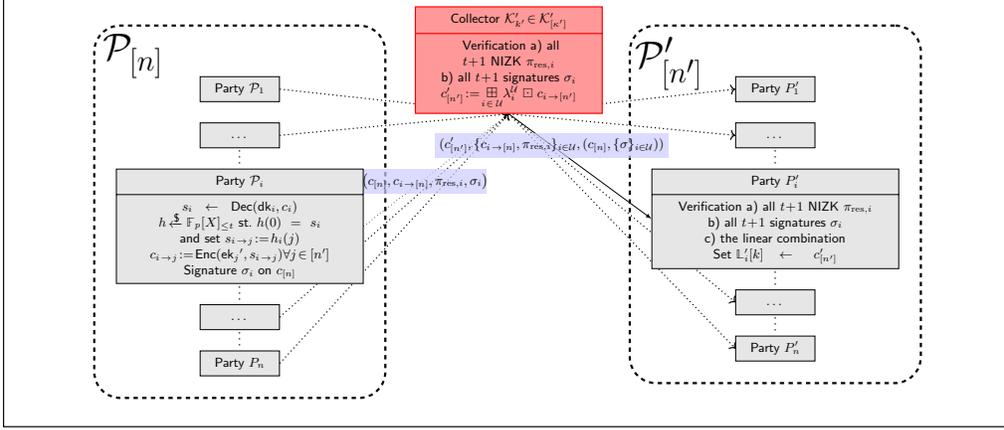


Fig. 12: Refresh($\mathcal{P}_{[n]}, \mathcal{P}'_{[n']}$) in APSS0. $\mathcal{P}_{[n]} := \mathcal{P}_{[n_e]}^e$ is the old committee, $\mathcal{P}'_{[n']} := \mathcal{P}_{[n_{e+1}]}^{e+1}$ the new one. The illustration shows one single collector $\mathcal{K}'_{k'} \in \mathcal{K}'_{[n']}$. Each player \mathcal{P}_i of $\mathcal{P}_{[n]}$ has a list \mathbb{L}_i of vector of ciphertext shares received at the end of the previous Refresh. For each such $c_{[n]}$ in its list (**only one is depicted**), \mathcal{P}_i computes a publicly verifiable resharing of its coordinate c_i : $(c_{i \rightarrow [n]}, \pi_{\text{res},i})$ which it sends to the collectors (**only one is depicted**) appended with $c_{[n]}$ and a signature σ_i on it. Each collector waits until it collects a batch of such encrypted resharnings, out of a subset $\mathcal{U} \subset [n]$ of $t+1$ players, all verifiably generated out of the same $c_{[n]}$, such that all $t+1$ proofs $\pi_{\text{res},i}$ and signatures σ_i are valid. Then, it form $c'_{[n']}$ by Lagrange linear combination, and multicasts the batch $(c'_{[n]}, (c_{i \rightarrow [n]}, \pi_{\text{res},i})_{i \in \mathcal{U}}, (c_{[n]}, \{\sigma_i\}_{i \in \mathcal{U}}))$ to $\mathcal{P}'_{[n']}$. Every player $\mathcal{P}'_j \in \mathcal{P}'_{[n']}$, for every such batch which it receives (**only one is depicted**), verifies that *i*) all $t+1$ NIZK $\pi_{\text{res},i}$ and signatures σ_i are valid and *ii*) that $c'_{[n]}$ has been correctly formed. If it is the case, then it sets $\mathbb{L}'_j[k] \leftarrow c'_{[n]}$.

shares to \mathcal{L} . Now, assume that $\mathcal{F}_{\text{bPKI}}$ does not follow our specifications, and that it published as \perp the key of some honest player i which was too slow. Then the dealer has two options: either broadcast i 's share in clear, in which case the privacy threshold is downgraded to $t-1$. Or, do not broadcast it, in which case it is not true anymore that any $t+1$ honest players can reconstruct s , since i could be in this $(t+1)$ -set

B.4 When do committees start to exist ? For simplicity in the toy model Section 2 we assumed that all committees exist since the beginning of the execution. But actually, the toy model can be extended to allowing each player to be initialized at any time, or even never. The only modification to be made is in the functionality $\mathcal{F}_{\text{P-AVSS}}$. It is the following: $\mathcal{F}_{\text{P-AVSS}}$ refuses to advance $e_o \leftarrow e_o + 1$ until all players of $\mathcal{P}_{[n]}^{e_o+1}$ exist.

Then, in the model we would just have to give the power to the environment \mathcal{Z} to initialize players at any time, or even never. The UC proof holds unchanged, since the honest players of $\mathcal{P}_{[n]}$, both in the real and in the simulated execution, do not start Refresh ($\mathcal{P}_{[n]}, \mathcal{P}'_{[n']}$) until all keys of $\mathcal{P}'_{[n']}$ have been published, in particular, until all honest players of $\mathcal{P}'_{[n']}$ have been initialized. In conclusion, by universal composability, APSS0 can be composed with any external mechanism which would initialize players.

B.5 Why are resharnings not signed ? APSS0 is agnostic of “by whom” / “how” / “in which context” was produced the vectors of ciphertexts and the attached NIZK $\pi_{\text{res},i}$, relative to the resharing of the share of any player \mathcal{P}_i . For instance, our proof captures the following scenario. Consider two vector of ciphertext shares: $c_{[n]}$ and $\tilde{c}_{[n]}$, both in the list of some honest player \mathcal{P}_i , which happen to have the same coordinate $c_i = \tilde{c}_i$. Then \mathcal{P}_i generates distinct resharnings of both of them: $c_{i \rightarrow j}$ $\tilde{c}_{i \rightarrow j}$, which it sends to a corrupt collector $\mathcal{K}'_{k'}$. Then $\mathcal{K}'_{k'}$ forwards the resharing of the former: $c_{i \rightarrow j}$ in place of the re-sharing of the latter: $\tilde{c}_{i \rightarrow j}$.

Now, let us look at how this scenario is captured in the proof of correctness. Since \mathcal{P}_i is honest, the NIZK $\pi_{\text{res},i}$ attached to $c_{i \rightarrow j}$ is accepted by $\mathcal{F}_{\text{NIZK}}$. Then, by definition of $\mathcal{F}_{\text{NIZK}}$, it must be that $\mathcal{F}_{\text{NIZK}}$ was given as witness: the correct decryption s_i of $\tilde{c}_i = c_i$, and the plaintexts $s_{i \rightarrow j}$ of $c_{i \rightarrow j}$, such that they are

evaluations of some degree t polynomial f evaluating to s at 0. The same conclusion holds in the relaxation where $\mathcal{F}_{\text{NIZK}}$ is replaced by simulation-sound extractable (Sim-Ext) NIZK AoKs. Precisely, if an honest \mathcal{P}'_j receives a resharing: $(c_{i \rightarrow [n]}, \pi_{\text{res}, i})$ included in the batch received from some corrupt collector $\mathcal{K}'_{k'}$. Then, it must be the case that

- either \mathcal{A} received $\pi_{\text{res}, i}$ as such (from an honest player \mathcal{P}_i , or from the simulator on behalf of \mathcal{P}_i in the UC proof). This case is a particular case of the Sim-Ext game where \mathcal{A} queries proofs of arbitrary statements (even false ones) to its oracle;
- or \mathcal{A} can be extracted a witness.

Finally, let us look at how this scenario is captured by our simulator. In particular, in the case of both an honest \mathcal{D} and \mathcal{L} , existence of a simulator for APSS roughly implies *secrecy* of the secret. The simulator only modifies the fact that \mathcal{P}_i *does not use its secret decryption key* when it generates its re-sharings. The simulator does not modify how these re-sharings are used by collectors, nor how they are subsequently forwarded by public asynchronous channels. Indeed, what guarantees secrecy, roughly speaking, is only the fact that the simulated \mathcal{D} generates a PVSS of 0, instead of, of its secret.

B.6 Can players open outside of the window of opening ? By construction of **Open** in Section 4.3, a player in some $\mathcal{P}_{[n_e]}^e$ receiving **open-sig**, continues forever to send to \mathcal{L} a decryption of every new share arriving in its list (up to the maximum number κ_e , by construction), until it **shuts-off**. So these actions potentially spread beyond and after the timeframe which we denoted as window of opening. On the other hand, if a honest player of $\mathcal{P}_{[n_e]}^e$ receives **shutoff-sig** before it received any new share, then players of $\mathcal{P}_{[n_e]}^e$ may never be able to open, collectively, a secret. This can happen if $\Delta_{\text{wait}}^e \leq \delta$: Theorem 2 does not guarantee openability from $\mathcal{P}_{[n_e]}^e$ in this regime. In any case, this does not prevent that $\mathcal{P}_{[n_{e+1}]}^{e+1}$ will have again a chance to **Open**, from the finite time T^{e+1} , as precisely guaranteed in Theorem 2.

Let us give an intuition why if $\Delta_{\text{wait}}^e \leq \delta$ then $\mathcal{P}_{[n_e]}^e$ may never be able to **Open**. We refer to the description of the **refreshing squad** mechanism described in Section 6. Suppose for simplicity that $\Delta_{\text{wait}}^e = 0$. Consider a scenario in which there is a player of $\mathcal{P}_{[n_e]}^e$ which forms first a **qkc**, so outputs **refresh-sig**. Then it quickly engages into a **Refresh** with $\mathcal{P}_{[n_{e+1}]}^{e+1}$, assisted by the t corrupt players of $\mathcal{P}_{[n_e]}^e$. Messages in this **Refresh** are delivered much faster than δ . In the while, δ still did not elapse, and the other players of $\mathcal{P}_{[n_e]}^e$ did not receive yet any new share (technically: any *proven new sharing*) from the collectors $\mathcal{K}_{[\kappa_e]}^e$. So they are unable to send any opening shares to \mathcal{L} . Then (still before δ elapses) they all receive very quickly a **qrc** ^{$e+1$} so they all **shut-off**.

B.1 Optimizing for the good case, if not hurry in the worst case.

There exists an alternative approach to resharing, which is used in some PSS protocols [MZW+19; GKM+22; YXD22; YXXM23]. It consists in generating a fresh sharing of a random mask r , which is both shared within the exiting committee and the entering committee. Then, the exiting committee publicly opens the value $x \leftarrow r - s$ of the masked secret. Subtracting x to their shares of r , the entering committee obtains a fresh sharing of s . This approach requires one more round-trip of interaction than resharing, so what using it ? Its benefit is that the communication cost of generating shared randomness can be amortized with the following generic tool [BH08]. Consider any $n - t \times n$ matrix M , of which all minors are invertible. It is denoted as an hyperinvertible matrix. A Vandermonde matrix is hyperinvertible. //Such matrices are also known as MDS, see [CDN15, §8]. Then for any n -sized input vector X , of which any $n - t$ out of n coordinates vary uniformly at random, we have that the output $Y = M.X$ has its $n - t$ out of n coordinates which vary uniformly at random.

We now describe the variant of APSS0 using this randomness extraction. We consider the need to **Refresh** $W = n - t$ secrets in parallel. Broadly, the goal is for each collector is to obtain W *double encrypted sharings of random values*. Each is of the form: $(c_{[n]}^w, c_{[n]'}^w)$, such that both sides are vector of ciphertext shares of the same value $r^w \in \mathbb{S}$, and such that r^w varies uniformly conditioned on the view of \mathcal{A} . //This is to be

understood in a computational sense: upon being given either the plaintext r^w or a uniformly random one, a PPT adversary could not distinguish between them. More precisely, for each collector index $k \in [\kappa']$, each player of $\mathcal{P}_{[n]}$:

- waits until it accepts W double sharings;
- then, for each secret s^w , it sends to $\mathcal{P}'_{[n']}$ its decryption share of $x^w = \leftarrow r^w - s^w$.

Notice that a corrupt collector could manage so that two distinct players receive double sharings, for the same index w , which are sharings of distinct randomnesses: $r^w \neq u^w$. But in this case, the generation which we are going to describe shows that the adversary can be extracted the *difference* between the two conflicting randomnesses: $\delta_{r^w, u^w} := r^w - u^w$. So the decryption shares sent by honests players associated to index w are still simulatable.

We now describe the generation. For simplicity we consider a threshold $n = 2t + 1$. Of course the extraction would be far easier in the setting $t < n/3$, as in [YXD22; YXXM23].

- Each $\mathcal{P}_i \in \mathcal{P}_{[n]}$ samples a random value $r_i \in \mathbb{S}$ then generates a double encrypted sharing of it : once under the $[n]$ keys of $\mathcal{P}_{[n]}$ and once under the $[n']$ keys of $\mathcal{P}'_{[n']}$. It multicasts to the collectors the encrypted sharings, appended with NIZK's of r , plaintext shares and encryption randomnesses.
- Each collector $\mathcal{K}'_{[\kappa']}$, upon receiving $t+1$ double sharings, it sets a timeout. Then, it gathers the total $|\mathcal{U}| \geq t+1$ double sharings received: $((c_{i \rightarrow [n]}, c'_{i \rightarrow [n']})_{i \in \mathcal{U}}$.
- In the bad case where it received no more than $|\mathcal{U}| = t+1$ double sharings, then it can extract only one double sharing out of them, which is varies uniformly independently of the view of \mathcal{A} . Namely, by computing their sum. So it queries players of $\mathcal{P}_{[n]}$ to generate and send to it the required $M - 1$ more double sharings of random values.
- In the very good case where it received n resharing, it homomorphically applies M to them, to obtain $n - t$ encrypted sharings of uniformly random values.
- In intermediary case where it would have received $n - f > t+1$ double sharings, e.g., where $n - f$ players behaved honestly and their messages were delivered before the timeout, then it extracts $n - f - t$ random double sharings out of them. To this end, it applies a hyperinvertible matrix of smaller size $n - f - t \times n$. Then it queries players of $\mathcal{P}_{[n]}$ to generate and send more double sharings of random values, until it can extract a total number of M .
- Upon having extracted M uniform double sharings, it sends them to players of $\mathcal{P}_{[n]}$ and $\mathcal{P}'_{[n']}$, appended with all the double sharings received including the NIZKs appended to them. A player accepts them if the NIZKs pass verification.

B.2 Extending to various forms of adaptive security

We discuss nested models of adaptive corruptions. We start with the simplest model, where \mathcal{A} can corrupt a participant \mathcal{P} only before any ciphertext under \mathcal{P} 's encryption key, i.e., resharing to the committee of \mathcal{P} , was publicly sent by any other honest player. This includes adaptive corruption of \mathcal{D} at any time, since it does not have a public encryption key. Then, we extend to adaptive corruptions of players before they used their decryption keys, i.e., before they started to reshare. We then extend to adaptive corruptions of players at any time.

Recall that ([CDN15, p112]), upon adaptively corrupting a dummy entity P , the simulator \mathcal{S} learns its true input(s), and must show to \mathcal{Z} an internal state of P . To prevent \mathcal{Z} from distinguishing, the internal state shown must be compatible both with the true input of P , and the view of \mathcal{Z} during the simulated execution.

B.1 Corrupting players before any honest player sent a resharing encrypted under their key
 Dummy players have their only inputs equal to the signals (open-sig), (refresh-sig). They were timely relayed to \mathcal{S} by $\mathcal{F}_{\text{P-AVSS}}$, so \mathcal{S} can reveal these corresponding inputs and reception timings. Moreover, simulated

honest players followed the protocol so far, which boils down to: honestly sample a private key then publish the public key. So the simulator needs only revealing the secret key of every P adaptively corrupt.

However, the latter creates the following problem that our statement of IND-CPA of encrypted sharing, in Appendix C.1, requires that the challenger \mathcal{M} gives the t corrupt indices $\mathcal{I} \subset [n]$ *before* it can see the keys. Indeed, in the proof, this is why the reduction \mathcal{A}_{PKE} knows for which indices it samples the keys itself, and for which indices it queries the keys to its oracle. We observe that there exists the following generic reduction. The reduction \mathcal{A}_{PKE} simply samples at random in its head a subset \mathcal{I} of t indices, hoping that these are the ones of which the challenger \mathcal{M} will ask to see the secret decryption keys. Accordingly, \mathcal{A}_{PKE} samples the keys in \mathcal{I} itself, while it assigns to the indices in $[n] \setminus \mathcal{I}$ the keys that it received from its oracle. In the bad event where \mathcal{M} would ask to see other keys than the ones in \mathcal{I} , \mathcal{A}_{PKE} cannot satisfy this request, so it simply outputs a bit at random to its oracle. In the good event where \mathcal{M} asks to see only keys in \mathcal{I} , then \mathcal{A}_{PKE} proceeds as in the proof of Appendix C.1. The good event happens with probability $1/\binom{n}{t}$, so this loss is affordable for the values of n (up to 15) in most concrete use-cases.

For large values of n , one may switch to encryption schemes which are by nature are resilient to these late adaptive requests of *selective opening* of keys. They are known as RIND-SO, and were proven practical in [HPW15; YLH+20]. Since they are built from (a tweaked variant of) non-committing encryption, our requirement of bilateral binding, in Definition 3, does not hold anymore. So encrypted subshares must be appended with public commitments to them, as in [Gro21] and Section 7.3.

B.2 Extending to corruptions of players before they use their decryption keys We are now adding scenarios of adaptive corruption of a player \mathcal{P} after (re)sharings under \mathcal{P} 's key were publicized. Our reduction in $1/\binom{n}{t}$ loss still applies, the loss does even not depend on the number of encrypted sharing requests made by the challenger \mathcal{M} (contrary to [YLH+20]).

On the other hand, it seems to us that applying RIND-SO schemes does not work anymore, contrary to what is claimed in [BGG+20] (p21, Hybrid 4). Indeed, RIND-SO schemes guarantee only indistinguishability of (non revealed) plaintexts following the *same* distribution. But in our case, consider the simplest scenario of the publicly verifiable secret sharing (PVSS) from an honest dealer \mathcal{D} under the keys of $\mathcal{P}_{[n_1]}^1$. Since the simulator \mathcal{S} does not know the secret s of \mathcal{D} , it is hopeless for \mathcal{S} to generate a PVSS of which the plaintexts follow the same distribution as a PVSS of s . Indeed, in the real execution, the plaintext shares lie on a degree t polynomial evaluating to s at 0. So if \mathcal{S} were able to generate such plaintexts with overwhelming probability (w.o.p.), then this would mean that \mathcal{S} can guess s w.o.p. But it can guess s only with probability $1/|\mathcal{S}|$, a contradiction.

B.3 Corrupting players at any time We are now adding scenarios of adaptive corruption of a player \mathcal{P}_i after it decrypted-then-resshared its new share. This actually does not change anything to the simulator and proof. One could fear that giving to \mathcal{A} the key of \mathcal{P}_i , may impact the proof in Appendix C.8, of indistinguishability between the two hybrids $\text{Hyb}^{\text{Refresh}}[e, i-1]$ and $\text{Hyb}^{\text{Refresh}}[e, i]$. Recall that, in the former, \mathcal{P}_i does an actual decryption-then-encrypted-ressharing of its encrypted shares c_i , whereas in the latter, it generates a bogus vector of ciphertexts. But in the proof we made a reduction into a challenger \mathcal{M} big enough to be extracted the *plaintext* s_i of c_i . So, the view of the adversary in both hybrids is independent from the secret key of \mathcal{P}_i . Indeed, the only impact of the secret key in the view is to generate the decryption s_i of c_i , which the challenger *knows already*.

C Complements on the proof of APSS0

C.1 IND-CPA of encrypted sharing

Proposition 8 states that any PPT machine \mathcal{M} corrupting at most t key-holders, has negligible advantage in distinguishing between the encrypted t -out-of- n Shamir sharings of any two chosen secrets $(s_L, s_R) \in \mathbb{S}^2$.

