State Machine Replication under Changing Network Conditions

Andreea B. Alexandru[1], Erica Blum[1], Jonathan Katz[1], and Julian Loss[2]⋆

[1] University of Maryland, College Park
aandreea@umd.edu, erblum@umd.edu, jkatz2@gmail.com
[2] CISPA Helmholtz Center for Information Security
lossjulian@gmail.com

Abstract. Protocols for state machine replication (SMR) are typically differently designed for synchronous or asynchronous networks, with a lower corruption threshold in the latter case. Recent *network-agnostic* protocols are secure when run in either a synchronous or an asynchronous network. We propose two new constructions of network-agnostic SMR protocols that improve on existing protocols in terms of either adversarial model or communication complexity:

1. an adaptively secure protocol with optimal corruption thresholds and quadratic amortized communication complexity per transaction;
2. a statically secure protocol with near-optimal corruption thresholds and linear amortized communication complexity per transaction.

We further explore efficient SMR protocols run in a network that may change between synchronous and asynchronous arbitrarily often; parties can be uncorrupted (as in the proactive model), and the protocol should remain secure as long as the appropriate corruption thresholds are always maintained. We show that proactively secure SMR using threshold cryptography is impossible without some form of synchronization between the parties. Motivated by this negative result, we consider a model where the adversary is limited in the total number of parties it can corrupt over the duration of the protocol and show, in this setting, that our SMR protocols remain secure under arbitrarily changing network conditions.

1 Introduction

Protocols for *state-machine replication (SMR)* allow a set of parties $P_1, \ldots, P_n$ to agree on a continuously growing, ordered log of transactions. SMR protocols enable the evolving state of a distributed system to be replicated across multiple parties, even when some of them are malicious. SMR lies at the core of many distributed applications and has recently received a considerable amount of attention in the context of blockchain protocols. Most of the literature focuses on protocols that are secure in either the *synchronous* or the *asynchronous* model. Protocols for the synchronous model can tolerate any $t < n$ corrupted parties,

⋆ Part of this work was done while the author was a postdoctoral researcher at the University of Maryland, College Park and at the Carnegie Mellon University.

but may fail if the synchrony assumption is violated. On the other hand, asynchronous protocols are secure under arbitrary network conditions, but do not exist when $t \geq n/3$.

Recent work of Blum, Katz, and Loss [6] introduced the *network-agnostic* model in which a single protocol is required to be secure, for different corruption thresholds, regardless of whether it is run in a synchronous or asynchronous network. In subsequent work [7], they show that for any thresholds $t_a \leq t_s$ with $2t_s + t_a < n$, there is an SMR protocol that tolerates t_a corrupted parties if the network is asynchronous and simultaneously tolerates t_s corrupted parties if the network is synchronous. A major benefit of network-agnostic protocols over classical ones is that t_a, t_s can be chosen arbitrarily subject to the above constraints. This allows a protocol designer to flexibly choose t_a, t_s so as to minimize the probability of failure based on assumed properties of the environment.

Although network-agnostic protocols have recently received significant attention [6,8,7,29,4,17], several open questions regarding network-agnostic SMR remain. For one, existing results are primarily concerned with feasibility rather than efficiency; this is especially true when considering protocols secure against an adaptive adversary who can choose which parties to corrupt during the execution of the protocol. Perhaps the most significant limitation of prior work is that it either requires the network to be synchronous for the lifetime of the protocol, or else guarantees security only if the attacker never exceeds the corruption threshold of t_a. Providing a more elegant treatment of networks that can change arbitrarily often between synchronous and asynchronous was left as an explicit open question in prior work.

1.1 Challenges and State-of-the-Art

We begin with a brief overview of network-agnostic SMR, and then explain how existing solutions (do not) deal with the issues raised above.

Network-agnostic SMR. The goal of an SMR protocol is to impose order on transactions that arrive in parties' buffers in an arbitrary fashion. An SMR protocol must ensure *consistency*, which means that all parties agree on the order in which transactions are committed to some log, and *liveness*, which means that any transactions in the buffers of honest parties are eventually appended to the log. SMR is significantly more challenging than the related problem of Byzantine agreement, where parties agree only on a single value.

A network-agnostic SMR protocol must remain secure if the network is synchronous and there are at most t_s corruptions, or if the network is asynchronous and there are at most t_a corruptions. As a key building block for SMR in this setting, Blum *et al.* [7] introduced a novel protocol for asynchronous common subset (ACS) that allows parties to agree on a subset of $n - t_a$ inputs in the presence of t_a corrupted parties in an asynchronous network. Their protocol has the special property that if all honest parties supply *the same input* B to the protocol, then honest parties include B in their output even when t_s parties are corrupted. This facilitates the following strategy: parties first attempt to agree on an input B using a synchronous protocol. If the network is synchronous, this step will succeed even in the presence of t_s corrupted parties. Thus, parties input the same input B to ACS which outputs this block even against t_s corrupted parties. On the other hand, if the network is asynchronous, the t_a security of ACS ensures that all parties can agree on B without relying on the synchronous protocol part. Designing ACS with both the above properties is challenging.

Problems with existing solutions. Blum *et al.* [7] present two SMR protocols, Tardigrade and Upgrade. Tardigrade is secure against an adaptive adversary and requires $O(n^4)$ bits of communication for $O(n)$ transactions. Upgrade gives a more efficient alternative against a static adversary that requires only $O(n^3)$ bits of communication for $O(n^2)$ transactions. However, Upgrade relies on random subcommittees to execute the most expensive steps of the protocol. It is well-known that running protocols in this way requires very large committees (several hundreds of parties) in order to provide meaningful corruption bounds. This arguably offsets the communication improvements made by Upgrade, as it only offers an asymptotic improvement if the total number of parties in the system is in the order of hundreds of thousands. Moreover, their protocols solely consider non-switching networks, i.e. the network is either synchronous or asynchronous for the whole duration of the protocol. We are interested in a more flexible model that tolerates arbitrary transitions between synchronous and asynchronous behaviors of the network under a mobile adaptive adversary. However, if, at any point in the lifetime of the protocol, the adversary surpasses t_a corrupted parties, then these protocols might remain forever insecure in an asynchronous network.

1.2 Our Contributions

We study protocols in a more realistic model where network conditions can arbitrarily change over time, but parties can also recover from corruptions over time. In this manner, it becomes possible to tolerate optimal corruption thresholds in a network with fluctuating synchronicity properties.

Modeling key exposure. Modeling parties that are temporarily corrupted (sometimes referred to in the literature as *transient faults*) turns out to be non-trivial in the context of digital signatures. To model the process of uncorruption, we endow parties with a mechanism to forcibly "flush out" the adversary. This could be achieved, for example, by having parties restart their computer in safe mode at the onset of a new protocol epoch. The (adaptive) adversary can then choose to re-corrupt those parties or new ones. However, without additional measures in place, the internal state of those parties remains exposed to the adversary, which can forge signatures on behalf of exposed parties despite no longer having full control over them. Proactive secret sharing is the main technique to refresh parties' keys for threshold signatures and related primitives commonly used in randomized SMR protocols. Unfortunately, we disprove any secure proactive secret sharing protocol in the pure asynchronous (or network-agnostic) setting. Our result severely complicates refreshing keys in this context.

To deal with exposures, we consider a different model assuming that over the lifetime of the protocol, a total of at most t_s parties can become corrupted. Subject to this constraint, the adversary has flexible budget of corruptions (and corresponding uncorruptions). Since transient corruptions are rarely considered in the context of SMR, limiting the total number of faults to t_s seems like a reasonable assumption which is in line with most of the existing literature.

The impossibility of asynchronous proactive secret sharing without even modest synchronicity assumptions seems to be a folklore result. However, modelling and proving such a result turns out to be rather non-trivial. One of our contributions is to formalize this statement and provide a rigorous proof.

Practical network-agnostic SMR. We start by proposing two new efficient protocols for SMR, Update and Upstate.

- Update achieves security with optimal corruption thresholds and $O(n^3)$ communication complexity to commit a block of $O(n)$ transactions against an adaptive adversary. This presents an $O(n)$ improvement over Tardigrade, which requires $O(n^4)$ communication to commit blocks of $O(n)$ transactions (note that the average cost *per transaction* improves in our protocol). We obtain these communication improvements by carefully applying error correcting codes in a new ACS protocol.
- Upstate is secure for near optimal corruption thresholds and $O(n^2)$ communication complexity to commit blocks of $O(n)$ transactions under network changes against a static adversary. Upstate achieves its improved communication complexity using committees, making it well suited for large n. Upstate compares favorably to Upgrade: while Upgrade requires $O(n^3)$ communication to commit blocks of $O(n^2)$ transactions, Upstate commits blocks of size $O(n)$ transaction and requires $O(n^2)$ communication.

SMR tolerating key exposure. We show that our protocols are also secure in a changing network setting in which the adversary can be mobile across epochs, but is limited to moving among at most t_s unique parties. As a result, some parties may be exposed, i.e., have become uncorrupted but have state that was revealed to the adversary. We prove that by adding reboots at the beginning of each protocol epoch, Update and Upstate remain secure under key exposures caused by the adversary's mobility. Security in this case follows naturally from the structure of network-agnostic protocols. In order to be secure under a higher number of corruptions during the synchronous phase, some parts of the protocol have to use high thresholds for message collection. Although the adversary can know up to t_s keys/key shares during an asynchronous phase following a transition from a synchronous phase, it can only actively corrupt t_a parties and is not able to break security even if it forges or erases keys.

Open questions. Following our results, designing a secure SMR protocol with security under network changes and total quadratic communication complexity per committed block is an open problem if the adversary is adaptive or one does not rely on committees. Similarly, although our protocols use threshold cryptosystems to boost efficiency and censorship resilience, they are not a necessity for SMR. By removing the threshold cryptosystems and dealing with censorship resilience in a more heuristic way as in [25], one could bypass the impossibility result of asynchronous proactive secret sharing. Thus, it is plausible that a solution for secure independent key refresh and SMR against a mobile adaptive adversary and dynamic networks can be achieved without limiting the adversary to a total number of t_s corruptions. This could be done by using the same network-agnostic ACS protocol to agree on a new list of valid public keys obtained from distributed key generation. We leave this extension to future work.

1.3 Related work

Network-agnostic protocols were introduced by Blum *et al.* in the context of Byzantine agreement [6], and later extended to multi-party computation [8] and SMR [7]. Blum *et al.* [7] present two network-agnostic SMR protocols, Tardigrade and Upgrade. Tardigrade achieves total communication of $O(n^4 + n^3\ell)$ against adaptive adversaries, in terms of the number of parties n and a block size ℓ. Upgrade uses committees to achieve total communication of $O(n^3 + n\ell)$ against static adversaries (but tolerates a smaller corruption fraction). Appan *et al.* [4] proposed a protocol for network-agnostic perfectly secure multi-party computation; their protocol uses a novel network-agnostic perfectly secure VSS protocol (but not proactive).

Since our protocols need to support both synchronous and asynchronous networks, and asynchronous SMR protocols are less communication efficient compared to their synchronous counterparts [2,3], we focus here on asynchronous SMR protocols tolerating $t < n/3$ corruptions. Canonical constructions for SMR and atomic broadcast are based on multi-value validated asynchronous Byzantine agreement or asynchronous common subset [24,10,28,15,21] with cubic communication complexity for input sizes linear in n. Only a few existing protocols in the asynchronous setting tolerate *adaptive* corruptions. EPIC [25] and DAG-Rider [23] achieve adaptive security with cubic total communication complexity; Dumbo2 [21] can be modified to achieve adaptive security by using the MVBA from [26]. Neither can be easily adapted to a network-agnostic case.