Proposition 8 (IND-CPA of encrypted sharing). *For any integers $0 \leq t < n$, we consider the following game between a machine \mathcal{M} and an oracle \mathcal{O} . \mathcal{O} is parametrized by a secret $b \in \{L, R\}$ (“left or right oracle”).*

Setup. \mathcal{M} gives to \mathcal{O} : a subset of t indices $\mathcal{I} \subset [n]$, and a list of t public keys $(\mathbf{ek}_i)_{i \in \mathcal{I}} \in (\mathcal{e}\mathcal{K} \sqcup \perp)^t$. For each $i \in [n] \setminus \mathcal{I}$, \mathcal{O} generates $\mathbf{ek}_i \leftarrow \text{EKGen}(\text{dk}_i \xleftarrow{\$} \mathcal{d}\mathcal{K})$ and shows \mathbf{ek}_i to \mathcal{M} . \mathcal{M} is allowed to query \mathcal{O} an unlimited number of times as follows.

Query. \mathcal{M} gives to \mathcal{O} a pair $(s_L, s_R) \in \mathbb{S}^2$. Depending on $b \in \{L, R\}$, \mathcal{O} replies as either \mathcal{O}^L or \mathcal{O}^R :

- \mathcal{O}^L : samples a degree t polynomial h at random such that $h(0) = s_L$, sets $s_i := h(i)$, $\forall i \in [n]$, returns $(\text{Enc}_{\mathbf{ek}_i}(s_i))_{i \in [n]}$;
- \mathcal{O}^R : samples a degree t polynomial h at random such that $h(0) = s_R$, sets $s_i := h(i)$, $\forall i \in [n]$, returns $(\text{Enc}_{\mathbf{ek}_i}(s_i))_{i \in [n]}$.

At some point \mathcal{M} may output a string, e.g., a bit. Then for any PPT machine \mathcal{M} , the distinguishing advantage $\text{Adv}_{LR} = |\mathbb{P}(1 \leftarrow \mathcal{M}^{\mathcal{O}^L}) - \mathbb{P}(1 \leftarrow \mathcal{M}^{\mathcal{O}^R})|$ is negligible.

Proof. First, we consider the straightforward reduction into the slightly easier variant of the game, which we denote as [IND-CPA of encrypted sharing with plaintext adversary shares]. There, the challenger \mathcal{M} is given shares of corrupt players in the clear. Notice that this variant is strictly easier when \mathcal{M} badly generated some of its keys, and thus is unable to decrypt the shares of corrupt players. //Formally, consider a challenger \mathcal{M}_{enc} of Proposition 8. The reduction forwards to its oracle the requests of \mathcal{M}_{enc} . It receives the plaintext shares $(s_i)_{i \in \mathcal{I}}$ and the ciphertexts $(c_i)_{i \in [n] \setminus \mathcal{I}}$. It forwards to \mathcal{M}_{enc} : $(\text{Enc}_{\mathbf{ek}_i}(s_i))_{i \in \mathcal{I}}$ and the same ciphertexts $(c_i)_{i \in [n] \setminus \mathcal{I}}$.

We now consider the game of [IND-CPA of encrypted sharing with plaintext adversary shares]. We denote again its oracles as \mathcal{O}^L and \mathcal{O}^R , although now they return in the clear the t corrupt shares. We first define two apparent modifications of \mathcal{O}^L and \mathcal{O}^R , denoted as $\tilde{\mathcal{O}}^L$ and $\tilde{\mathcal{O}}^R$, which only differ from the previous, in that they *first* sample the corrupt shares $(s_i)_{i \in \mathcal{I}} \xleftarrow{\$} \mathbb{S}^t$ uniformly at random, *then* interpolate h through them and s_L or s_R . Actually, by the secrecy of Shamir secret sharing recalled in Section 2.2, they produce exactly the same distribution as \mathcal{O}^L and \mathcal{O}^R . We formalize them below, then formalize the previous claim in Equation (7).

- $\tilde{\mathcal{O}}^L$: samples $(s_i)_{i \in \mathcal{I}} \xleftarrow{\$} \mathbb{S}^t$; interpolate the degree t polynomial h through $(s_L, (s_i)_{i \in \mathcal{I}})$; sets $s_i := h(i)$ $\forall i \in [n] \setminus \mathcal{I}$; returns $((s_i)_{i \in \mathcal{I}}, (\text{Enc}_{\mathbf{ek}_i}(s_i))_{i \in [n] \setminus \mathcal{I}})$;
- $\tilde{\mathcal{O}}^R$: samples $(s_i)_{i \in \mathcal{I}} \xleftarrow{\$} \mathbb{S}^t$; interpolate the degree t polynomial h through $(s_R, (s_i)_{i \in \mathcal{I}})$; sets $s_i := h(i)$ $\forall i \in [n] \setminus \mathcal{I}$; returns $((s_i)_{i \in \mathcal{I}}, (\text{Enc}_{\mathbf{ek}_i}(s_i))_{i \in [n] \setminus \mathcal{I}})$.

For any possibly unlimited machine \mathcal{M} ,

$$(7) \quad |\mathbb{P}(1 \leftarrow \mathcal{M}^{\tilde{\mathcal{O}}^L}) - \mathbb{P}(1 \leftarrow \mathcal{M}^{\mathcal{O}^L})| = 0 \text{ and } |\mathbb{P}(1 \leftarrow \mathcal{M}^{\tilde{\mathcal{O}}^R}) - \mathbb{P}(1 \leftarrow \mathcal{M}^{\mathcal{O}^R})| = 0$$

To conclude the proof, we introduce an intermediary oracle, defined as $\tilde{\mathcal{O}}^Z$ below, and prove that the distinguishing advantage between both $\tilde{\mathcal{O}}^L$ and $\tilde{\mathcal{O}}^R$, with $\tilde{\mathcal{O}}^Z$, is negligible. $\tilde{\mathcal{O}}^Z$ is the common modification of $\tilde{\mathcal{O}}^L$ and $\tilde{\mathcal{O}}^R$, which sets to 0 the $n - t$ honest plaintext shares. In particular, it completely ignores the request (s_L, s_R) given to it.

- $\tilde{\mathcal{O}}^Z$: samples $(s_i)_{i \in \mathcal{I}} \xleftarrow{\$} \mathbb{S}^t$; sets $s_i := 0 \forall i \in [n] \setminus \mathcal{I}$;
- returns $((s_i)_{i \in \mathcal{I}}, (\text{Enc}_{\mathbf{ek}_i}(s_i))_{i \in [n] \setminus \mathcal{I}})$;

Claim: the maximum distinguishing advantage with $\tilde{\mathcal{O}}^Z$ is less than the one for $n - t$ -keys IND-CPA for PKE. We recall the game defining it, from which the *Claim* should be clear enough. It is between a challenger \mathcal{A}_{PKE} , and an oracle \mathcal{O}_{PKE} parametrized by a secret $b \in \{E, 0\}$.

- \mathcal{O}_{PKE} samples $(n - t)$ PKE public keys $(\mathbf{ek}_h)_{h \in [n - t]}$ which it gives to \mathcal{A}_{PKE} ;

- Upon receiving, from \mathcal{A}_{PKE} , $(n - t)$ chosen plaintexts $(s_h)_{h \in [n - t]}$, then \mathcal{O}_{PKE} replies depending on $b \in \{E, 0\}$.

In the former case it behaves as $\mathcal{O}_{\text{PKE}}^E$ (actual $n - t$ -keys encryption), in the latter case as $\mathcal{O}_{\text{PKE}}^0$ (encryptions of 0).

- $\mathcal{O}_{\text{PKE}}^E$ returns $(\text{Enc}_{\mathbf{ek}_h}(s_h))_{h \in [n - t]}$;
- $\mathcal{O}_{\text{PKE}}^0$ returns $(\text{Enc}_{\mathbf{ek}_h}(0))_{h \in [n - t]}$.

Recall that the distinguishing advantage in this $t+1$ -keys IND-CPA game, is upper-bounded by $n-t$ times the advantage for one-message indistinguishability, see e.g. [BS20, Thm 5.1].

We now fully formalize the proof of the *Claim*, as the following straightforward reduction from the game $(\widetilde{\mathcal{O}}^L/\widetilde{\mathcal{O}}^Z)$ (and likewise $(\widetilde{\mathcal{O}}^Z/\widetilde{\mathcal{O}}^R)$) into the $n-t$ -keys IND-CPA game $(\mathcal{O}_{\text{PKE}}^E/\mathcal{O}_{\text{PKE}}^0)$. We (still) denote as \mathcal{M} a challenger for the former game. The reduction sets to \mathcal{M} the honest encryption keys: $(\mathbf{ek}_h)_{h \in [n-t]}$ as those received from its oracle \mathcal{O}_{PKE} . Upon receiving from \mathcal{M} the request (s_L, s_R) (we keep the same syntax, but actually s_R is never used here), the reduction samples $(s_i)_{i \in \mathcal{I}} \xleftarrow{\$} \mathbb{S}^t$; interpolates the degree t polynomial h through $(s_R, (s_i)_{i \in \mathcal{I}})$; sets $s_i := h(i) \forall i \in [n] \setminus \mathcal{I}$ which it gives to its oracle \mathcal{O}_{PKE} as a request. Upon being answered the challenge ciphertexts $(c_i)_{i \in [n] \setminus \mathcal{I}}$, it replies to \mathcal{M} : $((s_i)_{i \in \mathcal{I}}, (c_i)_{i \in [n] \setminus \mathcal{I}})$. The *Claim* now follows from the fact that in the case $\mathcal{O}_{\text{PKE}}^E$, then \mathcal{M} is facing the same behavior as $\widetilde{\mathcal{O}}^L$, while in the case $\mathcal{O}_{\text{PKE}}^0$, then \mathcal{M} is facing the same behavior as $\widetilde{\mathcal{O}}^Z$. \square

C.2 Simulator with an honest dealer \mathcal{D} , and proof of indistinguishability

We first describe, in Algorithm 13, the simulator for an honest dealer \mathcal{D} and a corrupt learner \mathcal{L} . Then we give the full details of the hybrids outlined in Section 5.3. In addition, we describe inline the modifications to be made to the hybrids for an honest \mathcal{L} .

We start by considering a real execution $\text{REAL}_{\mathcal{A}}$ of APSS0, with adversary \mathcal{A} fully controlled by the environment \mathcal{Z} . \mathcal{Z} assigns its input to \mathcal{D} and listens to the output (if any) of \mathcal{L} . Then we go through a series of hybrid games, which we show indistinguishable from one with the next, from the point of view of \mathcal{Z} . In the final game, denoted as $\text{Hyb}^{\text{OShare}}$, the view of \mathcal{A} is generated without any direct interaction with the honest players. The only indirect interaction with honest players happens in the opening, via $\mathcal{F}_{\text{P-AVSS}}$, which delivers the actual value of s , which helps us to simulate the opening shares of s . So what we are describing in $\text{Hyb}^{\text{OShare}}$ is a simulator \mathcal{S} , as described in Algorithm 13, which interacts only with $\mathcal{F}_{\text{P-AVSS}}$ and \mathcal{Z} , which concludes the UC proof.

The purpose of the games $\text{Hyb}^{\text{ShSim}}$ and $\text{Hyb}^{\text{sOpen}}$, which are not needed if \mathcal{L} is honest, is to make so that the view of \mathcal{Z} is generated without using the private keys of honest players which **Open** nor using the plaintext shares of honest players which **Open**. This allows to apply IND-CPA of PKE in subsequent games. In particular in $\text{Hyb}^{\text{ORefresh}}[1, n]$ we achieve that all re-sharings are actually mere PVSS of 0.

C.1 Game $\text{REAL}_{\mathcal{A}}$ This is the actual execution of the protocol APSS0 with environment \mathcal{Z} , adversary \mathcal{A} and ideal functionalities $\text{RB}^{\mathcal{D} \rightarrow \mathcal{P}_{[n]}^1}$, $\mathcal{F}_{\text{bPKI}}$, $\mathcal{F}_{\text{NIZK}}$, \mathcal{F}_{AT} .

We make the change (not formalized by a game) that $\mathcal{F}_{\text{NIZK}}$ does not check validity of witnesses (if any) received from honest players nor from \mathcal{D} . This does not change its outputs, since honest participants always provide correct witnesses when querying $\mathcal{F}_{\text{NIZK}}$ in the actual protocol.

We also make formal changes. We initiate **dummy players** which receive the **refresh-sig**'s and **open-sig**'s from \mathcal{Z} , in place of the players. We initiate a $\mathcal{F}_{\text{P-AVSS}}$ to which they send **refresh-req**'s and **open-req**'s. Finally, $\mathcal{F}_{\text{P-AVSS}}$ notifies to players when it receives **refresh-req**'s and **open-req**'s. Upon receiving these notifications, players act as if receiving **refresh-sig** and **open-sig**.

C.3 Game $\text{Hyb}^{\text{schedule}}$ (only for \mathcal{L} honest.)

We initiate a **dummy \mathcal{L}** . Now, the output notified to \mathcal{Z} is the one of this **dummy \mathcal{L}** . We make the **dummy \mathcal{L}** output as follows. We upgrade $\mathcal{F}_{\text{P-AVSS}}$ to the full functionality specified in Fig. 5. From now on \mathcal{D} , in addition to playing the protocol, gives its actual input to $\mathcal{F}_{\text{P-AVSS}}$. We influence the internal counter e_o of $\mathcal{F}_{\text{P-AVSS}}$ as follows. When a quorum of $t_{e_o} + 1$ **dummy honest players** requested **refresh-req**, we delay the increment $\mathcal{P}_{[n_{e_o}]}^{e_o} \leftarrow \mathcal{P}_{[n_{e_o}]}^{e_o} + 1$ (by sending **delay** $_{e_o}$ to $\mathcal{F}_{\text{P-AVSS}}$) until the moment when all honest players of $\mathcal{P}_{[n]}^{e_o+1}$ receive at least one proven new sharing in common. We also influence the time of the output delivery as follows. When a quorum of $t_{e_o} + 1$ **dummy honest players** requested **open-req**, we delay the output to the **dummy \mathcal{L}** (by sending **delay** to $\mathcal{F}_{\text{P-AVSS}}$) until the moment when \mathcal{L} outputs in the protocol.

Setup. \mathcal{S} initiates in its head simulated \mathcal{D} , \mathcal{L} and committees $\mathcal{P}_{[n_e]}^e$ of players, $\forall e \geq 1$. It initially receives corruption requests from \mathcal{Z} for t_e players per committee $\mathcal{P}_{[n_e]}^e$. Let \mathcal{I}^e and \mathcal{H}^e denote the set of indices of f_e corrupt players and of the $n_e - f_e$ honest players in each $\mathcal{P}_{[n_e]}^e$.

General behavior towards \mathcal{Z} . \mathcal{S} simulates functionalities $(\text{RB}^{\mathcal{D} \rightarrow \mathcal{P}_{[n_1]}^1}, \mathcal{F}_{\text{bPKI}}, \mathcal{F}_{\text{NIZK}}, \mathcal{F}_{\text{AT}}, \mathcal{F}_{\text{ST}})$, which all follow a correct behavior. The only exception is $\mathcal{F}_{\text{NIZK}}$, which does not check validity of witnesses (if any) received from simulated honest players nor from \mathcal{D} . \mathcal{S} behaves towards \mathcal{Z} as the dummy \mathcal{A} . Specifically, it relays to \mathcal{Z} every incoming message from simulated corrupt players and simulated functionalities. Upon receiving an instruction from \mathcal{Z} to send some message to some functionality, or to send some instruction to some corrupt player, \mathcal{S} internally sends the message or the instruction to the simulated functionality or the corrupt player. Simulated corrupt players behave as instructed. Simulated honest players have a different behavior, described below.

General behavior towards $\mathcal{F}_{\text{P-AVSS}}$. \mathcal{S} also interacts with $\mathcal{F}_{\text{P-AVSS}}$ on behalf of \mathcal{A} (without notifying it to \mathcal{Z}). Following its specification, $\mathcal{F}_{\text{P-AVSS}}$ relays to \mathcal{S} every message from the actual dummy honest players ((refresh-req), (open-req) and notifications of (shut-off)). \mathcal{S} has an exact control over the delays of *early* and of *collective refreshing*, which is enabled by the instructions *refresh-order* and *delay_e* which it sends to $\mathcal{F}_{\text{P-AVSS}}$. This enables \mathcal{S} to keep track in real time of the internal counters e_o and $e_{\mathcal{A}}$ of $\mathcal{F}_{\text{P-AVSS}}$.

Share The simulated honest dealer \mathcal{D} is initialized with input $\tilde{s} := 0$, it follows the protocol. \mathcal{S} triggers the notification of (committed) to dummy honest players via $\mathcal{F}_{\text{P-AVSS}}$ once their simulated counterparts have received the $\text{RB}^{\mathcal{D} \rightarrow \mathcal{P}_{[n_1]}^1}$.

Refresh Upon being notified that a dummy honest $\mathcal{P}_i^e \in \mathcal{P}_{[n_e]}^e$ sent (refresh-req) to $\mathcal{F}_{\text{P-AVSS}}$, then: for every $c_{[n_e]}$ in the list \mathbb{L}_i^e of a simulated honest \mathcal{P}_i^e , it follows the protocol except for step (Resharing).1 of Algorithm 2, where its sets $s_i \leftarrow 0$. In addition, upon being informed that a quorum of $t_{e_o} + 1$ dummy honest

players requested *refresh-req*, \mathcal{S} delays the increment $\mathcal{P}_{[n_{e_o}]}^{e_o} \leftarrow \mathcal{P}_{[n_{e_o}]}^{e_o} + 1$ (via *delay_{e_o}*) until the moment when all simulated honest players of $\mathcal{P}_{[n]}^{e_o+1}$ receive at least one proven new sharing in common.

Open Upon being leaked by $\mathcal{F}_{\text{P-AVSS}}$ that a dummy honest player \mathcal{P}_i^e , in some $\mathcal{P}_{[n_e]}^e$, sent *open-req* //which means that \mathcal{P}_i^e received *open-sig*, then do the following. If the simulated \mathcal{P}_i^e was already assigned simulated decryption shares for all the vectors of ciphertext shares in its local list \mathbb{L}_i^e , then, the simulated \mathcal{P}_i^e sends all these shares to \mathcal{L} along with fake proofs of correct decryption, i.e., without giving a witness to $\mathcal{F}_{\text{NIZK}}$. Otherwise, do the following steps to assign simulated decryption shares to \mathcal{P}_i^e , then proceed as above.