A final group of related works concerns secret sharing and distributed key generation (DKG) in a settings where parties may crash or fail and then recover, or where the set of participants may change. *Proactive secret sharing* was introduced in Herzberg *et al.* [22]. Canetti *et al.* [11] and Frankel *et al.* [16] gave solutions against adaptive proactive adversaries for synchronous DKG using verifiable secret sharing schemes. Recently, Benhamouda *et al.* [5] introduced a novel secret sharing protocol for passing secrets from one anonymous committee to another, while Groth [20] proposed a publicly verifiable secret sharing-based DKG protocol that allows refreshing the key shares to a new committee. In the asynchronous case, Cachin *et al.* [9] presented a proactive refresh protocol under clock ticks that define epochs. Castro and Liskov investigate maintaining a common state in a distributed system via byzantine fault tolerance protocols [12,13]. Schulze *et al.* [32] proposed a mobile proactive secret sharing protocol in a partially synchronous network. Very recently, several works [27,33] have proposed more efficient dynamic/mobile proactive secret sharing protocols assuming eventual synchrony or short periods of synchrony at the end of an epoch. Concurrent work by Rimbaud and Urban [31] presented an asynchronous dynamic proactive secret sharing protocol under the assumption that the environment sends signals to start and end the epochs.

A related notion of security in the presence of exposed parties has been considered in [19], which studied synchronous authenticated broadcast with t_s corrupted parties and t_e exposed parties where $2t_s + \min(t_s, t_e - t_s) < n$.

Paper Organization. We describe our model in Section 2, and provide definitions in Section 3. In Section 4, we present an ACS protocol that uses error-correcting codes in order to achieve $O(n^3)$ communication against an adaptive adversary, and prove its special properties. This ACS protocol is used as a building block in the Update SMR protocol presented in Section 5, which achieves optimal corruption thresholds in a network-agnostic setting. In Section 6, we describe an asymptotically more efficient SMR protocol, Upstate, that is secure under near optimal thresholds against a static adversary and provide the committee-based building blocks. In Section 7, we prove that under a restricted adversarial model, the SMR protocols discussed so far remain secure under arbitrary network transitions. In Section 8, we model and provide an impossibility proof for proactive asynchronous verifiable secret sharing. This result motivates our restricted mobile adversarial model.

2 Model

Network. We consider n parties $P_1, \dots, P_n$ that are connected via pairwise authenticated channels and have access to a public key infrastructure. During the protocol's execution, transactions are delivered to parties' local buffers. We are not concerned with how these transactions originate; in practice, there is an external mechanism where clients gossip these transactions in the network.

When the network is *synchronous*, messages between parties are delivered with a finite, known delay Δ. In this setting, the local clocks of the parties are synchronized. When the network is *asynchronous*, messages between parties are assumed to be eventually delivered to their intended recipient, but may be adversarially delayed or reordered. In this case, the local clocks of parties are only assumed to be monotonically increasing and are not necessarily synchronized anymore. If an asynchronous phase is followed by a synchronous phase, all messages sent during the asynchronous phase of the network are delivered by the beginning of the synchronous phase. Transitions between a synchronous and an asynchronous behavior can happen arbitrarily.

An SMR protocol operates in logical intervals called *epochs*, which are measured and incremented locally. Another concept is that of a *round of communication*. In the synchronous setting, a round r refers to the time between $(r-1)\Delta$ and $r\Delta$. In the asynchronous case, the round number will describe some particular send actions that are performed by a party.

We assume that parties perform *atomic send operations*, i.e., parties can send a message to multiple parties simultaneously in such a way that the adversary cannot corrupt them in between individual sends. Moreover, we assume that the adversary cannot perform *after the fact removal*, i.e., the adversary cannot indefinitely prevent a message from being delivered once it is sent by an honest party, even if the adversary corrupts it at some point after the send action.

Threat model. We consider a *Byzantine fault* model, in which some fraction of the parties may be corrupted by an adversary. The adversary controls the local computations, messages, and current state of any corrupted party, and can coordinate the actions of all corrupted parties. Uncorrupted parties are called *honest* or *nonfaulty.* For any honestly-initiated communication, the adversary receives the epoch τ, the sender identity S, the receiver identity R and the message m (which can be encrypted, in which case the adversary does not see its contents). The adversary determines when to deliver each message.

We assume that the adversary is (t_a, t_s)-limited, i.e., for some fixed thresholds t_s, t_a $(t_a \leq t_s)$, up to $t_s < n/2$ parties may be corrupted if the network is synchronous and up to $t_a < n/3$ parties may be corrupted if the network is asynchronous. (The optimal trade-off between t_s, t_a is known to be $2t_s + t_a < n$ [7]). In Sections 4–5 we consider an *adaptive and rushing* adversary that adaptively corrupts parties over the course of a protocol execution; in Section 6, we consider a *static* adversary who corrupts parties prior to the start of an epoch.

Further, we address a mobile adversary. In Section 8, we consider an *epoch-wise mobile adaptive adversary* that can move freely between parties from epoch to epoch as long as it does not exceed more than t_s adaptive corruptions in the synchronous case and t_a adaptive corruptions in the asynchronous case at a given moment in time or in a given epoch. In Section 7, we consider a slightly different adversary who adaptively corrupts at most t_s parties *over the lifetime of the protocol*, and is only permitted to move between those t_s parties between epochs. We will explicitly mention the adversary's capabilities in each section.

Reboot. To enable protocols to withstand network changes, we assume a reboot mechanism that causes a party to restart its device, thereby flushing out the adversary. Reboots occur at specified points during the protocols. The adversary can immediately corrupt a party after rebooting, as long as it does not exceed the allowed threshold at that time. The restart is performed via code written in untamperable memory, such that the adversary cannot influence it. Importantly, rebooting does not remove the previous state of a corrupted party from the adversary's view; in particular, the adversary still knows the secret state of a party, including any secret keys that were held by that party during corruption. Furthermore, the internal state of a corrupted party that has restarted may have been arbitrarily modified by the adversary. For clarity, we call a party *actively corrupted* when the adversary actively controls that party's behavior and *passively corrupted* or *exposed* if the party was uncorrupted either by the adversary or by reboot.

Keys. Every party P_i holds a private key sk_i of a threshold signature scheme with individual public signature key pk_i and public key pk. Further, every party P_i holds a private key dk_i of a threshold encryption scheme with individual public verification key vk_i and public key ek. The threshold for both threshold schemes is t_s+1. We assume a trusted dealer that generates $\mathsf{PK} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_n, \mathsf{pk}, \mathsf{vk}_1, \ldots, \mathsf{vk}_n, \mathsf{ek})$ and $\mathsf{sk}_1, \ldots, \mathsf{sk}_n, \mathsf{dk}_1, \ldots, \mathsf{dk}_n$ and outputs a signature and encryption private keys $\mathsf{sk}_i, \mathsf{dk}_i$ and the public key PK to each party P_i.

A party P_i can use its signature key sk_i to generate a signature share σ_i on a message m. The signature share σ_i can be verified using the message m and the public verification key pk_i, and is called *valid* if the verification is successful. As a shorthand notation for legibility, we use $\langle m \rangle_i$ for a threshold signature σ_i of message m under secret key sk_i. A set of $t_s + 1$ valid signature shares on the same message m can be used to compute a signature σ for that message, which can be verified using the public key pk and m.

A party P_i can encrypt a message m using the public encryption key ek to generate a ciphertext c, and can use its decryption key dk_i to obtain a decryption share c_i of c. A decryption share c_i can be verified with respect to c, ek and vk_i and is called *correct* if the verification is successful. A set of $t_s + 1$ correct decryption shares can be used to obtain the decryption m of the ciphertext c.

We assume adaptively secure idealized threshold signature scheme and threshold encryption scheme. The sizes of a signature share and a full signature are $O(\kappa)$. We implicitly assume that parties use domain separation when constructing signatures, so that signatures are only valid in the context in which they were generated. The size of an encryption of a message m of length $|m|$ is assumed to be of length $|m| + O(\kappa)$, and a decryption share is assumed to have length $O(\kappa)$; these criteria can be met using standard KEM/DEM mechanisms.

3 Preliminaries

State machine replication (SMR) protocols enable a set of parties to emulate a single server by agreeing on an ever-growing, ordered log of transactions.[1] Given that SMR protocols usually continue indefinitely, we opt for a definition that clearly states how the logs are constructed and committed, and their relation order depending on epochs.

A party maintains an ever-growing append-only log consisting of *blocks* of transactions: $\mathsf{blocks}_i = (\mathsf{block}_i[1], \mathsf{block}_i[2], \ldots)$, where the notation $\mathsf{block}_i[e]$ refers to the block output by party P_i in epoch e. Each $\mathsf{block}_i[e]$ is initialized with a special character $\perp$ and populated by a set of transactions by P_i in epoch e. A party's epoch number is incremented after it outputs a block.

Definition 1 (State Machine Replication (SMR)). *Let Π be a protocol executed by n parties $P_1, \ldots, P_n$. Parties are provided with transactions as input,*

[1] Following [29], we distinguish between SMR and atomic broadcast in that the former explicitly requires an externally verifiable proof of the output validity.

locally maintain arrays blocks *as described above, and output* blocks *and a publicly verifiable proof* $\pi_i[e]$ *for each* $\mathsf{block}_i[e]$ *in* blocks*. Π is a secure state machine replication protocol tolerating t corruptions if the following properties hold:*

- *(t-Consistency) If an honest party outputs a block B in epoch e then all honest parties output B in epoch e.*
- *(t-Completeness) Every honest party outputs a block in all epochs.*
- *(t-Liveness) If a transaction* tx *is input to at least $n-t$ honest parties, then all honest parties eventually output a block containing* tx*.*
- *(t-External validity) If an honest party outputs (B, π), then for a fixed public Boolean function* Verify *it holds that* $\mathsf{Verify}(\mathsf{PK}, B, \pi) = 1$*.*

Definition 2 (Binary Byzantine Agreement (BA)**).** *Let Π be a protocol executed by n parties $P_1, \ldots, P_n$, where each party P_i begins holding input $x_i \in \{0,1\}$ and parties terminate upon generating output. Π is a secure byzantine agreement protocol tolerating t corruptions if the following properties hold:*

- *(t-Validity) If every honest party's input is equal to the same value x, then every honest party outputs x.*
- *(t-Consistency) All honest parties output the same message x.*
- *(t-Termination) Every honest party eventually terminates with output x.*

Definition 3 (Asynchronous Common Subset (ACS)**).** *Let Π be a protocol executed by n parties $P_1, \ldots, P_n$, where each party P_i begins holding input $x_i \in \{0,1\}^*$ and parties output sets of cardinality at most n. Π is a secure asynchronous common subset protocol tolerating t corruptions if the following properties hold:*

- *(t-Validity) If every honest party's input is equal to the same value x, then every honest party outputs the value $\{x\}$.*
- *(t-Validity with termination) If every honest party's input is equal to the same value x, then every honest party outputs the value $\{x\}$ and terminates.*
- *(t-Consistency) If an honest party outputs S, all honest parties output S.*
- *(t-Set quality) If an honest party outputs a set S, then S contains the input of at least one honest party.*
- *(t-Termination) Every honest party generates output and terminates.*

Block agreement (introduced in [7]) is a form of validated agreement on objects called *pre-blocks*. A pre-block is a vector of length n where the ith entry is either $\perp$ or a message along with a valid signature by P_i on that message. The *quality* of a pre-block is defined as the number of entries that are not $\perp$; a *k-quality pre-block* has quality at least k.

Definition 4 (Block Agreement (BLA)**).** *Let Π be a protocol executed by n parties $P_1, \ldots, P_n$, where each party P_i begins holding input $x_i \in \{0,1\}^*$ and terminates upon generating output. Π is a secure block agreement protocol tolerating t corruptions if the following properties hold:*

- *(t-Validity) If every honest party has input an $(n-t_s)$-quality pre-block, then every honest party outputs an $(n-t_s)$-quality pre-block.*
- *(t-Consistency) Every honest party outputs the same pre-block B.*

Next, we briefly introduce some standard cryptographic primitives we use throughout. Further details can be found in Appendix A.

Threshold signature scheme. A (t, n)-threshold signature scheme is a signature scheme allowing $t+1$ parties out of n to construct a signature on a message, out of which $t < n$ can be corrupted. It is *non-interactive* if parties can non-interactively construct signature shares that can be combined in order to construct the signature on a message, using the following protocols: $\mathsf{TS.Setup}$, $\mathsf{TS.KeyGen}$, $\mathsf{TS.Sign}$, $\mathsf{TS.ShVer}$, $\mathsf{TS.Verify}$ for setup, key generation, partial signing, share verification and signature verification, respectively. The desired properties are correctness, security (unforgeability under chosen-message attack) and robustness (any number of signature shares greater than $t+1$ can be combined to yield a signature) against a probabilistic polynomial-time adversary.