1. If $\mathcal{F}_{\text{P-AVSS}}$ did not leak the stored \tilde{s} yet //this happens if $< t_e + 1 - f_e$ dummy honest players requested *open-req* so far, $\forall e$. So this does not happen in the flagship setting where $t_e = f_e$. Then for every vector of ciphertext shares in the local list \mathbb{L}_i^e of \mathcal{P}_i^e , assign an arbitrary decryption share to \mathcal{P}_i^e .
2. Else this means that \mathcal{S} was just leaked the stored secret \tilde{s} from $\mathcal{F}_{\text{P-AVSS}}$. Then, do the following assignments from now on. For every vector of ciphertext shares $c_{[n_e]}$ accepted (now or in the future) in the local list \mathbb{L}_i^e of any simulated player \mathcal{P}_i^e , in any committee $\mathcal{P}_{[n_e]}^e$, do the following. Denote as $(c_{[n_{e-1}]}, c_{i \rightarrow [n_e]}, \pi_{\text{res}, i})_{i \in \mathcal{U}}$ the set, indexed by \mathcal{U} , of the $t_{e-1} + 1$ encrypted resharings out of which $c_{[n_e]}$ was formed by homomorphic Lagrange combination //recall that each \mathcal{P}_i^e checks in Item 1 (c') that this homomorphic linear combination is correctly computed.
 - (α) Extract from $\mathcal{F}_{\text{NIZK}}$ (in straight-line) the plaintext coordinates $(s_{i \rightarrow j})_{j \in \mathcal{I}^e}$ of corrupt players from the NIZK proofs of resharing, and sets $s'_j := \sum_{i \in \mathcal{U}} \lambda_i^{\mathcal{U}} s_{i \rightarrow j}$, $\forall j \in \mathcal{I}^e$. Under honest majority, \mathcal{S} internally *decrypts* the $t_e + 1$ opening shares $(s'_j)_{j \in [n_e] \setminus \mathcal{I}^e}$ of $c_{[n_e]}$, then *interpolates* the $(s_{i \rightarrow j})_{j \in \mathcal{I}^e}$ from them.
 - (β) Interpolate a degree t_e polynomial \hat{h} , such that $\hat{h}(0) = s$ and $\hat{h}(j) = s'_j \forall j \in \mathcal{I}^e$.
 - (γ) Assign the simulated decryption share $\hat{h}(i)$ to each honest \mathcal{P}_i^e having $c_{[n_e]}$ in its local list.

Algorithm 13: Description of \mathcal{S} for APSS0, for an honest \mathcal{D} and a corrupt \mathcal{L} and $f_e = t_e$ corruptions in each committee $\mathcal{P}_{[n_e]}^e$

Claim. $\text{REAL}_{\mathcal{A}} \equiv \text{Hyb}^{\text{schedule}}$. Moreover, only a polynomial number of requests to $\mathcal{F}_{\text{P-AVSS}}$ (*delay_e* and *delay*) need to be sent.

Proof. The two variables which potentially changed in the view of \mathcal{Z} , compared to $\text{REAL}_{\mathcal{A}}$, are the value of the output of \mathcal{L} and the time when it outputs. The value is unchanged, by Proposition 7. The timing is unchanged, since we delayed it until the moment when \mathcal{L} outputs in the protocol. In conclusion, the view is unchanged. The last claim follows from Lemma 4 and Proposition 5, which state that a Refresh or an Open from $\mathcal{P}_{[n_{e_o}]}^{e_o}$ always complete in polynomial time. \square

C.1 Game $\text{Hyb}^{\text{ShSim}}$ (skipped if \mathcal{L} honest.) For any $\mathcal{P}_{[n_e]}^e$ in which honest players receive open-sig, then for each $c_{[n_e]}$ opened by $\mathcal{P}_{[n_e]}^e$ to \mathcal{L} , the opening shares of honest players in $\mathcal{P}_{[n_e]}^e$ are now computed by Lagrange interpolation from $(s_{c_{[n_e]}}, (s_j)_{j \in \mathcal{I}^e})$, where:

- $s_{c_{[n_e]}}$ is the secret shared by the vector of ciphertext shares $c_{[n_e]}$ //concretely: decrypt the $n_e - t_e$ shares of $c_{[n_e]}$ of honest players. If $t_e < n_e$, then this is enough to reconstruct $s_{c_{[n_e]}}$, else, use in addition the shares of corrupt players defined below;
- $(s_j)_{j \in \mathcal{I}^e}$ are the opening shares of $c_{[n_e]}$ of corrupt players, computed as follows. Each $(c_{i \rightarrow [n]}, \pi_{\text{res}, i}), \forall i \in \mathcal{U}$, must be of the form $c_{i \rightarrow [n_e]} = (\text{Enc}_{\text{ek}_{j'}}(s_{i \rightarrow j}; \rho_j))_{j \in [n_e]}$, where $(s_{i \rightarrow j})_{j \in [n_e]}$ is a vector of shares of s_i . For an honest player this is automatic since it follows the protocol. For a corrupt player this is guaranteed by the NIZK $\pi_{\text{res}, i}$ //More precisely, we extract $s_{i \rightarrow j}$ by rewinding the environment. We can do so because what we are describing is an intermediate distribution, not a simulator. We finally set $s_j := \sum_{i \in \mathcal{U}} \lambda_i^{\mathcal{U}} s_{i \rightarrow j}, \forall j \in \mathcal{I}^e$

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

Proof. For any given vector of (n_e, t_e) shares of some secret $s_{c_{[n_e]}}$, the sharing polynomial is fully determined by $\{s_{c_{[n_e]}}$ and any t_e shares $\}$. In turn, the sharing polynomial fully determines the remaining $n_e - t_e$ shares. So the $n_e - t_e$ shares output by the simulation are the same as the actual $n_e - t_e$ shares. So the view is unchanged compared to the real execution. \square

C.4 Game $\text{Hyb}^{\text{sOpen}}$ (skipped if \mathcal{L} honest.)

This game differs from $\text{Hyb}^{\text{ShSim}}$ in that, in the interpolation of the $n_e - t_e$ honest decryption shares, we replace the input $s_{c_{[n_e]}}$ by the actual secret s of the dealer \mathcal{D} leaked by $\mathcal{F}_{\text{P-AVSS}}$.

Claim. $\text{Hyb}^{\text{ShSim}} \equiv \text{Hyb}^{\text{sOpen}}$.

Proof. By Proposition 7, for each $(c_{[n_e]})$ relatively to committee $\mathcal{P}_{[n_e]}^e$, we have that $c_{[n_e]}$ is a vector of ciphertext shares with threshold opening equal to s . So the output of the interpolation is identical, hence the view is unchanged. \square

From this point, neither the secret keys of honest players in $\mathcal{P}_{[n_e]}^e$, nor their plaintext shares, are used anymore to generate the view of \mathcal{Z} .

C.5 Games $\text{Hyb}^{\text{0Refresh}}[e, i]$ for each $e \in [e_{\mathcal{A}} - 1, \dots, 1]$ (in this backwards order) then each $i \in [0, \dots, n_e]$

We set $\text{Hyb}^{\text{0Refresh}}[e_{\mathcal{A}} - 1, 0] := \text{Hyb}^{\text{ShSim}}$. Then for each $e \in [e_{\mathcal{A}} - 1, \dots, 1]$ and $i \in [0, \dots, n_e - 1]$: if $P_{i+1}^{(e)}$ is corrupt then we leave $\text{Hyb}^{\text{0Refresh}}[e, i+1] := \text{Hyb}^{\text{0Refresh}}[e, i]$ unchanged, otherwise if it is honest, then we modify $\text{Hyb}^{\text{0Refresh}}[e, i]$ into $\text{Hyb}^{\text{0Refresh}}[e, i+1]$ as follows. For each $(c_{[n_e]}^e = (c_i^{(e)})_{i \in [n_e]})$ in the local list of $P_{i+1}^{(e)}$, we substitute s_i^e by 0. In particular, *the secret decryption key of $P_{i+1}^{(e)}$ 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 players. When reaching $i = n_e$, if $e \geq 2$, then we set $\text{Hyb}^{\text{0Refresh}}[e-1, 0] := \text{Hyb}^{\text{0Refresh}}[e, n_e]$.

Claim. $\text{Hyb}^{\text{sOpen}} \equiv \text{Hyb}^{\text{0Refresh}}[1, n_1]$

Proof. It is enough to show that for each $e \in [e_{\mathcal{A}} - 1, \dots, 1]$, for $i \leq n_e - 1$ such that $P_{i+1}^{(e)}$ is honest, then $\text{Hyb}^{\text{Refresh}}[e, i]$ is indistinguishable from $\text{Hyb}^{\text{Refresh}}[e, i + 1]$. Let us consider one of the vector of ciphertext shares: $c_{[n_e]}^e = (c_i^{(e)})_{i \in [n_e]}$ in the list of $P_{i+1}^{(e)}$. The high level idea simply consists in showing indistinguishability between the two re-sharings $c_{i \rightarrow [n_{e+1}]}$ of $c_i^{(e)}$: the actual one, in $\text{Hyb}^{\text{Refresh}}[e, i]$, and the bogus one, in $\text{Hyb}^{\text{Refresh}}[e, i + 1]$. Let us make the simplifying *Assumption* that PKE perfectly hides the plaintext of the coordinates of $c_{i \rightarrow [n_{e+1}]}$ encrypted under the keys of honest players. Then, the view of \mathcal{Z} is fully determined by the plaintext coordinates $(s_{i \rightarrow j})_{j \in [n_{e+1}]}$, where $\mathcal{I}^{e+1} \subset [n_{e+1}]$ is the subset of indices of corrupt players. But by t -privacy of Shamir sharing, they also vary uniformly at random also in the actual $c_{i \rightarrow [n_{e+1}]}$. So the two distributions of views are equal. It remains to substantiate this *Assumption*, which will conclude this sketch proof. The reason is that we have that the view of \mathcal{Z} is generated without using (i) the secret decryption keys of players in $\mathcal{P}_{[n_{e+1}]}^{e+1}$, (ii) nor the plaintext shares of $c_{i \rightarrow [n_{e+1}]}$ with indices of the honest players in $\mathcal{P}_{[n_{e+1}]}^{e+1}$. Indeed, if $e = e_{\mathcal{A}} - 1$ then (i) and (ii) are thanks to $\text{Hyb}^{\text{sOpen}}$, while if $e < e_{\mathcal{A}} - 1$ then (i) and (ii) are thanks to $\text{Hyb}^{\text{Refresh}}[e + 1, n_e]$. So by (i) and (ii) we are in the conditions of applicability of *IND-CPA of encrypted sharing*, which is proven in Appendix C.1.

Actually, the “we are in the conditions of” is not completely straightforward, so we fully formalize the reduction in Appendix C.8. The subtlety is that the input of the re-sharing is a ciphertext c_i , not a plaintext. So in order to escape issues with IND-CCA, we consider a challenger of the reduction which is large enough, i.e., which concatenates enough entities, so that a plaintext of c_i can be extracted from it. \square

C.6 Game $\text{Hyb}^{\text{Share}}$

We modify $\text{Hyb}^{\text{Refresh}}[1, n_1]$ in that the dealer \mathcal{D} plays the protocol as if it had input 0, even though it still sends its actual input s to $\mathcal{F}_{\text{P-AVSS}}$.

Claim. $\text{Hyb}^{\text{Refresh}} \equiv \text{Hyb}^{\text{Share}}$

Proof. Since $\text{Hyb}^{\text{Refresh}}[1, n_1]$, to generate the view of \mathcal{Z} , we neither use the private decryption keys of honest players of $\mathcal{P}_{[n_1]}^1$, nor the plaintext shares of the PVSS of the dealer. Thus we can apply IND-CPA of encrypted sharing (Proposition 8) to the PVSS of \mathcal{D} , which is encrypted under the public keys of $\mathcal{P}_{[n_1]}^1$. \square

The following game $\text{Hyb}_{\text{Open}}^{\text{ShInfer}}$ is only used in the case of honest majority to avoid extraction from NIZK proofs.

C.7 Game $\text{Hyb}_{\text{Open}}^{\text{ShInfer}}$

If \mathcal{L} is honest, this game is identical to $\text{Hyb}^{\text{Share}}$. Else (is \mathcal{L} is corrupt): for each vector of ciphertext shares: $c_{[n_e]}$ held by at least one honest player in $\mathcal{P}_{[n_e]}^e$, we now change the method to infer the opening shares of $c_{[n_e]}$ of corrupt players. Recall that the purpose of these opening shares is to be fed as input in the interpolation of the simulated opening shares of honest players. The current method to compute them so far is the method in $\text{Hyb}^{\text{ShSim}}$: for each $(c_{i \rightarrow [n_e]}, \pi_{\text{res}, i})_{i \in \mathcal{U}}$ (if any) generated by a corrupt player P_i in $\mathcal{P}_{[n]}^{(e-1)}$, the plaintext coordinates $(s_{i \rightarrow j})_{j \in \mathcal{I}^e}$ of corrupt players were extracted from the NIZK AoK of resharing, before setting $s_j := \sum_{i \in \mathcal{U}} \lambda_i^{\mathcal{U}} s_{i \rightarrow j}$, $\forall j \in \mathcal{I}^e$. Instead, we make the change that the subshares of corrupt players, in each $c_{i \rightarrow [n_e]}$ reshared by a corrupt player, are computed as follows. We simply use the secret keys of the $t_e + 1$ honest players in $\mathcal{P}_{[n_e]}^e$ to correctly decrypt their opening shares $(s_{i \rightarrow j})_{j \in [n_e] \setminus \mathcal{I}^e}$ of $c_{i \rightarrow [n_e]}$. Then we interpolate the desired t_e shares $(s_{i \rightarrow j})_{j \in \mathcal{I}^e}$ of corrupt players, of $c_{i \rightarrow [n_e]}$.

Claim: the $(s_{i \rightarrow j})_{j \in \mathcal{I}^e}$ obtained are unchanged. Indeed, $c_{i \rightarrow [n_e]}$ is a vector of ciphertext shares, so the t_e plaintext coordinates, which were extracted from the NIZKs, lie on a polynomial of degree t_e which is fully determined by any $t_e + 1$ other coordinates.

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

Proof. Since $c_{i \rightarrow [n_e]}$ is a vector of ciphertext shares, Definition 3 guarantees that each coordinate decrypts to $(s_{i \rightarrow j} \bmod p)_{j \in [n_e]}$. Moreover, polynomial interpolation over any decryption shares of $t_e + 1$ coordinates, returns by evaluation the t_e remaining ones. \square

C.8 Reduction from indistinguishability between games $\text{Hyb}^{\text{OResfresh}}[e, i]$ and $\text{Hyb}^{\text{OResfresh}}[e, i + 1]$, into IND-CPA of encrypted sharing (Proposition 8)

We consider an environment \mathcal{Z} which is a challenger in the game consisting in distinguishing between $\text{Hyb}^{\text{OResfresh}}[e, i]$ and $\text{Hyb}^{\text{OResfresh}}[e, i + 1]$. It must output 1 in the former case and 0 in the latter. We construct the following challenger \mathcal{M} for the game of Proposition 8, with parameters t_{e+1} -out-of- n_{e+1} shares. It has access to \mathcal{Z} , which it can rewind. \mathcal{M} initiates a big machine consisting in the concatenation of all the system as in the game $\text{Hyb}^{\text{OResfresh}}[e, i]$: dummy adversary \mathcal{A} and all corrupt players, \mathcal{D} , the ideal functionalities, and all honest players in $\mathcal{P}_{[n_1]}^1, \dots, \mathcal{P}_{[n_e]}^e$. Then it makes this big machine interact with \mathcal{Z} . All participants in the big machine follow the hybrid $\text{Hyb}^{\text{OResfresh}}[e, i]$ (in particular, corrupt players, follow the instructions of \mathcal{Z} via the dummy \mathcal{A}), with two exceptions: players of $\mathcal{P}_{[n_{e+1}]}^{e+1}$, and P_{i+1}^e . We first describe the modifications for the former. \mathcal{M} queries public key $\text{ek}_{[n_{e+1}]}^{e+1}$ to its [encrypted sharing IND-CPA oracle], and publishes them on behalf of the honest players of $\mathcal{P}_{[n_{e+1}]}^{e+1}$. //So by construction, the secret keys are honestly sampled and not given to \mathcal{M} . Apart from this, players in $\mathcal{P}_{[n_{e'+1}]}^{e'+1}$ behave as in the hybrid $\text{Hyb}^{\text{OResfresh}}[e, i]$ (which is the same for them as in $\text{Hyb}^{\text{OResfresh}}[e, i + 1]$). Recall that they never use a secret key in these hybrids.

We now describe the modifications done by \mathcal{M} to the behavior of P_{i+1}^e . We consider every event in the execution where P_{i+1}^e accepts a valid tuple of the form $(c_{[n_e]}^e, \{c_{v \rightarrow [n_e]}, \pi_{\text{res}, i}\}_{v \in \mathcal{V}}, (c_{[n_e]}^{e-1}, \text{qvc}))$ from a collector in $\mathcal{K}_{[\kappa_e]}^e$. This happens at most κ_e times. For each of these events:

- denote, as usual, the new share c_{i+1} as the homomorphic Lagrange linear combination of the $(c_{v \rightarrow [n_e]})_{v \in \mathcal{V}}$.
- \mathcal{M} extracts, from the big machine, a plaintext new share s_{i+1} of c_{i+1} . Let us recall how. Since the $\pi_{\text{res}, i}$ are at least weak-simulation-extractable NIZKs [FKMV12], it can be extracted from the big machine: $t+1$ plaintexts $(s_{v \rightarrow i+1})_{v \in \mathcal{V}}$ of the encrypted subshares $c_{v \rightarrow i+1}$. In particular, c_{i+1} is a ciphertext of the plaintext new share $s_{i+1} := \sum_{v \in \mathcal{V}} \lambda_v^{\mathcal{V}} s_{v \rightarrow i+1}$.
- Then, \mathcal{M} queries \mathcal{O} with the pair of challenge plaintexts $(s_{i+1}, 0)$, and is returned a vector of ciphertexts, denoted as $c_{i \rightarrow [n_{e+1}]}$.
- The modification for P_{i+1}^e is now that \mathcal{M} replaces the encrypted resharing of $P_{i+1}^{e'}$ by the challenge $c_{i \rightarrow [n_{e+1}]}$.

The rest of the execution of $\text{Hyb}^{\text{OResfresh}}[e, i]$ goes unchanged. Then, \mathcal{M} outputs $b = 1$ if \mathcal{Z} outputs i , or outputs $b = 0$ if \mathcal{Z} outputs 0. But, if \mathcal{M} is facing \mathcal{O}^L , then the view of \mathcal{Z} is generated exactly as in $\text{Hyb}^{\text{OResfresh}}[e, i]$. Likewise, \mathcal{M} is facing \mathcal{O}^R , then the view of \mathcal{Z} is generated exactly as in $\text{Hyb}^{\text{OResfresh}}[e, i + 1]$. In conclusion, we have that the distinguishing advantage of \mathcal{M} is equal to the one of \mathcal{Z} , so in particular: at least at large, which concludes the proof.

D Details and proof of Refreshing squad

Refreshing squad is formalized in Algorithm 14.

We now make the assumptions of Theorem 2 and prove it.

We first prove the fact that all T^e 's are monotonically increasing, under the following more precise form.

Lemma 9. *Claim for all $e \geq 1$, at $T^e + \Delta_{\text{wait}}^e$, no honest collector in any $\mathcal{K}^{e' \geq e+1}$ has sent any (done) message yet and no $\text{qkc}^{e' \geq e+1}$ has been formed yet*

In particular, no $T^{e' \geq e+1}$ has happened yet. In particular, no honest player of $\mathcal{P}_{[n_e]}^e$ is shut-off yet.

Refreshing squad

Participants: collectors $\mathcal{K}_{[\kappa_e]}^e$, $e \geq 2$, with $\kappa_e = 2\ell_e + 1$. Players $\mathcal{P}_{[n_e]}^e$, $e \geq 1$.

Outputs: (keys-sig), (refresh-sig), (shutoff-sig)

Data structures: A *quorum of collectors certificate for epoch $e \geq 2$* , denoted as qkc^e , is the concatenation //or the succinct aggregation of signatures from $\ell_e + 1$ collectors in $\mathcal{K}_{[\kappa_e]}^e$ on the message (done). //In the special case of $e = 1$: a player of $\mathcal{P}_{[n_1]}^1$, upon receiving an input from $\text{RB}^{\mathcal{D} \rightarrow \mathcal{P}_{[n_1]}^1}$, multicasts a signed (Ack-RB) to $\mathcal{P}_{[n_1]}^1$. Then, a qrc^1 consists of a set of signatures on (Ack-RB) from $t_1 + 1$ distinct players of $\mathcal{P}_{[n_1]}^1$. (*)

A *quorum of refreshers certificate for epoch $e \geq 2$* , denoted as qrc^e , is the concatenation //or the succinct aggregation of signatures from $t_e + 1$ players in $\mathcal{K}_{[\kappa_e]}^e$ on the message (Ack-qkc).

Each collector $\mathcal{K}_k^e \in \mathcal{K}_{[\kappa_e]}^e$, $e \geq 2$: upon completing its last step (4.) in APSS0 (Algorithm 2), does:

- generate a signature τ_k on the message (done),
- **{[Rfr] multicast (done, τ_k) to $\mathcal{P}_{[n_e]}^e$ }**,
- shut-off oneself.

Each player $\mathcal{P}_i^e \in \mathcal{P}_{[n_e]}^e$, $\forall e \geq 1$:

- upon obtaining for the first time a qkc^e [Either upon receiving it from message, or by concatenating / aggregating $\ell_e + 1$ signatures from $\mathcal{K}_{[\kappa_e]}^e$ on (done)] then: **{[Trm] multicast a signed (Ack-qkc) to $\mathcal{P}_{[n_{e-1}]}^{e-1}$ }**, **{[Rfr] multicast the qkc^e to $\mathcal{P}_{[n_e]}^e$ {[Key] and to $\mathcal{P}_{[n_{e+1}]}^{e+1}$ }, then output (refresh-sig)}**.
- **{[Trm] upon obtaining for the first time a $\text{qrc}^{e'}$ for any $e' > e$ then: multicast the $\text{qkc}^{e'}$ to $\mathcal{P}_{[n_{e-1}]}^{e-1}$, and to $\mathcal{P}_{[n_e]}^e$. Then output (shutoff-sig)}**.
- **{[Key] upon obtaining for the first time a qkc^{e-1} then: multicast the qkc^{e-1} to both $\mathcal{P}_{[n_e]}^e$ and $\mathcal{P}_{[n_{e-1}]}^{e-1}$, and output (keys-sig)}**

(*) More specifically, if the RB used guarantees that honest players output at most δ from each other, e.g., as the one of [GPS19, §5], then we have the following optimization. The \mathcal{D} appends its message with a signature on it. A qkc^1 directly consists of this signature.

Algorithm 14: We tag as follows the steps relative to: **{[Rfr] starting Refresh }**, **{[Trm] termination }** and **{[Key] publication of keys }**. Recall that the outputs are to be used as inputs in APSS0. Namely, (keys-sig) instructs to generate and publish a key pair, (refresh-sig) to start Refresh as old committee, after some fixed delay Δ_{wait}^e , and (shutoff-sig) to shut-off oneself immediately.

//Notice that the claim can be rephrased as “ T^e , $e_A = e$ ”, following the formalism of Fig. 5 in Appendix A.1.3. Notice that $T^{e' \geq e+1}$ not having happened yet, does not prevent that some players of $\mathcal{K}^{e' \geq e+1}$ possibly received $\leq t_{e'}$ (done) signed by corrupt collectors, possibly forwarded to them by corrupt players

Proof. By definition of T^e and of Δ_{wait}^e , no player of $\mathcal{P}_{[n_e]}^e$ has started Refresh before $T^e + \Delta_{\text{wait}}^e$. So at $T^e + \Delta_{\text{wait}}^e$, no honest collector in \mathcal{K}^{e+1} could have sent (done) yet. So by the honest majority of collectors assumption, no player of $\mathcal{P}_{[n]}^{e+1}$ has received a quorum of (done) yet, so none has output (refresh-sig) yet, so none has started to Refresh yet. The *Claim* follows from an easy induction on $e' \geq e + 1$, which uses the same argument as the previous one for $e + 1$.

Finally, the two consequences stated as “in particular” directly follow from the fact that no $\text{qkc}^{e' \geq e+1}$ was formed yet. \square

Proof of the guarantee called *Window of opening*. At T^e , by definition, there exists a quorum of $\ell_e + 1$ signed messages (done) from distinct collectors in $\mathcal{K}_{[\kappa_e]}^e$. Since we assumed existence of $\ell_e + 1$ honest collectors, one of the signers must be honest. So it must have completed the step in Item 4, i.e., multicast a proven new sharing to all players of $\mathcal{K}_{[\kappa_e]}^e$. So they will receive it before $T^e + \delta < T^e + \Delta_{\text{wait}}^e$. By Lemma 9, none of them is shut-off when it receives it. In conclusion, at latest at $T^e + \Delta_{\text{wait}}^e$, all players of $\mathcal{P}_{[n_e]}^e$ have accepted at least one vector of new shares, $c_{[n_e]}$, in their local lists. Said otherwise, the common set of shares holds at this point. Since by assumption they all received open-sig by then, they will have all sent their share of $c_{[n_e]}$ to the learner \mathcal{L} . So \mathcal{L} receives all these shares by $T^e + \Delta_{\text{wait}}^e + \delta$. By the honest majority assumption there exists at least $t_e + 1$ honest players in $\mathcal{P}_{[n_e]}^e$, so \mathcal{L} has then enough shares to output.

Proof of the guarantee called *Last-minute key generation*. When the first honest player of $\mathcal{P}_{[n_{e+1}]}^{e+1}$ outputs (keys-sig), it must be the case that it received a qrc^e and forwarded it to $\mathcal{P}_{[n_e]}^e$. So T^e happens no later than δ from this point. The result follows since players of $\mathcal{P}_{[n_{e+1}]}^{e+1}$ generate and publish their keys Δ_{wait}^e after they output (keys-sig).

Proof of the guarantees called *Last-minute key Refresh* and *Fast shutoff*. We divide the set of executions in three cases, and prove that the guarantee holds in each of these cases. Notice that cases I] and II] possibly overlap, while case III] is disjunct from both cases I] and II].

Case I]: at $T^e + \Delta_{\text{wait}}^e + 3\delta$, some honest players of $\mathcal{P}_{[n_{e+1}]}^{e+1}$ are already shut-off. By Lemma 9 this implies that T^{e+1} already happened, proving *Last-minute key Refresh*. Furthermore, before it shuts-off, a honest player of $\mathcal{P}_{[n_{e+1}]}^{e+1}$ multicasts a $\text{qrc}^{e'} \geq e+1$ to $\mathcal{P}_{[n_e]}^e$. So all of them will have shut-off before $T^e + \Delta_{\text{wait}}^e + 4\delta$, proving *Fast shutoff*. //The multicast to $\mathcal{P}_{[n_e]}^e$ could be skipped, although this would increase by δ the guaranteed delay in *Fast shutoff*.

Case II]: at $T^e + \Delta_{\text{wait}}^e + 3\delta$, some honest players of $\mathcal{P}_{[n_e]}^e$ are already shut-off. Then it must be that they received a $\text{qrc}^{e'} \geq e+1$, so it must be that some $T^{e'} \geq e+1$ already happened. By Lemma 9, these times being monotonically increasing, it must be that T^{e+1} has happened, proving *Last-minute key Refresh*. Furthermore, they have forwarded the $\text{qrc}^{e'} \geq e+1$ to all $\mathcal{P}_{[n_e]}^e$ before they shut-off. This enables all honest players of $\mathcal{P}_{[n_e]}^e$ to shut-off before $T^e + \Delta_{\text{wait}}^e + 4\delta$ //this is where we use the forwarding of the $\text{qrc}^{e'} \geq e+1$ to all $\mathcal{P}_{[n_e]}^e$. Not doing so would make some honest players of $\mathcal{P}_{[n_e]}^e$ possibly wait δ longer before they receive a $\text{qrc}^{e'} \geq e+1$ (generated in a higher committee).

Case III]: at $T^e + \Delta_{\text{wait}}^e + 3\delta$, no player of $\mathcal{P}_{[n_e]}^e$ nor $\mathcal{P}_{[n_{e+1}]}^{e+1}$ is shut-off yet. Consider P^e a first player of $\mathcal{P}_{[n_e]}^e$ which receives a qkc^e , at T^e . At most at $T^e + \delta$, the following three events have happened:

- all players of $\mathcal{P}_{[n_e]}^e$ received the qkc^e , and thus output (refresh-sig);
- all players of $\mathcal{P}_{[n_e]}^e$ received a proven new sharing in common //this is because there exists a honest collector in the quorum of the $\ell_e + 1$ signers of qkc^e . Thus, they have a set of encrypted shares, $c_{[n_e]}$, in common in their local lists \mathbb{L}_i^e ;
- all players of $\mathcal{P}_{[n_{e+1}]}^{e+1}$ have output (keys-sig).

From these three events, it follows that, at $T^e + \delta + \Delta_{\text{wait}}^e$:

- all keys of honest players in $\mathcal{P}_{[n_{e+1}]}^{e+1}$ have been published;
- there exists a vector of ciphertext shares of the secret, $c_{[n_e]}$, such that all honest players of $\mathcal{P}_{[n_e]}^e$ have generated an encrypted resharing of their share of $c_{[n_e]}$ and sent it to all collectors $\mathcal{K}_{[\kappa_{e+1}]}^{e+1}$.

So at $T^e + 2\delta + \Delta_{\text{wait}}^e$, all $\ell_{e+1} + 1$ honest collectors of $\mathcal{K}_{[\kappa_{e+1}]}^{e+1}$ will have sent (done) messages to $\mathcal{P}_{[n_{e+1}]}^{e+1}$. They are received at most at $T^e + 3\delta + \Delta_{\text{wait}}^e$, which concludes the proof of the *Last-minute key Refresh*.

At this point, all honest players of $\mathcal{P}_{[n_{e+1}]}^{e+1}$ will have sent (Ack-qkc) to $\mathcal{P}_{[n_e]}^e$. So at most at $T^e + 4\delta + \Delta_{\text{wait}}^e$, all players of $\mathcal{P}_{[n_e]}^e$ will have received a quorum of $t_{e+1} + 1$ signed (Ack-qkc) of $\mathcal{P}_{[n_{e+1}]}^{e+1}$, when they are not already shut-off. Such a quorum forms a qrc^{e+1} , which triggers a shut-off, concluding the proof of *Fast shutoff*.

E Fixing the related asynchronous *mobile-PSS* model: *mobile* corruptions, in a *static* committee

E.1 The related *mobile-PSS* model

E.1.1 Broad definition Following [OY91], we denote the following model as *mobile-PSS*. It is considered in [AGY95; FGMY97; ADN06; BELO14; CMP20; GDK22; ABKL22]. It is also considered in [SLL10, p12], although their main focus is dynamic-PSS. This model considers a fixed set of n players, in which the set of corrupt players changes over time. Time is broadly measured with a counter, which is either local or

global, denoted as *epoch*. Broadly speaking, players transition to the next epoch upon finishing a Refresh. The meaning of “finishing” varies from works to others. This meaning impacts the model and its implementability, as exemplified in Appendix E.1.2 on the case of [ABKL22, p 27]. Broadly speaking, the adversary \mathcal{A} is allowed to corrupt at most t players per epoch, where t is a public parameter denoted *threshold*. The precise meaning of “per epoch” varies from works to others. In turn, their stated corruption tolerance varies depending on this meaning, as explained in Appendix E.1.2. Corrupt players have their memory erased. At any point, \mathcal{A} may de-corrupt a player. In some PSS [HJKY95; SLL10], there is a specific protocol, denoted as *Recover*, which aims at somehow providing those players with a share of the secret, which is consistent with the other shares owned by honest players in the current epoch. In some other PSS [BELO14], the job of *Recover* is handled by an all-in-one Refresh. In these PSS, de-corrupt players are put in quarantine ([CMP20]) until they join the protocol again at the beginning of the next Refresh, from which they are guaranteed to automatically obtain a new share for the next epoch. Notice that [SLL10; VAFB22] have both a *Recover* and an all-in-one Refresh.

E.1.2 The problem in the existing *asynchronous mobile-PSS* model of [SLL10] It seems to us that only two rigorous models for asynchronous mobile-PSS exist so far. The one of [CKLS02] defines global epochs as ticks of an external global clock. As a result, their liveness is guaranteed only in executions ([CKLS02, p 18]) in which the global clock ticks after all messages of the current epoch were delivered. So this is a form of synchrony assumption.

The one of [SLL10, §5.1], by contrast, defines epochs purely from the protocol. In short, the main problem in the model of [SLL10], is that it allows potentially $2t$ players to be corrupt while they hold secret material related to epoch $e + 1$: those doing the $(e - 1)$ -to- e Refresh, counting in the corruption budget for e , and those in epoch $e + 1$, counting in the corruption budget for $e + 1$. Let us now describe precisely what this means and its implications in terms of corruption threshold. The model of [SLL10, §5.1] calls a player as “(locally) in epoch e ”: from the point where it finishes the $(e - 1)$ -to- e Refresh, until the point where it finishes the e -to- $(e + 1)$ Refresh. Notice that finishing a Refresh typically involves, in particular, erasing some memory. They allow that, for each e , a maximum of t players are corrupt while they are locally in epoch e //they actually phrase this in an equivalent way, in terms of so-called global “system epochs”. Notice that [ABKL22, p 27] (their second attack scenario) specify a variant, in which a player is said to finish an epoch as soon as it receives its new share //this event could happen before it terminates the Refresh. For instance, they point that, in [YXXM23], the player continues the Refresh to help other players finish the consensus and deduce their new share. However, they prove that this variant is actually unimplementable //the reason being that they allow such a player to be corrupt using the corruption budget of the new epoch, even before it finished the Refresh. The corruption model of [SLL10, §5.1] is currently borrowed in [GDK22] and has the following concrete implication. It allows that t players are corrupt at the end of a Refresh, after they learned their new share. Furthermore it allows, in addition, that t other players are corrupt after the end of a Refresh, when by definition they a fortiori know their new share. As a result, since their sharing degree is $\leq n/3$, it follows that their corruption threshold is actually such that $2t < n/3$. The same problem happened in the v1 of [YXXM23], leading also to a twice-lower-than advertised corruption threshold, of $2d < n - t - 1$. We notified in end-February 2023, to both the authors of [YXXM23; GDK22], the problem with their model inherited from [SLL10]. We notified to them a fix, which is well-known in the synchronous mobile-PSS setting ([FGMY97; ADN06; BELO14]). It consists in counting in the corruption budget of *both* adjacent epochs, a player which is corrupt during a Refresh. They acknowledged that they had already identified the problem and were planning to fix it. The fix was quickly done in the March 2023 version of [YXXM23].

Finally, we observe that the problem with [SLL10, §5.1] can also be illustrated with the related model of [ABKL22, p26] (their first attack scenario). This related model does not count either, in the corruption budget of epoch e , in some corner-cases, a player still performing a e -to- $(e + 1)$ Refresh //their corner-case is a player which receives a late private message related to the e -to- $(e + 1)$ Refresh, so this player somehow still uses its old decryption key. As a result, they show that their model is not implementable.

E.2 A repaired general asynchronous mobile-PSS model

We first state a model, denoted as *asynchronous mobile-PSS* and shortened as mobile-APSS. It is both a repairing of the model of [SLL10, §5.1], and a generalization of it to higher corruption thresholds than $t < n/3$. Then we briefly fix or cast existing models or protocols in it.

For simplicity we state informal standalone properties. Transforming them into an ideal functionality would follow exactly the same formalism as $\mathcal{F}_{\text{P-AVSS}}$ (Fig. 5). We consider n players, a parameter t denoted as *threshold*, and a protocol Π which consists of the following subprotocols. **Share** is between any distinguished dealer(s) \mathcal{D} , then for each $e \geq 1$ we have subprotocols $\text{Refresh}^{e \rightarrow e+1}$ between the players, and finally **Open** is between the players and any distinguished learner(s) \mathcal{L} . Moreover, $\text{Refresh}^{e \rightarrow e+1}$ contains a special instruction, which we denote as **shut-off** ^{e} . Informally, it instructs the player to erase from its memory all material related to epoch e //although it may still continue $\text{Refresh}^{e \rightarrow e+1}$, e.g., terminate the consensus, in consensus-based APSS for $t < n/3$. We say that a player is in epoch e if it has started $\text{Refresh}^{e-1 \rightarrow e}$, and did not **shut-off** ^{e} yet in $\text{Refresh}^{e \rightarrow e+1}$ //but can possibly have terminated $\text{Refresh}^{e-1 \rightarrow e}$. Hence, a player doing $\text{Refresh}^{e \rightarrow e+1}$ and did not **shut-off** ^{e} yet, is simultaneously in epochs e and $e + 1$. The model is orthogonal of the various possible mechanisms triggering players to start a **Refresh** or **Open**, exemplified below. For each e , the adversary \mathcal{A} can corrupt at most players which are in epoch e . In particular, if a player is corrupt while simultaneously in epochs $e-1$, e and $e + 1$, then it counts in the corruption budget of all those three epochs. \mathcal{A} can decorrupt players at any time, e.g., just before it starts a **Refresh**. We denote the model as *static* if \mathcal{A} cannot corrupt an honest player in the middle of a **Refresh**//the proof of APSS0, in Section 5, is in the static corruptions model. Extensions to the adaptive corruptions model are discussed in Appendix B.2. We say that Π is a *PSS in the mobile-APSS model*, if it satisfies the rough standalone specifications made in the beginning of Section 1. They were called *Secrecy*, *Liveness* and *Correctness*, with the straightforward changes of notation needed ($t_e \rightarrow t$ etc.). More precisely, *Liveness* implies the following. Consider an execution in which **Share** (s) terminate(s), then, for all e up to some $e_o - 1$, players are triggered to start $\text{Refresh}^{e \rightarrow e+1}$. Then it is guaranteed that all these **Refreshes** terminate, and that subsequently, players are in a state in which they can **Open** the (desired linear combination of) secrets to \mathcal{L} .

A first example can be seen as [CKLS02, p18], when instantiated with a global clock which would wait that all messages of a **Refresh** are delivered, before ticking the next **Refresh**. There, a mechanism instructing players to **shut-off** is by definition the ticking of the next **Refresh**. More precisely, upon being notified a tick, a player realises an AVSS of its new share, if it has one. In any case, it stops playing the ongoing previous **Refresh**, if it did not terminate yet, and immediately erases its old share, if it has one.

A second example can be seen as [GDK22; YXXM23], provided, in the former, an adaptation of their corruption model to the mobile-APSS one, in which corruptions during a **Refresh** count in both epochs //the fix was done very recently in [YXXM23]. Their termination mechanism is simply that players terminate a **Refresh** upon terminating in the consensus. On the face of it, it is left unspecified in [GDK22; YXXM23] if players should **shut-off** immediately in the middle of an ongoing **Refresh** and start a new one when instructed to do so, e.g., by the trusted coordinator of [YXXM23]. The answer seems to be implicitly no, that players in [YXXM23] actually wait for terminating the previous **Refresh** the normal way, even if urged to start a new one. So this differs from [CKLS02, p18]. Otherwise, their claimed liveness in [YXXM23, §B] would not hold. In both [GDK22; YXXM23], the corruption threshold does not go beyond $t < n/3$. However, privacy is still guaranteed in [YXXM23] even if up to a total number of d players are corrupt, where $d < n - t$ is a parameter.

The third example is APSS, when compiled into the mobile-APSS model, as explained Appendix E.3. For simplicity we assume that collectors \mathcal{K}_k^e are equal to all players. Liveness is guaranteed as long as there is at least one collector in each epoch which plays the protocol, i.e., is no more than passively corrupt. The model straightforward extends to any mechanism which would sample collectors for each epoch as a subset of the players. Players are instructed to start a **Refresh** in two ways. Upon receiving (**refresh-sig**) from **Refreshing squad** (Section 6), they wait Δ_{wait}^e , then wait until all keys of epoch $e + 1$ are published, then start playing $\text{Refresh}^{e \rightarrow e+1}$ as exiting committee $\mathcal{P}_{[n_e]}^e$. Upon receiving (**keys-sig**) from **Refreshing squad** (Section 6), they wait Δ_{wait}^e , then start playing $\text{Refresh}^{e \rightarrow e+1}$ as entering committee $\mathcal{P}_{[n_{e+1}]}^{e+1}$. Finally, the instruction to **shut-off** simply comes as the (**shutoff-sig**) delivered by **Refreshing squad**.