Linear error correcting codes. We adopt from [30] the description of error correcting codes, in particular, the Reed-Solomon (RS) code. An (n, b)-RS code encodes b data symbols into codewords of n symbols, and can decode the codewords to recover the original data.

Encoding: $(s_1, \ldots, s_n) \leftarrow \mathsf{ENC}(m_1, \ldots, m_b)$. Given inputs $m_1, \ldots, m_b$, an encoding function ENC computes codewords $s_1, \ldots, s_n$. By the property of the RS code, knowledge of any b elements of the codeword uniquely determines the input message and the remaining of the codeword.

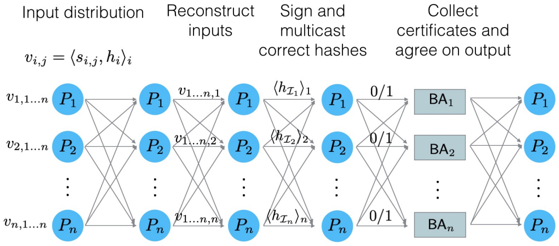
Decoding: $(m_1, \ldots, m_b) \leftarrow \mathsf{DEC}(s_1, \ldots, s_n)$. The function DEC computes $(m_1, \ldots, m_b)$, and is capable of tolerating up to c errors and d erasures in codewords $(s_1, \ldots, s_n)$, if and only if $n - b \geq 2c + d$.

Committee election. A first method to elect a committee uses threshold signatures to produce an unpredictable coin. The coin is used to determine an ordering of parties by computing the hash $H(\mathsf{coin}, i)$ and ordering the parties accordingly. To elect a size κ committee, one simply takes the first κ parties in the ordering. The second method, known as *cryptographic sortition*, uses verifiable random functions (VRF) to allow each party to individually determine whether they are part of a committee, and then prove their membership to others [18,1]. During the protocol, parties are elected to a committee if and only if the output of the VRF on a specific string is less than a parameter b.

Security parameters. Throughout the paper, we deal with several constants: the signature size and the hash output size, which depend on a security parameter λ that ensures computational security, and the committee sizes, which depend on a security parameter λ' that ensures a negligible failure probability. To not overburden notation, we denote all these constant sizes by κ.

4 Asynchronous Common Subset

Protocol Overview. The protocol proceeds as outlined in Figure 1. Each party $P_1, \ldots, P_n$, starts with an input of size ℓ and splits it into b blocks. These b blocks are then encoded into n codewords of size ℓ/b using an error correcting code (Section 3). Each party P_i forms a message containing the j-th codeword and a hash of the input, signs it and sends it to party P_j. Upon receiving a validly signed message, each party multicasts it, along with the associated signature which will serve as a proof of the codeword validity. We refer to this procedure of input distribution as INDI, and present it in Figure 2. This step, performed before agreement on whose messages to output, ensures that all parties are eventually able to reconstruct the selected inputs despite an adaptive adversary.



Input distribution
Reconstruct inputs
Sign and multicast correct hashes
Collect certificates and agree on output
$v_{i,j} = \langle s_{i,j}, h_i \rangle_i$
$v_{1,1\ldots n}$
$v_{2,1\ldots n}$
$v_{n,1\ldots n}$
P_1
P_2
P_n
$v_{1\ldots n,1}$
$v_{1\ldots n,2}$
$v_{1\ldots n,n}$
$\langle h_{\mathcal{I}_1} \rangle_1$
$\langle h_{\mathcal{I}_2} \rangle_2$
$\langle h_{\mathcal{I}_n} \rangle_n$
0/1
0/1
0/1
BA_1
BA_2
BA_n

Fig. 1. Diagram of the steps in the Π_{ACS} protocol. BA stands for Byzantine Agreement. $\mathcal{I}_i$ is the set of indices j for which party P_i reconstructed the initial message of party P_j.

Upon receiving $n - t_s$ messages containing codewords, parties attempt to reconstruct the input. Instructions related to reconstruction (referred to as RECON) are shown in Figure 2. Upon reconstructing a valid input from some party P_j, parties multicast a signed vote message. If an honest party has not managed to reconstruct an input yet, it waits for more messages, then calls RECON again.

Upon receiving $t_s + 1$ votes for a party P_j, parties assemble a certificate of validity for the reconstructed value of a party P_j, which consists of $t_s + 1$ signatures on h_j, used to form a full signature. The parties multicast a commit message carrying this certificate and the combined signature. We note that very recently, Das *et al.* [14] proposed an asynchronous reliable broadcast protocol using error correcting codes (but without digital signatures) that is related to this first step.

Finally, upon receiving a unique commit message for party P_j, parties input 1 to the corresponding BA_j instances. We implicitly assume that if honest parties receive conflicting commit messages, they do not input 1 to the respective BA. We use a binary BA protocol with t_a-validity, t_a-consistency, and t_a-termination in the presence of $t_a < n/3$ adaptive corruptions, and communication complexity of $O(n^2)$ if using threshold signatures (e.g., the BA protocol of [6]).

Protocol Π_{Term} (Figure 4) assembles an output certificate that allows parties who receive it to output and terminate (OC 0), ensuring no honest parties are "left behind".

Across the protocols, we use PK as the public keys output by $\mathsf{TS.KeyGen}$ and sk_i the secret key associated to P_i. ENC and DEC are associated to a (n, b)-RS code. For simplicity, in Π_{ACS} and the corresponding functionalities, we use $\varphi_{i,j}$ as both the signature of P_i over $s_{i,j}$ and separately, over h_i, sent to party P_j.

```
INDI($x$)

1. Encode $x$ using ENC into codewords $s_{i,1} \ldots, s_{i,n}$. Compute $h_i := H(x)$.
2. For $j \in [n]$, compute $\varphi_{i,j} := \mathsf{TS.Sign}(\mathsf{PK}, \mathsf{sk}_i, (s_{i,j}, h_i))$. Set $v_{i,j} := (s_{i,j}, h_i, \varphi_{i,j})$. Send $v_{i,j}$ to party $P_j$.
3. Upon receiving a valid $v_{j,i} = (s_{j,i}, h_j, \varphi_{j,i})$, multicast $\langle v_{j,i} \rangle_i$.
4. Output the received set of $\{\langle v_{j,k} \rangle_k\}$ for $P_j$ from $P_k$.

RECON($\{\langle v_{j,k} \rangle_k\}$)

1. Parse $v_{j,k}$ as $(s_{j,k}, h_j, \varphi_{j,k})$ and ignore the ones with invalid signatures
   (either from $P_j$ or from $P_k$). Let $K$ be the set of remaining messages.
2. If there exists a subset $K' \subseteq K$ such that $|K'| \geq n - t_s$ and all contained
   messages $v_{j,k}$ have the same value $h_j$, compute $x = \mathsf{DEC}(\{s_{j,k}\}_{k \in K'})$.
3. If $H(x) = h$, output $x$. Else, output $\perp$.
```

Fig. 2. Input distribution and reconstruction from the perspective of party $P_{i \in \{1,\ldots,n\}}$.