E.3 Compilation from dynamic-asynchronous PSS, into mobile-APSS

We consider any *dynamic-PSS*, which we define as following the rough standalone specifications made in the beginning of Section 1. We will consider later more detailed specifications, which vary in the literature. We consider committees of fixed size n and with all corruption thresholds at (n, t) . Each player \mathcal{P}_i initiates in its head all $(\mathcal{P}_i^e)_{e \geq 1}$ in parallel. By definition \mathcal{P}_i starts $\text{Refresh}^{e-1 \rightarrow e}$ when the first of the two simulated $(\mathcal{P}_i^{e-1}, \mathcal{P}_i^e)$ starts Refresh //in practice in APSS the latter \mathcal{P}_i^e is always the one which starts first, since \mathcal{P}_i^{e-1} does not start before all keys of $\mathcal{P}_{[n_e]}^e$ are published. By definition \mathcal{P}_i shuts-off in $\text{Refresh}^{e-1 \rightarrow e}$ when \mathcal{P}_i^{e-1} shuts-off. Hence, \mathcal{P}_i may be in multiple epochs in parallel.

$$\begin{aligned} \text{dynamic-PSS} &\longrightarrow \text{mobile-PSS} \\ (\mathcal{P}_i^e)_{e \geq 1} \forall i \in [n] &\longrightarrow \text{each } \mathcal{P}_i \text{ runs all } (\mathcal{P}_i^e)_{e \geq 1} \text{ in parallel.} \\ \mathcal{P}_i^{e-1} \text{ or } \mathcal{P}_i^e \text{ started } \text{Refresh}^{e-1 \rightarrow e} \text{ and } \mathcal{P}_i^{e-1} \text{ not shut-off} &\longrightarrow \mathcal{P}_i \text{ is in epoch } e \end{aligned}$$

The main property achieved by the compiler is that, if the dynamic-PSS protocol on the left has corruption threshold t , then also does the resulting mobile-APSS protocol.

F More on generalizations and applications

F.1 More on LHE with Limited evaluation mod p

We survey some schemes satisfying our Definition 3, of LHE supporting a limited number of linearly homomorphic evaluations and with bilateral binding. In Appendix F.1.1 we start by a general remark to greatly improve efficiency in APSS0. Throughout we use the notation of Section 3.

F.1.1 Allowing slack in the NIZKs of smallness For simplicity and clarity, below Definition 3 we allowed to pick any plaintext bound and randomness bound, (M, R) , such that

$$(8) \quad M \geq p^2(t+1) \text{ and } R \geq p^2(t+1)R_{\text{enc}} .$$

Precisely, we recall that R_{enc} is a public parameter such that, when encrypting, encryptors are meant to choose encryption randomness below R_{enc} . As a result, in APSS0 we specified that resharers *prove* in NIZK that the norms of the plaintexts subshares and encryption randomnesses, are below p and R_{enc} . If they did not, then correctness would be broken. Now, we would like to increase efficiency by using NIZKs “of smallness”, i.e., allowing that a malicious prover can pass verification with input size larger, by some *slack*, than the maximum size that an honest prover is able to input. To this end, we narrow the choices of (M, R) , roughly as follows:

$$(9) \quad M \geq (p + \text{slack})p(t+1) \text{ and } R \geq p(t+1)(R_{\text{enc}} + \text{slack}); .$$

The idea is that the slack in the LHS enables to tolerate that malicious prover could pass the NIZK verification, despite using plaintexts which are up to slack larger than the p allowed, and encryption noises which are slack larger than the R_{enc} allowed.

Turning to Paillier, one can observe that NIZKs of re-sharing are eased by publicizing Pedersen (or Feldman) commitments to the Paillier plaintext subshares (precisely, as in Section 7.3), which come appended to the encrypted resharing. Then, one is left with proving equality of the plaintexts with the openings of the commitment, then proving smallness of the committed values. State of the art implementations of such NIZKs are in [LNR18, §6.2] and [CGG+20], applied to threshold ECDSA. This same observation is made, in GH [GHL22], for lattice-based schemes. They bring optimized NIZK relations (instantiated with Bulletproofs) for resharing, on the example of the Peikert-Vaikuntanathan-Waters PKE. They were recently improved in [LNP22].

F.1.2 Exponential Elgamal Encoding the secret *in the exponent* of Elgamal, in order to use additively homomorphic properties, was leveraged in [Sch99, p11] in the context of electronic voting. Concretely, consider $(\mathbb{G}, 0, +)$ a cyclic group with hard DDH and H any public generator. We set $R = \infty$, and $M < |\mathbb{G}|/2$ an arbitrarily chosen bound such that, for any $x \leq M$, computing x given $x.H$ is deemed efficient. The (front-end) plaintext space is $\mathbb{S} := \mathbb{F}_p$, for any prime $p \leq M$. Given a plaintext $m \in \mathbb{S} = \mathbb{F}_p \subset \mathbb{M} = \mathbb{Z}$, encrypt $m.H$ with Elgamal in \mathbb{G} . Decryption proceeds by: (i) Elgamal decryption, into a plaintext $X \in \mathbb{G}$, (ii) followed by computing the discrete logarithm, i.e., extracting x such that $X = x.H$, (iii) then outputting $x \bmod p$. In [TLC+22], following [CMTA20; CCN21], they consider \mathbb{G} an elliptic curve group with security 128 bits, and report on a decryption of Exponential Elgamal ciphertexts in 0.35ms on one thread, for plaintext sizes of 40bits (“about 30 times faster than Paillier”), then in 1s on 16 threads for 54bits sizes. The terminology “exponential Elgamal” is from the proposition [ISO19].

F.1.3 Paillier-mod-p Paillier encryption does not support unlimited homomorphic additions modulo a fixed prime p , since the plaintext space is a $\mathbb{Z}/N_i\mathbb{Z}$ for a composite modulus N_i . This modulus N_i is furthermore unavoidably different for each player \mathcal{P}_i since it depends on its private decryption key. One observation which we put forth in this paper is that it is enough for players to use a common (front-end) plaintext space \mathbb{F}_p , which concretely is a common interval $[0, \dots, p-1] \subset [0, \dots, N_i] \subset \mathbb{Z}$, as long as the few linearly homomorphic linear combinations applied in APSS0, do not take plaintexts outside of the interval $[-(N_i-1)/2, (N_i-1)/2] \subset \mathbb{Z}$. This observation can be formalized into the following instantiation of the Definition 3 of LHE, which one may denote as “Paillier-mod p ”. Set the parameters of key generation of Paillier, such that the modulus is of minimal size N , to be specified later. Define the (front-end) and (back-end) plaintext spaces as $\mathbb{S} := \mathbb{F}_p = [0, \dots, p-1] \subset \mathbb{M} := \mathbb{Z}$. To encrypt an element $m \in \mathbb{F}_p$, encrypt it with Paillier. To decrypt a ciphertext under key N_i : decrypt it with Paillier into some x , take the representative $\tilde{x} := x \bmod N_i$ in the interval $[-(N_i-1)/2, (N_i-1)/2]$ //concretely: subtract N_i is the decryption is in $](N_i-1)/2, N_i-1$, otherwise do nothing, then output $\tilde{x} \bmod p$. It satisfies Definition 3 of LHE, with respect to parameters $M := (N-1)/2$ and $R = \infty$. For usage in APSS0, given targets t and p , we choose N such that $M \geq (p + \text{slack})^2(t+1)$ (where the meaning of *slack* is given in Appendix F.1.1). We refer to Appendix F.1.1 for efficient NIZKs.

F.1.4 Lattice-based schemes The goal of this paragraph is to exemplify that, since lattice-based schemes allow in principle unbounded encryption noises, it is necessary for our purpose to cut-off the queues of their distribution, so as to force malicious encryptors and decryptors to use noises which are small enough to ensure correctness. This cut-off is more or less implicit in existing works on maliciously secure MPC from FHE [GLS15; Coh16]. A variant of Regev over $\mathbb{Z}/p\mathbb{Z}$ is provided in [BDOZ11, §2.1]. They sketch bounds on the plaintext and noise, denoted M and R , for correct decryption mod p for any encryptors and decryptor which honestly sample their randomnesses. However this analysis is not enough for our purpose, since nothing prevents a malicious colluding pair of encryptor-decryptor to choose their key randomness, and encryption randomness, outside of these statistical bounds. So, in order to enforce correctness mod p even if they are malicious and colluding, as in Definition 3, one must furthermore specify the randomness of the key generation, as belonging to a finite interval \mathcal{R}_{key} . Concretely, one must cut-off the queues of the Gaussian. Recall that in our formalism we also incorporated to dk all the noise necessary to produce ek. So when we specify, in APSS0, that the NIZK should prove that “ $\text{dk} \in d\mathcal{K}$ ”, what we are implying is that this noise is proven to be below a cap (up to some slack).

The same necessary cut-off also applies to instantiations of LHE from other lattice-based schemes such as BFV/BGV [BFV/BGV21]. We refer to Appendix F.1.1 for efficient NIZKs.

F.2 DKG and Applications to threshold signatures / randomness generation / decryption

F.2.1 More on the commitments-to-subshares variant of APSS in Section 7.3 The resharer \mathcal{P}_i appends an argument of knowledge of: decommitments of the $(\text{com}_{v \rightarrow i})_{v \in \mathcal{V}}$, such that they are equal to plaintexts of $c_{i \rightarrow j}$, and such that they form a resharing of s_i , such that s_i is a Lagrange reconstruction of openings of the $(\text{com}_{v \rightarrow i})_{v \in \mathcal{V}}$.

F.2.2 Biased DKGs The goal of this paragraph is merely to formalize a weaker form of DKG, which is thus simpler to implement, in which the shared secret (key) has some bias controlled by \mathcal{A} . Then we recall the state of the art applications which allow such biased DKG. As pointed out by [GJKR07], a number of DKG protocols implement a *biased* dealer. This is formalized by [BDO22, Fig. 2] as follows. The dealer samples s uniformly at random and leaks a Feldman commitment $s.H$ to s . Then the adversary can decide to offset s by the value δ of its choice: $s \leftarrow s + \delta$.

We quote [GJM+21] for a recent account of the impacts of such bias: “[GJKR07] previously observed that the Pedersen DKG suffices to construct threshold Schnorr signatures. Recently, Benhamouda et al. [BLL+21] found an attack on this approach when the adversary is concurrent. (Gennaro et al. had not considered concurrent adversaries.) Komlo and Goldberg [KG20] show that it is possible to avoid the attack but, in doing so, they lose robustness (e.g., if a single party goes offline a signature will not be produced). This raises questions as to whether it is still okay to use the Pedersen DKG with respect to other signature schemes such as BLS. In this paper, we provide a positive answer in the form of a security proof that holds concurrently and does not rely on rewinding the adversary. Specifically, we show that the Pedersen DKG is security-preserving with respect to any rekeyable encryption scheme //including BLS, signature scheme, or VUF scheme //the one considered in Section 7.2 where the sharing algorithm is the same as encryption or signing (see Definition 5)”

We quote [BDO22] for an update: “In Section 3.1 we showed that the bias can also be tolerated in ElGamal-style encryption schemes i.e., we showed that the [[CCL+20, §3.2]] encryption scheme does still provide IND-CPA security even when the adversary is allowed to bias the distribution of the key. ”

F.2.3 Step-by-step APSS0-Open of a BLS signature on a message m Consider a committee $\mathcal{P}_{[n_e]}^e$, holding a secret-shared signing key $s \in \mathbb{F}_p$, which is instructed to open a BLS signature on some message m , i.e., $H(m)^s$. Concretely, each $\mathcal{P}_i^{(e_o)}$, for each element $\mathbb{L}_i[k]$ in its list, denoting as $c_{[n]}$ the vector of ciphertext shares and $s_i.G$ the i -th public key share, sends to \mathcal{L} : $s_i.H(m)$, with a proof of equalities of discrete logs between this and $s_i.G$. Upon receiving $t_{e_o} + 1$ such proven signature shares, generated from the same $(c_{[n]}, (vk_i)_{i \in n_{e_o}})$, vouched by a quorum of signatures, \mathcal{L} does the further check of APSS2. Namely, it checks if the $(vk_i)_{i \in n_{e_o}}$ do interpolate in the exponent into $s.G$ (this guarantees correctness even if all signatures in the quorum are from corrupt players). Then, it interpolates in the exponent the signature: $s.H(m)$.

F.2.4 Threshold opening of a (R)LWE ciphertext In [KJY+20, §B] the threshold decryption of RLWE-based ciphertext (c_1, c_2) , is roughly the opening of the linear map

$$(10) \quad s \mapsto c_1 s + c_2$$

followed by a local rounding of the value opened. The multiplication $s.c_1$ should be understood between polynomials. To achieve this, they consider in [KJY+20, §A] an extension of the Shamir scheme, with secrets and shares in polynomial rings, defined by evaluation of a degree t polynomial with polynomial coefficients. This scheme has the property that multiplication of the shares by a polynomial c_1 commutes with polynomial multiplication of the secret s by c_1 . However, opening of the plain value Equation (10) is unsafe. Informally, it leaks information on the secret key s . Formally, the value opened is not simulatable from the final rounded value. To restore simulatability, a common trick consists in adding noise to the opening. This traditionally comes in the form of noise added locally by the players to their decryption shares. A trick is introduced in [GLS15], which consists instead in generated a unique secret-shared noise e . Then, Equation (10) becomes the opening of the linear map: $(s, e) \mapsto c_1 s + c_2 + e$.

F.2.5 Oblivious Preimage-sampleable commitments In the proof of APSS0, the simulator produces openings which are not correct decryptions of encrypted shares. This is possible since the simulator generates NIZK proofs of false statements, and since we specified that extraction should work even after \mathcal{Z} saw such proofs. Now in the variant described in Section 7.3, the simulator must exhibit openings to commitments of

opening shares, which are equal to the target opening shares. To this end, we must assume that there is a local setup of the commitment scheme, so that the simulator can generate a trapdoor of it. This trapdoor enables to generate an opening of any commitment Com , into any target value x . As stressed in [GPV08; CGM19], revealing to \mathcal{Z} such openings may leak information about the trapdoor. So we are aiming at commitment schemes satisfying a property, denoted as *uniform preimage sampleable*. In [GPV08] it is shown that Ajtai’s commitment satisfies this property, which comes as follows. Consider the uniform distribution over the space of (values, randomness): SampleDom , in particular, which is public independent from the trapdoor. There is a function SamplePre which, given a commitment com , uses the trapdoor to output an opening (x, r) . It has the property that the distribution of pairs (opening (r, r) output by SamplePre , input commitment com) is indistinguishable from the distribution of pairs $((x, r) \leftarrow \text{SampleDom}, \text{commitment } \text{Com}(x, r))$. In conclusion, the simulator just needs to sample every commitment in the simulation using SampleDom -then- Com , then it can safely open them later, using SamplePre , without leaking any information about the trapdoor. They even achieve that SampleDom can be allowed to be the uniform distribution. Their result applies to SIS-based commitments such as BDLOP [NS22, §2.4] or Ajtai’s or a combination of both [LNP22, §3.1].

Pedersen commitment trivially satisfies the property of *uniform preimage sampleability*, since the trapdoor enables to sample uniformly in the set of openings of a given value.

G Related specifications of (asynchronous and/or proactive) VSS

The terminology *verifiable* secret sharing (VSS) is a legacy one [CGMA85; Ped92; CR93; BCG93; Can95; BKR94; PCR13; PCR14; BKP11; BDK13; BTA+19; KMS20; AVZ21; YLF+22; CCP21]. It is given as many different specifications as different works on the topic. *Verifiability* is the guarantee that, after the Share algorithm “completed” (this word having many different meanings), then the dealer is **committed** to a well-defined value s , such that only this value can subsequently be reconstructed in the subsequent Open protocol.

G.1 Standalone definitions of (asynchronous) Verifiable secret sharing (A)VSS

We start by stating a standalone definition of asynchronous verifiable secret sharing (AVSS), which is matched by APSS even after an unlimited number of Refreshes . Then we compare to the many specifications of AVSS in the literature. AVSS is a pair of algorithms ($\text{Share}, \text{Open}$), parametrized by $t < n$, with the following syntax and requirements.

- Share is between a dealer \mathcal{D} with an input in \mathbb{S} , and n players//in our dynamic setting they are $\mathcal{P}_{[n_1]}^1$. We consider here $n_1 = n$ for simplicity. Players may output (**committed**).
 - Open takes place between n players//[$\mathcal{P}_{[n_1]}^1$ or a subsequent committee, if Refreshes took place in-between] and a learner \mathcal{L} . In the general case of several instances of Share to the same players, possibly from several dealers, then, (each instance of) Open is parametrized by a public linear map $A : \mathbb{S}^* \rightarrow *$. \mathcal{L} may output a value.
1. [*Correctness, a.k.a., verifiability (VSS); with strong linearity (SL):*] as soon as one player outputs in Share , there exists a well defined value, $s \in \mathbb{S}$, which we say is **committed**, such that the following properties hold:
 - if \mathcal{D} is honest then s is its input;
 - consider an execution where all input variables of A were **committed**. Denoting as $(s_w)_{w \in W}$ the values **committed**, we have that $A((s_w)_{w \in W})$ is the only value that \mathcal{L} can possibly output in Open .
 2. [*Termination of Share:*] if one honest player outputs in Share , then all honest players ultimately output in Share .
 3. [*Secrecy:*] if \mathcal{D} is honest, then \mathcal{A} cannot distinguish between two chosen secret inputs of \mathcal{D} from their Share .
 4. [*Liveness (Completeness) (ACSS):*] after all honest players output in the Share of each secret involved in the linear combination A , consider any subset of $t+1$ players which are passively corrupt or honest and which start Open . Then, even if the others are completely offline forever, \mathcal{L} ultimately outputs in Open .

(ACSS) The first works on AVSS [BCG93; CR93; Can95], and also [PCR14], specify a guarantee which is weaker than ACSS. We may call it *non-complete*. In a non-complete AVSS, even after all players terminated share, it is guaranteed only that $n - 2t$ honest players received shares of s . So, considering secrets shared from several dealers, it could be the case that the intersection of players having shares of all of them is not large enough to enable reconstruction of a linear combination of the secrets //a thinner analysis, in [CP22, §1.2], shows that *non-complete* prevents linear combinations of even only two secrets. On the other hand, the non-complete specification is perfectly fine for the purpose of reconstructing one secret. A method is clearly exposed in [Can95, p. 5.7] how to build a 1/4-common coin from this sole specification. For this reason, [BKR94] introduced and implemented a stronger guarantee called *ultimate* secret sharing. Roughly and informally, it specifies that all honest players outputs a share in *Share*. It is then renamed as asynchronous *complete* secret sharing (ACSS) in [Cho20b; CP22; YLF+22] It is then re-named as *recoverable* verifiable secret sharing in [BTA+19] (which are under partial synchrony). Completeness is matched by [CKLS02; PCR13], even if they do not use this word. Thanks to replicated secret sharing, the share recovery protocol of [ZSV05, Figure 5] is non-interactive, although at the cost of an exponential complexity. Completeness is also matched by [CKLS02; PCR13], even if they just use the plain terminology of “AVSS”. From a remote perspective, completeness is incorporated in the property denoted as *strong commitment* (SC) in [BKP11, §2.2 2.]. But actually, as specified, the SC of [BKP11, §2.2 2.] is surprisingly *weaker* than the plain “commitment” specification of [BKP11, §2.2], at least when $t \geq n/3$. The reason is that, their “commitment” specifies that the *opened value* is equal to the *committed* one. Whereas, their “strong commitment” only requires that *Share* delivers *shares* of the *committed* value to honest players. However, raw Shamir shares are not enough to guarantee robust reconstruction of the shared secret if $t \geq n/3$. We refer to [CDN15] (p121: “Minimal Distance Decoding”) and to [Can95] (4.4.4 “on-line error correcting”) for this well-known result. This is a mere definitional problem since the VSS protocols of [BKP11] do have robust reconstruction, thanks to a common view on public commitments to shares. Also, their definitions under asynchrony, in [BKP11, §4.1], do not capture either completeness. The reason is that they do not guarantee that a secret will be opened, unless *all* honest players start *Open*.

To avoid the above definitional problems linked to shares, we instead phrased the completeness property above (ACSS) in terms of *openability* of secrets *from any subset* of $t + 1$ passively corrupt or honest players. This somehow implies the aforementioned rough and informal definition of ACSS, which is that every player outputs a share in *Share*. Said in another way, our specification of (ACSS) can be seen as an upgrade of the “strong commitment” (SC) of [BDK13]. Namely, instead of just guaranteeing that all $n - t$ honest players get a share, we guarantee *openability* of secrets from any $n - t$ players.

Finally, we can observe that [PCRR09; PCR14] denote as “strong commitment” a property which we denoted above as *correctness, a.k.a., verifiability*.

(SL) The terminology *strong linearity property* is introduced in [GGOR13]. They observe that there exists alternative possible specifications of linearity. They make such an alternative specification, which they call *proper + improper*. This alternative specification is that a *committed* value can possibly be in \perp , and that a linear combination involving a *bot committed* value returns \perp .

We now make the observation that this alternative specification is strictly weaker than strong linearity. The reason is that it gives to the adversary the power to *force* the output of a linear combination to be equal to a value of its choice, i.e., \perp . By contrast, strong linearity gives only the power to the adversary to *add* an offset to the output, which it must choose *before* learning any information on the values *committed* by honest players. Notice that, in the case of DKG, then the adversary may also learn a Feldman commitment to values *committed* by players, before it decides the offset to be added to the sum of these values (Appendix F.2.2). Notice that in these observations, we *do not* consider that the adversary can equivocate in the opening, i.e., if it is *committed* to some $v \in \mathbb{S}$, then we did not consider that it can change it later for \perp . So these observations are orthogonal to the distinction between VSS and WSS, in which such equivocation is possible.

In [PCRR09; PCR14], they also consider the weaker specification of VSS where the *committed* secret can be in $\mathbb{S} \cup \{\perp\}$. Furthermore they do not specify any openability of linear combinations. So they observe, [PCRR09, §2.1], that their specification is “not suitable for use in MPC”. It is however sufficient for the

use-case of implementing a common coin, as in [PCR14] and discussed above. They also observe that, while they are at requiring only the opening of *one* secret, then, allowing the committed value to be in $\mathbb{S} \cup \{\perp\}$ actually emulates the case where the committed value is in \mathbb{S} only //if the output of `Open` is \perp , then output 0. In [BDK13], the requirement that the committed value is in \mathbb{S} , not only $\mathbb{S} \cup \{\perp\}$, is one of the two specifications which are included in their “strong commitment” definition but not in their plain “commitment” definition (the other one being described above in (ACSS)). The same non-commitment-to- \perp shows-up in the stronger definition of “commitment with 2-level sharing” of [KKK08, Definition 3], but not in their plain “commitment” specification. All works on VSS-based MPC protocols consider the committed value to be non- \perp , i.e., in \mathbb{S} : [BGW88; GRR98; CDD+99].

(VSS) Let us illustrate what could happen if it was not required that the values shared by corrupt dealers were well-defined before reconstruction //Notice that this requirement is not explicit in [HZC+22, p. 4]. Consider the scenario where some linear combination of secrets is opened to a corrupt learner, but to no honest learner yet. The adversary observes the output y . Then it manages to change the value shared by some corrupt dealers (verifiability prevents this); resulting in a modified output y' when the same linear combination is subsequently opened to a correct learner. Notice that the relaxed definition of AVSS, in [CCP21, Def 5.1], does not exclude such attack since it requires that a secret is committed only when *all* honest players have terminated `Share`. We observe that this relaxed definition nevertheless implies verifiability if those three additional conditions hold: (i) we require that players output Shamir shares in `Share`, (ii) $t < n/3$ players are corrupt (iii) reconstruction is done with the “on-line error correcting” algorithm of [Can95, p. 4.4.4]. On the other hand, for $t \geq n/3$, it is not excluded by their definition that only a subset of honest players output in `Share`, and that this subset is enough to perform reconstruction with the t dishonest players. We notified to them a pathological example allowing the adversary to manipulate the value reconstructed. It seems that such scenario cannot happen in any reasonable protocol where `Share` delivers a common view on commitments to subshares. There is also a relaxation of verifiability made in [CKLS02], named as *dual threshold*, and of which a state of the art implementation is [AVZ21]. One the one hand, it is needed that k honest players terminate the sharing, where k is a parameter, in order for a value to be committed. On the other hand, it is specified that if $< k - t$ honest players started reconstruction, then the adversary learns nothing on the secret. So it is again not excluded that the adversary manages so that only $k - t$ honest players output, be leaked the secret during the reconstruction, then manages to manipulate the value which will be reconstructed to honest players.

Finally, let us mention another use of the terminology (A)VSS. In [ZSV05, p. 4.2][BTA+19, p. 3.2][VAFB22, p. V], (A)VSS is not a (class of) message-passing protocols, but instead a mere list of local algorithms. So their specifications do not capture how or when a dealer becomes committed to a value towards players, nor how or when players are collectively able to `Open` this value.

G.2 Dual-threshold and high threshold AVSS

It was observed, since [CKLS02, §3.1], that AVSS can be customized to still guarantee some form of secrecy, beyond the correctness bound t . The notion introduced in [CKLS02, §3.1] is called *dual-threshold*, while a stronger notion appeared later, which we name as *high-threshold* (following [YXXM23]). The two terminologies are sometimes interchanged, so we first define them then list some conflicts.

Dual-threshold AVSS We follow the [AVZ21, definition 7]. Compared to AVSS, it brings in addition another threshold, named as $t \leq p < n - t$, which we may call the “priv-rec” threshold. Privacy is still guaranteed as long as at most $p - t$ honest players started to `Open`. Openability now requires $p + 1$ honest players. The definition of [CKLS02; KMS20, §3.1] is equivalent, up to a change in the numerology of the priv-rec threshold p . A limitation of *dual-threshold*, is that $t + 1$ corrupt players can nevertheless recover the secret. This apparent contradiction between t and p is nicely explained in [AVZ21, p3] //when queried to `Open`, the implementations of [CKLS02; KMS20; AVZ21] make players redistribute the secret from degree t into higher degree d , before revealing their share.

High-threshold AVSS is formalized in [YXXM23, §2.1]. A high threshold AVSS, with secrecy threshold d , guarantees secrecy against d corruptions //concretely, existing implementations are like our Share. They consist in doing a reliable broadcast of a degree d verifiably encrypted secret sharing, i.e., a PVSS. It is matched by [DXR21, Definition 5.3], and our $\mathcal{F}_{\text{P-AVSS}}$ //Concretely, our Share in APSS guarantees secrecy up to $d = t$ corruptions. The correctness and liveness threshold of Share is equal to the threshold of the reliable broadcast (RB) used.

We now observe some conflicts of terminology. The [DXR21, Definition 5.3] is advertized weaker than it is, since they call it only “dual threshold”. The special-purpose AVSS for zero sharing of [GDK22] is named as *high-threshold* (page 2), while, to the best of our understanding, its secrecy is guaranteed up to $t < n/3$ corruptions, by [GDK22, Definition 4 & Lemma 3].

G.3 Related ideal functionalities for (possibly proactive and/or asynchronous) VSS

The AVSS ideal functionality of [AAPP22; CP22]. It delivers plain Shamir shares of the stored secret, upon being allowed by the adversary, player by player. It does not deliver any other output, in particular, no commitment to shares, nor reconstruction of the secret. Plain Shamir shares do not enable robust reconstruction of the secret for $t \geq n/3$ corruptions. We refer to [CDN15] (p121: “Minimal Distance Decoding”) and to [Can95] (4.4.4 “on-line error correcting”) for this well-known result. So the AVSS specification of [AAPP22; CP22] is incompatible with $t \geq n/3$.

Notice that they consider an adversary \mathcal{A} which has the power to block forever the delivery of any output to any player. This power is actually standard in the classical UC model of [Can95]. This power is known as “delayed output”. In particular, a player may well never receive its share. In particular, the trivial protocol where players do nothing implements their functionality (the simulator just has to never allow the delivery of the output). For this reason, any non-trivial protocol implementing their functionality must come in addition with a proof of liveness.

The asynchronous dynamic PSS functionality of [YXXM23, B] It is specified to return a unique system of new shares, and a unique public set of Pedersen commitments to them. Their protocol implements it up to $t < n/3$ malicious corruptions plus $d - t$ extra passive corruptions, where $d \leq n - t - 1$. By contrast, our $\mathcal{F}_{\text{P-AVSS}}$ functionality does not return a unique public output, since APSS0 bypasses consensus. The return for this relaxation is that $\mathcal{F}_{\text{P-AVSS}}$ is implementable above $t < n/3$ malicious corruptions.

Another difference with the functionality of [YXXM23, B], is that the latter guarantees simultaneously: an output, correctness, and secrecy. So it is impossible to implement it above $t < n/2$. By contrast, $\mathcal{F}_{\text{P-AVSS}}$ separates liveness guarantees from correctness and privacy. As a result, it is implementable under *dishonest majority*, as the one of [CGG+20].

A last extra-generality of $\mathcal{F}_{\text{P-AVSS}}$ compared to [YXXM23, B], is that the latter does not handle scenarios where players would not start synchronously *Refresh* or *Open* (see also Appendix H.4 below). The reason is that in their model, both the players and $\mathcal{F}_{\text{P-AVSS}}$ receive the instructions to *Refresh* and *Open* from a trusted coordinator. This coordinator is assumed to always send these instructions simultaneously to all.

The synchronous distributed commitment functionality F_{COM} of [CDN15, p. 105] This functionality differs from VSS, in that an action of the dealer is necessary to open the secret. Excepted this difference, it is the closest to ours. Another difference is that F_{COM} is not specified to leak the secret to \mathcal{A} , in the scenario where the dealer and one isolated honest player (both being possibly equal) would start the *Open*. So a dummy protocol with F_{COM} can be UC emulated *only if* all dummy honest players are supposed to query *simultaneously* F_{COM} to *Open*. This restriction is made clear in a preliminary version⁸. The reason for this restriction is that the simulator is not able to simulate opening shares without this leakage.

The other difference is that, since F_{COM} is assumed to always deliver an output of *Open* if the committer is honest (and not leak the commitment until then), then it cannot be implemented under dishonest majority.

⁸In Contemporary Cryptology by D. Catalano et al, p62 “We require that all honest players agree to the fact that a commitment should be made because an implementation will require the active participation of all honest players”

The synchronous dynamic PSS functionality of [GKM+22, p5] they define an ideal functionality which delivers an output to an entity, denoted as “client”, which is assumed always honest. Hence, their definitions do not require simulatability of decryption shares. Nevertheless, it seems that they are able to simulate the opening shares of their protocol, as described in their Hybrid 11 p59.

The other difference is that their functionality does not capture refreshes between committees, nor does it capture which committee does the opening. The reason being that in their synchronous model, players are notified simultaneously when they have to Refresh or Open, and that Refreshes always terminates (within a fixed delay) thanks to honest majority

The synchronous dynamic PSS functionality of [HKMR22] The specification of their functionality involves a simulator, which moreover interacts simultaneously with an environment and an adversary. So it seems not to be in the UC model of [Can01].

H Existing PSS with $t < n/3$ resilience

We now survey PSS which do not guarantee simultaneously liveness, correctness and privacy beyond $t < n/3$ corruptions [ZSV05; CKLS02; SLL10; DM15; YXD22; VAFB22; YXXM23; HZC+22; GDK22]. Most of the PSS in this list are under asynchrony, some of them [SLL10; VAFB22] are under partial synchrony (Appendices H.1 and H.3.1). We also count as not resilient beyond $t < n/3$ the synchronous PSS “Opt-CHURP” [MZW+19; ADEO21]. The reason is that, when specialized to $n = 2t+1$, as soon as one player deviates, it must transition to Exp-CHURP-A [MZW+19] (Table 1) //notice that Exp-CHURP-A also serves as fallback for [YXD22], which has resilience to $t < n/4$ passive corruptions. Another exception is [DM15], which is synchronous but has information-theoretic security. In Appendix H.1 we remind partial synchrony, in Appendix H.2 we give an overview of the three main families of asynchronous PSS, in Appendix H.3 we analyze their complexities in more detail, and in Appendix H.4 we survey their termination mechanisms.

H.1 The Model with a hidden a priori upper bound Δ on δ (“Partial Synchrony”)

The model of [SLL10; VAFB22] assumes existence of an unknown finite time, denoted as GST, after which every message sent is delivered within a fixed public delay Δ . This is the classical model of partial synchrony [DLS88, 2.3 (3)]. In [DLS88, 2.3 (2)] they consider an alternative model of partial synchrony. There, the adversary commits on a *finite* upper bound Δ on δ at the beginning of every execution. Contrary to synchrony, Δ is hidden to the players. They show in [DLS88, p. 4.2] that this model has at least power as the classical one [DLS88, 2.3 (3)] in which GST is unknown. Concretely, the compilation is as follows. Start from a round-by-round protocol designed for the GST model. Increase regularly the duration of a round, by a public fixed parameter (see also [BCG22]). For instance, in [ACD+19], they double the duration of rounds at regular intervals. In conclusion, when the duration of rounds becomes at least as high as the hidden Δ , everything is as if GST had happened.

H.2 Overview of existing PSS with $t < n/3$ resilience

All existing PSS operating over an asynchronous network (a-PSS) or even partial synchrony, have threshold $t < n/3$. The reason is that their common blueprint is that players perform n AVSS in parallel, either to reshare their share or some randomness. Then they reach consensus on a subset of $t+1$ AVSS which terminated. Then they deduce their new shares from them. But, even under partial synchrony, it is trivial ([DLS88, Thm 4.4]) that *consensus is impossible to implement beyond $t < n/3$ corruptions, whatever the setup*. As a result, if such PSS protocols using consensus were executed with $f \geq n/3$ corruptions, then honest players could possibly output inconsistent new shares then erase their old ones, so the secret would be completely lost. A recent a-PSS, [YXXM23], guarantees privacy even up to a number d of corruptions, where $d < n$ is a parameter. On the other hand, its corruption bound for correctness is $t < n/3$, and its liveness requires at least $\max(d + 1, \lfloor 2n/3 \rfloor)$ passively corrupt or honest players, the $2n/3$ being due to its

use of consensus. By comparison in APSS, setting the privacy threshold to t yields a higher corruption bound for correctness: t , and a liveness bound of: $t+1$. So in our flagship setting of honest majority $t < n/2$, the latter is lower than the minimum $2n/3$ required by [YXXM23]. of an extra $n - 2t - 1$ passive corruptions in addition to the $t < n/3$ tolerated malicious corruptions. But this does not solve this consistency issue for $f \geq n/3$ corruptions.

The fastest possible AVSS consists in the reliable broadcast (RB) of a PVSS (of $B = O(n\gamma)$ bits). The state of the art RB takes 4δ and has $\text{RB}(B) = O(n|B| + \gamma n^2)$ communication complexity, see Appendix I.5.1. Then to reach consensus, a state of the art MVBA is [Guo et al, NDSS'22] [GLL+22], which takes 12δ in expectation and has $O(Bn^2)$ complexity. Summing-up, existing approaches for a-PSS cannot go much beyond a latency of 16δ , compared to 2δ only for APSS.

From a bird's eye, Refreshes in these works have the same high level structure: n AVSS in parallel, followed by consensus on a set of $n - t$ which terminated. At a high level point of view, they comes as the following three families, detailed in Appendices H.2.1 to H.2.3. There is one interesting exception [ZSV05], which we survey separately in Appendix H.3.4.

H.2.1 Masking shares with a 0-sharing. This technique is from [HJKY95], it is usable as such only in the static committee setting $\mathcal{P}_{[n]} = \mathcal{P}'_{[n']}$ with a mobile adversary. It is instantiated in [GDK22], which we now follow for the example. Each player deals an AVSS of a random sharing of 0 //it is instantiated in [GDK22] with the *dual threshold* AVSS of [AVZ21] (Appendix G.2). This is the reason why [GDK22, Definition 4 & Lemma 3] guarantee secrecy up to $t < n/3$ corruptions. Then, players reach consensus on a subset of AVSS which terminated //they suggest to instantiate the consensus either with MVBA (Appendix H.3.2 below) or ACS (Appendix H.3.3 below). Finally, players add their old share with the sum of the shares of 0 in the subset, in order to obtain their new share.

Notice that the technique of [HJKY95] is *adapted to the dynamic committee setting* in [SLL10], at the cost of more communication complexity.

Notice that the technique of [HJKY95] is *adapted to the dynamic committee setting* in the synchronous PSS [MZW+19], with yet another variation. There, old shares with threshold $t+1$ themselves come as degree- $2t$ polynomials: $B(i, y)$. Instead of directly handing their old shares to their counterpart in the new committee, as in [HJKY95; GDK22], old players give only to them an evaluation of them. So each player j in the new committee end up with a share of threshold only $2t$, which itself consists in a degree t polynomial: $B(x, j)$. Then players in the new committee generate a secret shared random degree $(t, 2t)$ bivariate polynomial Q such that $Q(0, 0) = 0$. Each new player j blinds its share by $Q(x, j)$. Then new players finally redistribute shares to each other to obtain shares with threshold $t + 1$ //so each share is equal to $B(j, y) + Q(j, y)$. Notice that this final degree-reduction step requires the participation of all $n = 2t + 1$ players, just as in MPC with passive security [CDN15, §3]. This is why [MZW+19] transitions to a costlier fallback protocol, Exp-CHURP-A [MZW+19], as soon as one corrupted player deviates. On the other hand, assuming $t < n/3$ corruptions, then the method of [MZW+19] terminates since $2t + 1$ players are assumed honest and so can carry-out the degree reduction. This is why its adaptation: [HZC+22] under *asynchrony* in the $t < n/3$ regime, does terminate.

H.2.2 Directly resharing one's share to new players in [CKLS02; YXXM23] players re-share their share with (asynchronous) verifiable secret sharing. This technique dates back from [DJ97] in the context of proactivity. Then players reach consensus on a subset of AVSS which terminated. Then they compute their new share by Lagrange linear combination of the received subshares in the subset. We refer to Section 2.2 for the background on the method of resharing, and to Appendix I.1 for a historical account.

H.2.3 Sharing a random mask to new players, then opening to them the masked secret in [VAFB22; YXXM23] they consider an equally classic method, in which both the entering and exiting committees $\mathcal{P}_{[n]}$ and $\mathcal{P}'_{[n']}$ generate a secret-shared random value r which is equal in both committees. Then, players of $\mathcal{P}_{[n]}$ open to $\mathcal{P}'_{[n']}$ the secret masked by the shared randomness: $s - r$. From this public value,

players of $\mathcal{P}'_{[n']}$ deduce their new shares of s by locally adding their share of r . This technique dates back from at least [GKM+22] in the context of proactivity.