Communication complexity. The Π_{ACS} protocol has a communication complexity of $O(n^2\ell + \kappa n^3)$ per input of size ℓ.

Encoding and reconstruction. In the reconstruct procedure RECON, before feeding the codewords into the DEC algorithm, parties first check that the corresponding signatures are correct. Then, parties check whether at least $n - t_s$ of the messages have the same associated hash value. Thus, each party feeds at least $n - t_s$ valid codewords in DEC.

The error-correcting code used here is an (n, b)-RS code (Section 3), which allows a party to split an input in b blocks and encode them into n codewords. In order to tolerate d erasures, it must be possible to reconstruct the b blocks from $n - d$ correct codewords. Furthermore, to tolerate c errors among $n - d$ codewords, it must hold that $n - b \geq 2c + d$.

If we let b equal t_s, we can tolerate either $t_s + t_a$ erasures, or tolerate t_a errors along with $t_s - t_a$ erasures (since $n > 2t_s + t_a$). This means we need to wait for $n - t_s + t_a$ codewords in total in order to guarantee correct reconstruction in the asynchronous case when t_a parties are corrupted. Thus, a gain in communication efficiency, obtained from using codewords to achieve agreement on length κ hashes instead of length ℓ inputs and from not multicasting the reconstructed

$\Pi_{\mathsf{ACS}}(x_i)$

1. Run $\mathsf{INDI}(x)$ and store $\{\langle v_{j,k}\rangle_k\}$ for P_j as they are received from P_k.
2. Input $\{\langle v_{j,k}\rangle_k\}$ to RECON. If RECON outputs x_j, multicast a vote $\mathsf{vote}_i := \langle \mathsf{vote}, \langle h_j\rangle_i, \varphi_{j,i}\rangle_i$.
3. Upon receiving $t_s + 1$ valid votes from distinct P_k on j, combine the threshold signatures into a full signature and form a certificate $c_j := (\mathsf{commit}, \langle h_j\rangle)$ and send it to all parties.
4. Upon receiving the certificate c_j for the input of a party P_j, forward it to all parties.
5. Upon receiving a certificate for party P_j input 1 to BA_j. After outputting 1 in at least $n - t_a$ BA instances, input 0 for the rest.
6. Set $\mathcal{S}$ to be the set of indices of the BA instances that delivered 1.
7. Output according to the following output conditions:

OC 0. If P_i has received a valid output certificate $(\mathsf{output}, \tilde{c}, x, h)$, multicast $(\mathsf{output}, \tilde{c}, x, h)$. Output x and terminate.

OC 1. If P_i (1) has obtained $n - t_s$ certificates $(\mathsf{commit}, \langle h_j\rangle)$ and (2) reconstructed inputs x_j such that $h_j = H(x_j)$ of distinct P_j, all have the same value x, then input (x_j, h_j) to Π_{Term}.

OC 2. Else if P_i has (1) $|\mathcal{S}| \geq n - t_a$, (2) all n BA instances have terminated, (3) P_i has obtained certificate $(\mathsf{commit}, \langle h_j\rangle)$ for $j \in \mathcal{S}$, (4) reconstructed input x_j such that $h_j = H(x_j)$ and such that a strict majority of $\{x_j\}_{j\in\mathcal{S}}$ has value x, then input (x_j, h_j) to Π_{Term}.

OC 3. Else if P_i has (1) $|\mathcal{S}| \geq n - t_a$, (2) all n BA instances have terminated, (3) P_i has obtained certificates $(\mathsf{commit}, \langle h_j\rangle)$ and (4) reconstructed input x_j such that $h_j = H(x_j)$ for all $j \in \mathcal{S}$, then output $S = \bigcup_{j\in\mathcal{S}} x_j$ and terminate.

Fig. 3. ACS protocol from the perspective of party $P_{i\in\{1,\ldots,n\}}$.

$\Pi_{\mathsf{Term}}(x, h)$

1. Multicast $\langle x, h\rangle_i$.
2. Upon receiving at least $t_s + 1$ valid signature shares $\langle x, H(x)\rangle_i$ from distinct parties, aggregate the signature shares into an output certificate $\tilde{c}$ for x and multicast $(\mathsf{output}, \tilde{c}, x, H(x))$. Output x and terminate.
3. Upon receiving a valid output certificate $\tilde{c}$ for x, multicast $(\mathsf{output}, \tilde{c}, x, h)$. Output x and terminate.

Fig. 4. Termination helper protocol from the perspective of party $P_{i\in\{1,\ldots,n\}}$.

output, leads to potentially having to wait for $n - t_s + t_a$ messages in order to reconstruct the correct output if the adversary delivered t_a bad codewords.

On the other hand, if we set b to be equal to t_a, we can tolerate either t_s errors and no erasures, or $2t_s$ erasures. This corresponds to the synchronous case when t_s parties are corrupted, and honest parties receive all messages that were sent after at most Δ time. Therefore, if an honest party only receives $n-t_s$ codewords, they are all correct. However, we will show below that there is no need to tolerate

t_s errors in the synchronous case. Briefly, we can use extra information—the hash value—in order to detect an incorrect reconstruction, and there will be sufficiently many inputs of the honest parties correctly reconstructed in order to achieve termination. Therefore it suffices to let $b = t_s$ throughout.