H.3 More detailed analysis of their consensus and complexities

We now focus on the complexities of the various mechanisms by which the PSS of Appendix H.2 reach consensus on a common set of $n - t$ sharings. One can identify three families of mechanisms, which we now detail in Appendices H.3.1 to H.3.3.

H.3.1 Leader-based partially synchronous consensus As explained in Appendix H.1, partially synchronous protocols [SLL10; VAFB22] have latencies equal to multiples of a public fixed Δ . The parameter Δ is set in practice as an estimate of the eventual upper-bound on the network delay, it can possibly be publicly re-adjusted over time. In some fault-free executions, partially synchronous protocols may terminate within $O(\delta)$. But, unless branded as “asynchronous protocols”, in general executions they do not have liveness under full asynchrony.

The PSS [SLL10] uses the well-known leader-based consensus called “PBFT” [CL99] in order to reach consensus on a common set of terminated AVSS. In PBFT, the time-line of the execution is divided into intervals, denoted as “views”. Each is assigned to a player denoted as the *leader*, or the *primary*, of the view. In a view in which the leader is honest AND in which the actual delivery delay δ is small enough, in a sense to be precised, then players output within 4δ . There is a mechanism enabling players to go to the next view synchronously enough. A first line of mechanisms, which originate from [CL99], is triggered by players if they suspect the leader to be corrupt, e.g., if they do not receive messages from it after a fixed public timeout Δ . A recent implementation is [BCG22]. Another line of mechanisms [LA23] addresses the more particular specification to make players change view every fixed interval of time, say, 4Δ , without adaptivity to the good or bad behavior of the leader. In conclusion, PBFT terminates *if* the public parameter Δ was chosen such that, after an unknown time denoted as “GST”, the actual message delivery delay is $\delta < \Delta$. This assumption is known as *partial synchrony* [DLS88].

This idea of instantiating consensus in PSS by PBFT, was popularized again by the partially synchronous PSS [VAFB22] (S&P’22). Let us provide a quick overview of their Refresh protocol, denoted as “(dynamic) Reshare” (E, page 9), which enables to handle refreshed shares to a *new* committee. Our main conclusion is that their termination is more constrained than the one of PBFT. Even if a view has an honest leader and the network is fast enough, a single dishonest player may force to go to the next view. In more detail, “(dynamic) Reshare” makes a number of calls to a subroutine which is denoted as GeneratePolynomial. The latter goes through the following delays:

- Election of a leader (δ delay, assuming a threshold coin)
- Sending an encrypted sharing to it (δ delay)
- Performing PBFT-style consensus on a set of $t+1$ sharings. This takes 3δ delay in executions where *all* players are honest AND the network fast enough. It takes 4Δ more if the first leader is corrupt, etc. Furthermore, one single dishonest player which deviates from the protocol may prevent the consensus from terminating in this view. We dub such a deviation as a *failure*. They write: “Notice that if a malicious server r_j sends a valid share to the leader r_l and invalid shares to other servers, [...] the consensus might not terminate with this proposal since S [the commitment to r_j ’s shares] will not be accepted by enough servers.”. In such scenario, the leader is replaced after 3Δ in the first view, then 4Δ in subsequent views. So up to $t + 1$ instances of GeneratePolynomial can abort, or more if GST did not happen yet (because then an honest-but-slow leader may be replaced after 3Δ).
- Requesting by peer to peer the missing encrypted sharings (2δ delay)

Now, the high level structure of “(dynamic) Reshare” is:

- GeneratePolynomial
- then, a subprotocol denoted as “Recover”, one instance in parallel for every party which detected a failure. The latency of a Recover is at least equal to $2\delta +$ a GeneratePolynomial

All in all, assuming that in all instances of PBFT the second leader elected after GST is honest, and assuming that the instances of GeneratePolynomial launched in Recover go through a total of $t + 1$ failures (after which all corrupt players are ignored), then the latency of “dynamic reshare” after GST is of, roughly: one GeneratePolynomial (11Δ) then $(t+1)$ times Recover, each equal to $2\delta + 11\Delta$.

H.3.2 Consensus with external validity, a.k.a. VBA/VABA/MVBA [CKLS02; YXXM23; HZC+22; GDK22] propose to use consensus with external validity, a.k.a., VBA/VABA/MVBA [CKS05; AMS19; GLL+22] to reach consensus on a common set of terminated AVSS. In an MVBA, players start with inputs which can be tested as *valid* by a public efficiently computable predicate. An MVBA guarantees that all players output the same value, and that it is valid. In the ingredient, the inputs are a list of $t+1$ senders of AVSS. The predicate which is tested by each player is if it output in each of these AVSS. So this predicate is not publicly verifiable, contrary to the requirement of an MVBA. So it is not obvious how to use MVBA to implement the ingredient. The trick is elegantly explained by [YXXM23]. Players input their $t+1$ list \mathbb{L}_i as input to the MVBA. When a player \mathcal{P}_i receives a message from a player \mathcal{P}_j , in the execution of MVBA, and that this message contains a list \mathbb{L}_j which \mathcal{P}_i does not recognize as valid, \mathcal{P}_i puts the message in quarantine. The message remains in quarantine until \mathcal{P}_i recognizes the list \mathbb{L}_j as valid, i.e., until \mathcal{P}_i output in all AVSS in the list \mathbb{L}_j . By definition of AVSS, the list of an honest player will be ultimately recognized as valid by all honest players, so the MVBA can terminate.

H.3.3 Agreement on a common subset (ACS) A primitive which is tailored to provide consensus on a $n - t$ sized subset of instances of AVSS which terminated, was discovered in the context of asynchronous MPC in [BCG93, p. 3.1], under the name “agreement on a core set”. They propose an implementation tolerating $t < n/3$ malicious corruptions. The broad idea is that players run n instances of binary consensus (BBA) in parallel. Each player j inputs 1 in every instance i for which it received an output of the AVSS from i . However it is unclear if their implementation terminates. Namely, it is not specified what value, nor when, do players input in instances of consensus corresponding to indices i for which they did not output in the AVSS from i .

Then, the paper [BKR94, §4] put forth a definition which is the same the one of [BCG93], under the different name of “agreement on a common subset” (ACS). Then, they propose an implementation of ACS tolerating $t < n/3$. It has the same structure as the one [BCG93, p. 3.1]. The difference is that they make a precision, which makes the implementation rigorous. It consists the simple but clever trick that, as soon as it receives an output 1 in $n - t$ binary consensus instances, a player inputs 0 to all the t remaining instances. This trick was popularized later by [MXC+16].

The use of ACS as a means of consensus in PSS was initiated by [YXD22]. Precisely, they propose to use the ACS of [BKR94, §4] instantiated with the binary consensus (BBA) of [MMR15]. This same proposition is made in [GDK22, A].

H.3.4 The n -concurrent-leader-based-executions paradigm Interestingly, the PSS [ZSV05] runs n instances of a leader-based refresh in parallel. The common point with APSS is that, in instances lead by a corrupt leader, players can obtain incompatible shares. It uses (n, t) -replicated secret sharing (RSS), instead of Shamir sharing, so each player has $\binom{n}{t}$ additive shares. Since shares are sub-shared, it has communication complexity in $O(\exp(2n))$. RSS allows a slightly simpler baseline mechanism than the resharing-then-Lagrange linear combination (Appendix I.1) which we use. Their simpler mechanism (their Figure 6) is: consider any set of $\binom{n}{t}$ resharings, one for each RSS share, each being issued by any arbitrary player. Then, each player p constructs its new RSS shares as follows. For each Q a t -subset of players such that $p \notin Q$, the new share of p corresponding to Q is the sum of the sub-shares received corresponding to Q . This mechanism is slightly simpler than Shamir with Lagrange linear combination, since it involves only additions. Their main motivation for using RSS, which they stress at the bottom of page 266, is that it enables a “straightforward” non-interactive share recovery protocol (their Figure 5). They observe in 6.4.2 that they could have instead used, instead of RSS, any Shamir-based AVSS satisfying the *completeness* property (ACSS), recalled in Appendix G.1. [Recall that it is the guarantee that, as soon as one player outputs, then every player eventually outputs.]

But they motivate their choice by the fact that ACSS would have cost more consecutive interactions. As they explain in their section 7, their main focus are small values of n , for which the complexity gain brought by ACSS (from $O(\exp(2n))$ to polynomial) would not have counterbalanced the price of more interactions.

The protocol of [ZSV05] does not withstand more than $t < n/3$ malicious corruptions. The reason is that players consider the resharing of a share as completed, and ignore other resharings of this share, upon upon hearing from $2t + 1$ players which claim having received consistent subshares of this share (in the form of messages denoted as verified). In them, there are at least $t+1$ which are honest. The number $t+1$ is the minimum one to be able to reconstruct their missing shares to other isolated honest players (their Figure 4).

From a high level perspective, the MPC protocol [BHN10] operates with the same structure as [ZSV05] and ours, with parallel instances of MPC each driven by a leader. However they assume that a threshold decryption key is secret-shared between the players without addressing how to proactively refresh it, which is the purpose of the present work. Moreover, their liveness and correctness are conditioned to existence of an unknown *fixed* set of $t+1$ honest out of n players. Players in this set are assumed to remain honest until the end of the execution. This assumption does not hold in the context of proactivity, where corruptions change over time (in the mobile-PSS model) or across committees. Concretely, if their protocol were to be emulated in our setting of dynamic committees with honest majority, its liveness and correctness would require that, for the subset of the $P_i^1 \in \mathcal{P}_{[n_1]}^1$ which are honest, then all $P_i^e, \forall e$ must also be honest. Thanks to the chain of correctness mechanism in APSS0, i.e., the quorums of signatures replacing the NIKZ's and so on, APSS0 needs only existence of one single honest leader per epoch. Notice that the n -leaders-in-parallel paradigm is also used in the recent asynchronous consensus of [AMS19].

H.4 Existing specific scheduling mechanisms for PSS

In [CKLS02], the inputs `refresh-sig` and `shutoff-sig` are sent by a global clock. In their fixed committee setting, by `shutoff-sig` we mean the instruction to erase the memory related to the old epoch (but keep participating in the new epoch). Both ticks are sent at the same time. Upon receiving these inputs together, a player must immediately send a batch of resharing messages (concretely: perform the first step of an AVSS of its share), then erase all its memory related to the old epoch just after. Nothing prevents this global clock from ticking its inputs faster than the pace of refreshes. This is why in [CKLS02, p18], they condition liveness to executions in which the global clock waits that all messages of an epoch have been delivered, before ticking the next `refresh-sig` and `shutoff-sig`. In APSS0, if we consider the collectors as separate from committees, then we would also have liveness with such a global clock, under the same condition. The reason is that, if an old committee $\mathcal{P}_{[n_e]}^e$ is in the state of the *common set of shares*, as considered in Lemma 4, then players of $\mathcal{P}_{[n_e]}^e$ need to speak only once before erasing their memories. In that sense, we say that such an instantiation of APSS0 is YOSO (borrowing the terminology from [GHK+21]).

A similar same assumption of a global clock is made in [YXXM23], under the name of a “coordinator”. It is assumed to send instructions to refresh to all players simultaneously, and also to the functionality in the ideal execution. However the timing in their implementation is different from [CKLS02, p18], since they implicitly instruct players to terminate the (MVBA in) the previous Refresh, before they start the new Refresh and erase their old shares. This is the reason why their functionality (as ours) can be delayed by the adversary before it returns new shares. This implicit scheduling mechanism of [YXXM23], i.e., all players wait for the consensus to terminate before launching a new Refresh, is then one considered in nearly all asynchronous PSS so far. //By contrast, in the second model under attack in [ABKL22], players erase their memory and transition to the next epochs as soon as they *output* a new share in the consensus, but not necessarily wait to terminate the Refresh, i.e., the consensus. They point that [YXXM23] escapes their attack because there, players wait for the Refresh, i.e., the consensus, to finish. The exceptions are the aforementioned [CKLS02, p18], [ZSV05] detailed in Appendix H.3.4, and APSS.

Interestingly, we realized after completion of this work that [ZSV05] also had a termination mechanism, which is closer to ours. In [ZSV05], players cease to execute a Refresh, upon having been notified that: there exists a consistent set of $\binom{n}{t}$ resharings, such that for each of them, at least $t+1$ honest players have received correct sub-shares. //Such notification comes in the form of messages of the type denoted “certified”, one for each resharing. Each of them consisting of a batch of $2t + 1$ messages, of the type denoted “verified”, in which their issuers

CKLS02; GG06; DM15; BGG+20; Gro21; GHK+21; GHL22; YXXM23; HZC+22; CDGK22; HKMR22]. They ([DJ97]) state it for secrets in any space equipped with a linear secret sharing scheme. This generality is what enables the number of applications considered in Section 7, and also the implementation from Elgamal in Section 8. It turns out that the construction of publicly verifiable vectors of encrypted secret shares was first suggested in [GMW91, §3.2]. In the context of proactivity, it turns out that [BGG+20; Gro21] also considered the same baseline approach as APSS, namely, of *encrypted* resharing of secret shares. They require that every old committee publishes its $O(n^2)$ subshares on a public ledger. This is the limitation that APSS overcomes.

I.2 Hybrid communication networks offering initial rounds of synchrony

[FN09] show that, for any number $t < n$ of corruptions, Byzantine broadcast (BC) can be achieved in a network offering $(3t - n + 1)/2 + 5$ initial rounds of synchrony. So this is in general better than the latency-optimal deterministic BC protocol of [DS83], which requires $t+1$ rounds of synchrony. In [BHN10] it was evidenced for the first time that honest majority MPC with guaranteed output delivery (GOD) is feasible, in the threshold PKI model, in an asynchronous network offering one initial BC. Combined with any DKG in one synchronous BC ([FS01; Gro21]), this was the first MPC protocol from two rounds of BC-then-asynchrony in the PKI model. In [RU21] (8 Nov 221 version, p10, then §4.4.1) it is further observed that honest majority MPC with GOD is feasible in 2 synchronous rounds-then-*one* asynchronous round. It holds in the PKI model without any random string setup, thanks to a simple compilation of [BJMS20] (moving their first round in the PKI, and making their last round asynchronous). The information-theoretic setting is studied in [PR18]. They show that perfectly secure AVSS under $t < n/3$ corruptions is feasible in a network offering only one synchronous round, not even a BC. They show that perfectly secure MPC, without input deprivation, is feasible in a network offering 3 initial synchronous rounds, not even a BC. By contrast they show that, it is unimplementable in a network offering only 2 initial synchronous rounds, surprisingly *even if* these two rounds provide BC. In [Cho20a], it is announced that one initial round of synchrony is enough to achieve the first almost-surely terminating asynchronous binary consensus with strong validity, such that it has both $t < n/3$ resilience and constant-round expected latency. In [CH20] it is shown that for $n = 4$ and $t = 1$, then perfectly secure AMPC, i.e., with input deprivation, is feasible provided only 2 initial synchronous rounds of BC-then-asynchrony (which is not enough for MPC [PR18]). They also provide a computationally secure MPC, without input deprivation, from symmetric primitives only. So this improves over the PKI tools in [BHN10], at the price of a lower threshold. In [UR22] it is proven that, even in the computational setting, GOD MPC under honest majority is impossible in a network offering only 1 initial round of synchronous BC, without further setup.

I.3 Notions related to bilateral-binding: committing, robust, undeniable

The notion of *undeniable encryption*, of Takahashi and Zaverucha [TZ21, Definition 10], specifies like Definition 3 that the sender cannot explain the ciphertext by a different plaintext than the one decrypted by the receiver. But their definition holds only if the receiver generates its private key honestly and away from the sender. If not in this scenario, then the probability of mismatch between the plaintext and the decryption, as considered in Definition 3, is potentially high.

The Complete Robustness “CROB” notion introduced in [Farshim et al, PKC’13] [FLPQ13] is orthogonal to Definition 3. They rule out decryptions (possibly equal) under *two distincts secret keys*.

[BH22] consider symmetric-key analogues of CROB, denoted as *committing* encryption. They strengthen it, by also ruling-out different decryptions of a given ciphertext under the *same* key (denoted as scenario D). They strengthen it in another direction, by also ruling-out different plaintexts explaining the same ciphertext under the *same* key (denoted as scenario E). However, they do not define explicitly the analogue of the bilateral binding guarantee of Definition 3. In their symmetric context, this bilateral binding guarantee would translate into: for a given ciphertext, exhibiting a plaintext different from a decryption, under the same key.

I.4 Complexities of Broadcast (BC)

BC [FLL21, Definition 1] involves a *sender* S and a set of receivers \mathcal{R} , typically a committee. It requires that: (*Termination*) all receivers eventually output; (*Consistency*): the same value; (*Validity*): which is furthermore equal to the input of S if it is honest. It is trivial that BC cannot be implemented without a *synchronous* network, whatever the setup. [Otherwise, players can never tell apart a corrupt and silent sender, from a honest sender of which the message has been delayed so far].

I.4.1 Under $t < n/2$ In [GP21] they build a compiler which takes as input a BC under $t < n/2$, for small inputs of size γ . It outputs a BC for inputs of large size B , with communication complexity amortized over n . They achieve a complexity of $O(nB + n|BC(\gamma)| + \gamma n^3)$, for B at least as large as $\Omega(n^3 + \gamma n^2)$, which is the case in our context. But actually, the techniques of [NRS+20] seem to enable better, even for any $t < (1 - \epsilon)n$ (see Appendix I.4.2).

The BC of [ANRX21, Fig. 6] has an optimal honest-sender latency of $\delta + \Delta$. However in the worst-case, it may take 4Δ plus a black box consensus (called “MVBA”). They also provide, in Fig. 9, an adaptation under the more conservative model where players are not anymore assumed to start synchronously. It takes $\delta + 1.5\Delta$ in the honest-sender case, whereas in the worst-case it takes 8.5Δ plus a black box MVBA. We now turn to the instantiation of the required MVBA. The MVBA of [ADD+19, §4] has $O(n^2)$ communication complexity and an expected latency of 10Δ (for a static adversary, vs 16Δ for a strongly rushing one), provided a threshold setup (both for leader sortition and for short threshold signatures). Somehow concurrently, [SBKN21, §8]⁹, present a constant-round MVBA under $t < n/2$ with $O(Bn^2 + \gamma n^3)$ communication complexity, for inputs of size B .

I.4.2 Under $t < n$ The Dolev-Strong BC [DS83, Thm. 3] has communication $BC(1) = O(n^3 + n^2\gamma)$ bits, according to [MR20, Table 1]. The Dolev-Strong BC [DS83, Thm. 3] terminates in $(t + 1)\Delta$.

The BC of [FN09] requires $(3t - n + 1)/2 + 5$ initial rounds of synchrony.

[WXSD20] terminates in the worst-case in an expected $O(\left(\frac{n}{n-f}\right)^2)$ rounds (and, in the honest-sender case, in $O(\lceil \frac{n}{n-f} \rceil + \lfloor \frac{n}{n-f} \rfloor)$), as credited to [ANRX21] by [AFRT22]).

[WXDS20] is both constant round and resilient against an adaptive adversary. The protocols [DS83; AFRT22] are deterministic, so they inherently tolerate an adaptive adversary, but determinism makes them hit by the $(t+1)\Delta$ latency lower bound of [DS83, Thm. 2].

In [GP21] they build a compiler, with 5Δ latency overhead, which takes as input a BC under any $t < n$, for small inputs of size γ . It outputs an amortized BC for inputs of large size B . They achieve a complexity of $O(nB + n|BC(n\gamma)| + n^2BC(n \log n))$, for B at least as large as $\Omega(n^5 \log n + kn^4 \log n)$. But actually the technique of [NRS+20] enables much better if $t < (1 - \epsilon)n$. They achieve an amortized communication complexity of $O(nB + |BC(k)| + \gamma n^2 + n^3)$, which shows-up as soon as $B = \Omega(n^2 + \gamma n)$.