Lemma 1. *Suppose there are at most t_a corruptions. Given a certificate for a party P, $(\mathsf{commit}, \langle h \rangle)$, all honest parties can eventually reconstruct the same output in a run of Π_{ACS}.*

Proof. If P is honest, then all honest parties will eventually receive $n - t_s$ valid codewords of the true input (since we assume unforgeable signatures), allowing them to correctly reconstruct x.

Assume P is dishonest. In order to obtain a valid certificate on P's hash $\langle h \rangle$, $t_s - t_a + 1$ honest parties need to have seen $n - t_s$ valid messages, all with the same $h = H(x)$. Of these $n - t_s$ messages, t_a could have been sent by corrupted parties in the multicast round. In the worst case, in the first round when P sent codewords, it could have sent only $n - t_s - t_a$ codewords (but all valid) to distinct honest parties. Eventually, all honest parties receive the $n - t_s - t_a$ codewords and can reconstruct the same input x if the code tolerates $t_s + t_a$ erasures.

On the other hand, the adversary might send t_a malicious codewords which will prevent correct reconstruction from $n - t_s$ codewords. However, assuming H is a collision-resistant hash function, except with negligible probability, there do not exist inputs $x \neq x'$ reconstructed by different sets of codewords such that $h = H(x) = H(x')$. Therefore, if after inputting $n - t_s$ codewords to RECON and not obtaining a valid output with respect to h, the honest parties wait until they receive sufficient codewords in order to be able to correctly reconstruct.

As mentioned in Section 4, each input of size ℓ is split into to $b = t_s$ blocks: $n - t_s > t_a + t_s = 2t_a + t_s - t_a$. This means that the code can tolerate either $t_a + t_s$ erasures, or $t_s - t_a$ erasures and t_a errors if parties wait for $n - t_s + t_a$ messages to honest parties. □

Lemma 2. *If there are at most t_a-corruptions, there cannot be two valid certificates $(\mathsf{commit}, \langle h \rangle)$, $(\mathsf{commit}, \langle h' \rangle)$, associated with P, and $h \neq h'$.*

Proof. If P is honest, then all honest parties eventually receive $n - t_s$ valid messages containing codewords and the same hash h of the true input, so they can correctly reconstruct x. Therefore, assuming unforgeable signatures, no valid commit message $(\mathsf{commit}, \langle h' \rangle)$ for $h' \neq h$ can exist.

Now suppose P is dishonest. Since there is a certificate $(\mathsf{commit}, \langle h \rangle)$ constructed from at least $t_s + 1$ signatures, and $t_s + 1 > t_a$, at least one honest party P_j signed h. This implies P_j reconstructed an input x such that $h = H(x)$ and saw $n - t_s$ distinct valid messages $v_{*,l} = (s_{*,l}, h)$. At most t_a messages could have originated from malicious parties, so $n - t_s - t_a > t_s + 1$ were messages that honest parties relayed honestly. Assume there is a different honest party P_i that participated in a different certificate on h' for P. Then that party also saw $n - t_s$ distinct valid messages $v_{*,l'} = (s_{*,l'}, h')$, out of which $n - t_s - t_a > t_s + 1$ were messages that honest parties relayed honestly. These sets of honest parties should not intersect, so $2(n - t_s - t_a) < n - t_a$, but then this contradicts our assumption that $n > 2t_s + t_a$. □

Remark 1. Note that if the network is synchronous and $t_s = \lfloor n/2 \rfloor, t_a = 0$, different honest parties could receive certificates on different hashes of the same malicious party (this certainly happens because honest parties multicast the received certificates). In such a case, honest parties detect equivocation and do not input 1 in the associated BA. However, if the network is asynchronous and there are $t_s \leq n/2$ corruptions, then equivocation is not detected. Nevertheless, as we see below, validity will still hold.

Lemma 3. *Π_{ACS} satisfies t_s-validity with termination.*

Proof. Suppose all honest parties have the same input x and up to t_s parties are corrupted. At most $t_s < \left\lfloor \frac{n-t_a}{2} \right\rfloor + 1 < n - t_s$ reconstructed values can be different than x, so there cannot exist an output certificate on a value $x' \neq x$ even if two honest parties accept different commit certificates for the same corrupted party.

Honest parties will eventually be able to obtain valid certificates for the inputs of at least $n - t_s$ honest parties, and therefore (by assumption) eventually obtain at least $n - t_s$ valid certificates for x. At this point, if an honest party has not yet output, it will input $\{x\}$ to Π_{Term} (in OC 1). If at least $t_s + 1$ parties call Π_{Term} via OC 1, then eventually, each party will receive an honest output certificate on $\{x\}$, output and terminate. Below we handle the case in which some honest parties output before the above conditions were satisfied.

Assume party P output before the above could occur. If P called Π_{Term} via OC 2 then it saw x' reconstructed in a strict majority of valid values associated with $n - t_a$ BA terminated BA instances. Any set of instances constituting a strict majority must contain at least one instance corresponding to honest party, since $\left\lfloor \frac{n-t_a}{2} \right\rfloor + 1 > t_s + 1$, and so $\{x'\} = \{x\}$ by assumption. Furthermore, in this case P would have input (x, h) to Π_{Term}, and so all parties eventually receive an output certificate on $\{x\}$. Since $n - t_s > \left\lfloor \frac{n-t_a}{2} \right\rfloor + 1$, and honest parties' inputs can always eventually be reconstructed, each honest party will be eventually able to output due to OC 0, even if it was not able to finish the reconstruction of the corrupted parties' inputs.

Finally, if P output S as a result of OC 3, then P did not observe a strict majority of BA instances in $\mathcal{S}$ corresponding to the same value. By assumption, the honest parties have the same input x, so this implies a strict majority of values $\mathcal{S}$ correspond to corrupted parties. However, this contradicts the assumption that only t_s parties are corrupted, because $\lfloor \frac{|\mathcal{S}|}{2} \rfloor \geq t_s$. Therefore, no honest party outputs via OC 3 when all honest parties have the same input. □

Lemma 4. *Π_{ACS} satisfies t_a-set quality.*

Proof. Suppose an honest party P_i output a set S.