I.4.3 Instantiating BC from Ethereum The security of the proof of stake protocol for Ethereum is conditioned on honesty of two-thirds of the stakeholders [PAT22]. The security of Bitcoin is conditioned to honesty of a large majority of miners [GRR22]. The current publication delay on Ethereum is the sum of the first confirmation time, which is 2 minutes in average periods, plus the extra delay to obtain sufficiently many confirmations, i.e., of typically 20 additional blocks, which is 5 minutes in average periods.

I.5 Complexities of Reliable Broadcast (RB)

The definition of reliable broadcast (RB) was introduced by [Bra87]. It is a protocol between a sender S and a set of receivers \mathcal{R} , which guarantees that

⁹they write “In addition, to the best of our knowledge, no MVBA protocol have been proposed in the synchronous communication model for $t < n/2$ case”

Liveness if S is honest, then all \mathcal{R} eventually output;
Validity if S is honest and has input s , then only s can be output;
Consensus no two honest receivers output different values;
Totality if one honest receiver outputs, then all honest receivers eventually output.

The difference with BC is that it relaxes the unconditional requirement to have (eventual) *termination*, even if the sender is corrupt, by the conditions of liveness and totality. The terminology *totality* was coined by [CKPS01]. Totality is equivalent to: all honest players eventually output, or, no honest player ever outputs. Notice that, compared to the presentation of [CKPS01], we separated their “validity” in two parts: our liveness, and our validity. We consider algorithms for RB allowing digital signatures. They are traditionally dubbed as *authenticated algorithms*. We survey only RB protocols up to $t < n/2$, since beyond we are not aware of any RB which would not be a BC.

I.5.1 Under $t < n/3$ We consider here algorithms for RB in a fully asynchronous network. Notice that BC is trivially impossible even under partial synchrony, whatever the setup. The fastest known fully asynchronous RB is the one of [ANRX21, Figure 1]. It terminates in 2δ rounds when the sender is honest, it has complexity $O(Bn + \gamma n^2)$ for a B bits message. If the sender is not honest, and if some player terminates, then all other players terminate after at most δ . Notice that the RB of Bracha takes longer but does not use signed messages. The state of the art complexity of fully asynchronous RB is $O(Bn + \gamma n^2)$, for messages of B bits. It is matched in [DXR21; ADD+22] in 4δ , the latter having furthermore a balanced communication load.

I.5.2 Under $t < n/2$ The RB of [GPS19, §5] operates on a network providing only two initial rounds of synchrony, then becomes fully asynchronous. It has communication complexity $O(\gamma n^2)$. In [MR20, v2, Table 1] they also credit the unpublished [MCK20] for a RB under $t < n/2$ with the same complexity.

I.6 Network delays

The 100ms intercontinental message delay is reported in [AWS21; Sys21][AAPP22, p5].

I.7 Parallel executions of protocols with probabilistic termination

It was observed by Ben Or and el Yaniv [BE03] that if n protocols are run in parallel and that each of them terminates before k rounds up to probability p^{-k} , for some $0 < p < 1$, then the expected time at which the last of them terminates is $O(\log(n))$. In [CCGZ21, A] they show that the expectation is greater than $\frac{1}{-\log(1-p)}(\log(n) + \gamma)$, where γ is the Euler constant. The ACS protocol described in Appendix H.3.3 requires n parallel instances of ABA to terminate, hence, terminates in an expected $O(\log(n))$ steps.

References for the Appendices.

- [AAPP22] I. Abraham, G. Asharov, S. Patil, and A. Patra. “Asymptotically Free Broadcast in Constant Expected Time via Packed VSS”. In: *TCC*. 2022.
- [ABKL22] A. B. Alexandru, E. Blum, J. Katz, and J. Loss. “State Machine Replication under Changing Network Conditions”. In: *ASIACRYPT*. 2022.
- [ACD+19] I. Abraham, T.-H. H. Chan, D. Dolev, K. Nayak, R. Pass, L. Ren, and E. Shi. “Communication Complexity of Byzantine Agreement, Revisited”. In: *Proceedings of the 2019 ACM PODC*. 2019.
- [ADD+22] N. Alhaddad, S. Das, S. Duan, L. Ren, M. Varia, Z. Xiang, and H. Zhang. “Balanced Byzantine Reliable Broadcast with Near-Optimal Communication and Improved Computation”. In: *PODC*. 2022.
- [ADEO21] D. F. Aranha, A. Dalskov, D. Escudero, and C. Orlandi. “Improved Threshold Signatures, Proactive Secret Sharing, and Input Certification from LSS Isomorphisms”. In: *LATINCRYPT*. 2021.

- [ADN06] J. F. Almansa, I. Damgård, and J. B. Nielsen. “Simplified Threshold RSA with Adaptive and Proactive Security”. In: *EUROCRYPT*. 2006.
- [AFRT22] T. Albouy, D. Frey, M. Raynal, and F. Taïani. “Good-Case Early-Stopping Latency of Synchronous Byzantine Reliable Broadcast: The Deterministic Case”. In: *DISC*. 2022.
- [AGY95] N. Alon, Z. Galil, and M. Yung. “Efficient dynamic-resharing “verifiable secret sharing” against mobile adversary”. In: *ESA*. 1995.
- [AMN+20] I. Abraham, D. Malkhi, K. Nayak, L. Ren, and M. Yin. “Sync HotStuff: Simple and Practical Synchronous State Machine Replication”. In: *IEEE S&P*. 2020.
- [AMS19] I. Abraham, D. Malkhi, and A. Spiegelman. “Validated asynchronous byzantine agreement with optimal resilience and asymptotically optimal time and word communication”. In: (2019).
- [ANRX21] I. Abraham, K. Nayak, L. Ren, and Z. Xiang. “Good-Case Latency of Byzantine Broadcast: A Complete Categorization”. In: *PODC*. 2021.
- [AVZ21] N. Alhaddad, M. Varia, and H. Zhang. “High-Threshold AVSS with Optimal Communication Complexity”. In: *FC*. 2021.
- [AWS21] A. AWS. *AWS Regions and Zones*. 2021. URL: <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html>.
- [BCG22] M. Bravo, G. V. Chockler, and A. Gotsman. “Making Byzantine consensus live”. In: *Distributed Comput.* (2022).
- [BCG93] M. Ben-Or, R. Canetti, and O. Goldreich. “Asynchronous Secure Computation”. In: *STOC*. 1993.
- [BDK13] M. Backes, A. Datta, and A. Kate. “Asynchronous Computational VSS with Reduced Communication Complexity”. In: *CT-RSA*. 2013.
- [BDO22] L. Braun, I. Damgård, and C. Orlandi. “Secure Multiparty Computation from Threshold Encryption based on Class Groups”. In: *ePrint* (2022).
- [BDOZ11] R. Bendlin, I. Damgård, C. Orlandi, and S. Zakarias. “Semi-homomorphic Encryption and Multiparty Computation”. In: *EUROCRYPT*. 2011.
- [BE03] M. Ben-Or and R. El-Yaniv. “Resilient-optimal interactive consistency in constant time”. In: *Distributed Comput.* (2003).
- [BELO14] J. Baron, K. El Defrawy, J. Lampkins, and R. Ostrovsky. “How to Withstand Mobile Virus Attacks, Revisited”. In: *PODC*. 2014.
- [BGW88] M. Ben-Or, S. Goldwasser, and A. Wigderson. “Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation”. In: *STOC*. 1988.
- [BH08] Z. Beerliová-Trubíniová and M. Hirt. “Perfectly-Secure MPC with Linear Communication Complexity”. In: *TCC*. 2008.
- [BH22] M. Bellare and V. T. Hoang. “Efficient Schemes for Committing Authenticated Encryption”. In: *EUROCRYPT*. 2022.
- [BHN10] Z. Beerliová-Trubíniová, M. Hirt, and J. B. Nielsen. “On the theoretical gap between synchronous and asynchronous MPC protocols”. In: *PODC*. 2010.
- [BJMS20] S. Badrinarayanan, A. Jain, N. Manohar, and A. Sahai. “Secure MPC: Laziness Leads to GOD”. In: *ASIACRYPT*. 2020.
- [BKP11] M. Backes, A. Kate, and A. Patra. “Computational Verifiable Secret Sharing Revisited”. In: *ASIACRYPT*. 2011.
- [BKR94] M. Ben-Or, B. Kelmer, and T. Rabin. “Asynchronous Secure Computations with Optimal Resilience (Extended Abstract)”. In: *PODC*. 1994.
- [Bra87] G. Bracha. “Asynchronous Byzantine agreement protocols”. In: *Information and Computation* (1987).
- [BS20] D. Boneh and V. Shoup. *A Graduate Course in Applied Cryptography*. Version 0.5 Jan 2020. 2020.
- [Can95] R. Canetti. “Studies in Secure Multiparty Computation and Applications”. PhD thesis. 1995.
- [CCGZ21] R. Cohen, S. Coretti, J. A. Garay, and V. Zikas. “Round-Preserving Parallel Composition of Probabilistic Termination Cryptographic Protocols”. In: *J. Cryptol.* (2021).
- [CCN21] P. Chatzigiannis, K. Chalkias, and V. Nikolaenko. *Homomorphic decryption in blockchains via compressed discrete-log lookup tables*. CBT workshop 2021 (ESORICS). 2021.

- [CCP21] A. Chandramouli, A. Choudhury, and A. Patra. “A Survey on Perfectly Secure Verifiable Secret-sharing”. In: *ACM Computing Surveys (CSUR)* (2021). retrieved eprint version of 2022-02-04.
- [CEK+16] J. Camenisch, R. R. Enderlein, S. Krenn, R. Küsters, and D. Rausch. “Universal Composition with Responsive Environments”. In: *ASIACRYPT*. 2016.
- [CGHZ16] S. Coretti, J. Garay, M. Hirt, and V. Zikas. “Constant-Round Asynchronous Multi-Party Computation Based on One-Way Functions”. In: *ASIACRYPT*. 2016.
- [CGMA85] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. “Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults (Extended Abstract)”. In: *FOCS*. 1985.
- [Cho20] A. Choudhury. “Brief Announcement: Almost-Surely Terminating Asynchronous Byzantine Agreement Protocols with a Constant Expected Running Time”. In: *PODC*. 2020.
- [CKPS01] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup. “Secure and Efficient Asynchronous Broadcast Protocols”. In: *CRYPTO*. 2001.
- [CL99] M. Castro and B. Liskov. “Practical byzantine fault tolerance”. In: *OsDI*. 1999.
- [CLO+13] A. Choudhury, J. Loftus, E. Orsini, A. Patra, and N. P. Smart. “Between a Rock and a Hard Place: Interpolating between MPC and FHE”. In: *ASIACRYPT*. 2013.
- [CMTA20] Y. Chen, X. Ma, C. Tang, and M. H. Au. “PGC: Decentralized Confidential Payment System with Auditability”. In: *ESORICS*. 2020.
- [CP22] A. Choudhury and A. Patra. *On the Communication Efficiency of Statistically-Secure Asynchronous MPC with Optimal Resilience*. Iacr ePrint 2022/913. long version of ICITS 2009 and INDOCRYPT 2020. 2022.
- [CR93] R. Canetti and T. Rabin. “Fast Asynchronous Byzantine Agreement with Optimal Resilience”. In: *STOC '93*. 1993.
- [DJ97] Y. Desmedt and S. Jajodia. *Redistributing secret shares to new access structures and its applications*. Tech Report, George Mason U. July 1997.
- [DM15] Y. Desmedt and K. Morozov. “Parity Check based redistribution of secret shares”. In: *ISIT*. 2015.
- [DMR+21] I. Damgård, B. Magri, D. Ravi, L. Siniscalchi, and S. Yakoubov. “Broadcast-optimal two round MPC with an honest majority”. In: *CRYPTO*. 2021.
- [DS83] D. Dolev and H. R. Strong. “Authenticated Algorithms for Byzantine Agreement”. In: *Siam Journal on Computing* (1983).
- [DXR21] S. Das, Z. Xiang, and L. Ren. “Asynchronous Data Dissemination and Its Applications”. In: *CCS*. 2021.
- [FGMY97] Y. Frankel, P. Gemmel, P. D. MacKenzie, and M. Yung. “Optimal-resilience proactive public-key cryptosystems”. In: *FOCS*. 1997.
- [FLL21] M. Fitzi, C.-D. Liu-Zhang, and J. Loss. “A New Way to Achieve Round-Efficient Byzantine Agreement”. In: *PODC*. 2021.
- [FLPQ13] P. Farshim, B. Libert, K. G. Paterson, and E. A. Quaglia. “Robust Encryption, Revisited”. In: *PKC*. 2013.
- [FN09] M. Fitzi and J. B. Nielsen. “On the Number of Synchronous Rounds Sufficient for Authenticated Byzantine Agreement”. In: *DISC*. 2009.
- [FS01] P. Fouque and J. Stern. “One Round Threshold Discrete-Log Key Generation without Private Channels”. In: *PKC*. 2001.
- [GG06] V. Gupta and K. Gopinath. “An extended verifiable secret redistribution protocol for archival systems”. In: *ARES*. 2006.
- [GJM+21] K. Gurkan, P. Jovanovic, M. Maller, S. Meiklejohn, G. Stern, and A. Tomescu. “Aggregatable Distributed Key Generation”. In: *EUROCRYPT*. 2021.
- [GLL+22] B. Guo, Y. Lu, Z. Lu, Q. Tang, J. Xu, and Z. Zhang. “Speeding Dumbo: Pushing Asynchronous BFT Closer to Practice”. In: *NDSS*. 2022.
- [GLS15] S. Dov Gordon, F.-H. Liu, and E. Shi. “Constant-Round MPC with Fairness and Guarantee of Output Delivery”. In: *CRYPTO*. 2015.
- [GMW91] O. Goldreich, S. Micali, and A. Wigderson. “Proofs That Yield Nothing but Their Validity or all languages in NP have zero-knowledge proof systems”. In: *J. ACM* (1991).

- [GO07] J. Groth and R. Ostrovsky. “Cryptography in the Multi-string Model”. In: *CRYPTO*. 2007.
- [GOS06] J. Groth, R. Ostrovsky, and A. Sahai. “Perfect Non-interactive Zero Knowledge for NP”. In: *EUROCRYPT*. 2006.
- [GPV08] C. Gentry, C. Peikert, and V. Vaikuntanathan. “Trapdoors for Hard Lattices and New Cryptographic Constructions”. In: *STOC*. 2008.
- [Gro21] J. Groth. *Non-interactive distributed key generation and key resharing*. ePrint 2021/339. 2021.
- [GRR22] P. Gazi, L. Ren, and A. Russell. “Practical Settlement Bounds for Proof-of-Work Blockchains”. In: *CCS*. 2022.
- [GRR98] R. Gennaro, M. O. Rabin, and T. Rabin. “Simplified VSS and Fast-Track Multiparty Computations with Applications to Threshold Cryptography”. In: *PODC*. 1998.
- [HHPV21] S. Halevi, C. Hazay, A. Polychroniadou, and M. Venkatasubramanian. “Round-Optimal Secure Multi-Party Computation”. In: *J. Crypto* (2021).
- [KMTZ11] J. Katz, U. Maurer, B. Tackmann, and V. Zikas. “Universally Composable Synchronous Computation”. In: *TCC*. 2011.
- [LA23] A. Lewis-Pye and I. Abraham. “Fever: Optimal Responsive View Synchronisation”. In: (2023). eprint: [Arxiv2301.09881](https://arxiv.org/abs/2301.09881).
- [LLM+20] C. Liu-Zhang, J. Loss, U. Maurer, T. Moran, and D. Tschudi. “MPC with Synchronous Security and Asynchronous Responsiveness”. In: *ASIACRYPT*. 2020.
- [LNP22] V. Lyubashevsky, N. K. Nguyen, and M. Plancon. “Lattice-Based Zero-Knowledge Proofs and Applications: Shorter, Simpler, and More General”. In: *CRYPTO*. 2022.
- [MMR15] A. Mostéfaoui, H. Moumen, and M. Raynal. “Signature-Free Asynchronous Binary Byzantine Consensus with $t \leq n/3$, $O(n^2)$ Messages, and $O(1)$ Expected Time”. In: *J. ACM* (2015).
- [MR20] A. Momose and L. Ren. “v2 2020-10-13 of Optimal Communication Complexity of Byzantine Consensus under Honest Majority”. In: *arxiv 2007.13175* (2020).
- [MXC+16] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song. “The Honey Badger of BFT Protocols”. In: *CCS*. 2016.
- [NRS+20] K. Nayak, L. Ren, E. Shi, N. H. Vaidya, and Z. Xiang. “Improved Extension Protocols for Byzantine Broadcast and Agreement”. In: *DISC*. 2020.
- [NS22] N. K. Nguyen and G. Seiler. “Practical Sublinear Proofs for R1CS from Lattices”. In: *CRYPTO*. 2022.
- [PAT22] U. Pavloff, Y. Amoussou-Guenou, and S. Tucci-Piergiovanni. *Ethereum Proof-of-Stake under Scrutiny*. 2022. URL: <https://arxiv.org/abs/2210.16070>.
- [Ped92] T. P. Pedersen. “Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing”. In: *CRYPTO*. 1992.
- [PR18] A. Patra and D. Ravi. “On the Power of Hybrid Networks in Multi-Party Computation”. In: *IEEE Trans. Inf. Theory* (2018).
- [RU21] M. Rambaud and A. Urban. *Almost-Asynchronous MPC under Honest Majority, Revisited*. ePrint 2021/503. 2021.
- [Sch99] B. Schoenmakers. “A Simple Publicly Verifiable Secret Sharing Scheme and its Application to Electronic Voting”. In: *CRYPTO*. 1999.
- [SLL10] D. Schultz, B. Liskov, and M. Liskov. “MPSS: Mobile Proactive Secret Sharing”. In: *ACM Trans. Inf. Syst. Secur.* (2010).
- [Sys21] A. Systems. *AWS Inter Region Latency*. 2021. URL: [%5Curl%7Bhttps://docs.aviatrix.com/HowTos/inter_region_latency.html%7D](https://docs.aviatrix.com/HowTos/inter_region_latency.html%7D).
- [TLC+22] F. Tang, G. Ling, C. Cai, J. Shan, X. Liu, P. Tang, and W. Qiu. *Solving Small Exponential ECDLP in EC-based Additively Homomorphic Encryption and Applications*. ePrint 2022/1573. 2022.
- [UR22] A. Urban and M. Rambaud. *Share & Shrink: Ad-Hoc Threshold FHE with Short Ciphertexts and its Application to Almost-Asynchronous MPC*. ePrint 2022/378. 2022.
- [WWW02] T. M. Wong, C. Wang, and J. M. Wing. “Verifiable Secret Redistribution for Archive Systems”. In: *IEEE SISW*. 2002.
- [WXDS20] J. Wan, H. Xiao, S. Devadas, and E. Shi. “Round-Efficient Byzantine Broadcast Under Strongly Adaptive and Majority Corruptions”. In: *TCC*. 2020.

- [YLF+22] T. Yurek, L. Luo, J. Fairoze, A. Kate, and A. K. Miller. “hbACSS: How to Robustly Share Many Secrets”. In: *NDSS*. 2022.
- [YLH+20] R. Yang, J. Lai, Z. Huang, M. H. Au, Q. Xu, and W. Susilo. “Possibility and Impossibility Results for Receiver Selective Opening Secure PKE in the Multi-challenge Setting”. In: *ASIACRYPT*. 2020.