If P_i output $S = \{x\}$ due to OC 0, then P_i must have obtained a valid output certificate of at least t_s+1 signatures on x, which requires that at least one honest party (call it P_j) input (x, h) to $\Pi_{\mathsf{Term}}(x, h)$ in OC 1 or OC 2. Consider each case. If P_j input (x, h) due to OC 1, then it gathered a valid certificate on at least $n-t_s$ values equal to x. At least $n-t_s-t_a \geq t_s+1$ of the parties associated to these values are honest, so RECON returns their correct original input value. Otherwise, if P_j input (x, h) due to OC 2, then it output 1 in at least $n-t_a$ BA instances and it saw a strict majority of the reconstructed corresponding inputs reconstruct to the value x. Because $n > n - t_s + \left\lfloor \frac{n-t_a}{2} \right\rfloor + 1$, x was input by some honest party. Thus, in either case some honest party input x.

If P output S due to OC 3, then it output 1 in at least $n-t_a$ BA instances but without the majority condition satisfied. At least one of these instances corresponds to an honest party, so S contains some honest party's input. □

Lemma 5. *Π_{ACS} is t_a-terminating.*

Proof. Assume no honest party has output yet. Eventually, all honest parties will obtain at least $n-t_a$ valid commit certificates, since there are at least $n-t_a$ honest parties. Moreover, by Lemma 2, even on malicious inputs, honest parties cannot obtain multiple valid certificates. By the t_a-terminating property of BA, all parties terminate all n BA instances eventually. By the t_a-consistency of BA, all honest parties will agree on the set $\mathcal{S}$ of BA instances that output 1. Finally, by Lemma 1, all honest parties reconstruct the same inputs associated to $\mathcal{S}$. This allows some honest party to output and terminate.

It remains to show that once some honest party has terminated, all honest parties eventually terminate. In the rest of the proof, suppose an honest party P_i terminated, and consider each possible output condition.

If P_i output due to OC 0 then eventually all honest parties receive the certificate multicast by P and terminate (if they have not already).

If P_i output due to condition OC 3, then it must have terminated all BA instances, and obtained certificates and reconstructed all inputs corresponding to $\mathcal{S} = \{i | \mathsf{BA}_i \text{ output } 1\}$ for some $|\mathcal{S}| \geq n - t_a$. Then, t_a-termination and consistency of BA ensure that each other honest party P_j eventually observes part (1) and (2) of OC 3 to be true. Furthermore, each honest party eventually reconstructs each $\{x_j\}_{j \in \mathcal{S}}$ and receives the certificates needed terminate, since P_i must have sent these certificates to all other parties during ACS. □

Lemma 6. *Π_{ACS} satisfies t_a-consistency.*

Proof. Assume an honest party P_i has output S. By Lemma 5, each other honest party eventually outputs some set S'. It remains to show that for each possible combination of output conditions, $S = S'$.

Suppose $S = \{x\}$ was output via OC 0, i.e., upon receiving a valid output certificate. There are two subcases.

First, suppose P_j output $S' = \{x'\}$ via OC 0. The existence of a certificate for x implies that there exists an honest party P who contributed a share via either OC 1 or OC 2; likewise, some honest party P' contributed a share for x'. If both P and P' contributed shares via OC 1, then quorum intersection among the two sets of $n-t_s$ certificates implies $x = x'$. If (say) P and P' contributed shares by OC 1 and OC 2, respectively, then any set of $n-t_s$ BA instances and any set of $\left\lfloor \frac{n-t_a}{2} \right\rfloor + 1$ BA instances must intersect at an honest party, and so $x = x'$. Finally, if both P and P' contributed shares via OC 2, then they agree on $\mathcal{S}$, and once again $x = x'$.

Second, suppose towards a contradiction that P_j output $S = \cup_{j \in \mathcal{S}} x_j$ for reconstructed values x_j via OC 3. Of those $n - t_a$ values, at most t_s can have a value $x' \neq x$. But this means that P_j saw at least $n - t_a - t_s \geq t_s + 1$ reconstructed values equal to x, in which case the order of else-if clauses would have caused P_j to output via OC 2, a contradiction.

Now, say P_i outputs S as a result of OC 3. The case in which P_j output $\{x'\}$ via OC 0 is equivalent to the second subcase above, so suppose P_j also output a set S' via OC 3. Both P_i and P_j must have seen all BA instances terminate, and furthermore agree on the set of BA instances $\mathcal{S}$ that output 1. By Lemma 1, we have $S' = S$. □

5 Update SMR

In this section, we consider an adaptive adversary without mobility, which can actively corrupt at most t_s parties if the network is synchronous, and can corrupt at most t_a parties if the network is asynchronous, in any given epoch. Protocol 5 describes our construction for a network-agnostic SMR protocol.

Apart from the asynchronous common subset protocol described in Section 4, we will also employ a block agreement protocol. The role of the block agreement is for parties to agree on the input to ACS if the network is synchronous. Honest parties are assumed to input $(n - t_s)$-quality pre-blocks of total length L to the block agreement protocol and ignore any pre-blocks with quality less than $n - t_s$.

We use the adaptively secure block agreement protocol from [7], which we denote by Π_{BLA}. The protocol has a total complexity of $O(\kappa n^3 + \kappa n^2 L)$ per pre-block of size L. Π_{BLA} has R inner rounds and guarantees t_s-validity, t_s-consistency and t_s-termination in a synchronous network when up to t_s parties are corrupted. We cannot guarantee these in an asynchronous network. However, even if the network is asynchronous, any honest party who terminates Π_{BLA} does so with output that is a valid $n - t_s$-quality pre-block.

The logical flow of the network-agnostic SMR is the following. In every epoch, each honest party first selects a random sample of L/n transactions from its buffer of transactions. The selected transactions are then threshold encrypted. Next, the parties multicast their encrypted samples and start to assemble a $(n - t_s)$-quality pre-block. If an honest party succeeds in assembling such a pre-block within the allotted time, it inputs it to Π_{BLA}, which is guaranteed to terminate with consistent output B^* if the network is synchronous. Regardless, honest parties will then input either B^* if obtained from Π_{BLA} or a $(n - t_s)$-quality pre-block to Π_{ACS}. Recall that Π_{ACS} is guaranteed to terminate regardless of the network condition. Lastly, honest parties participate in constructing the final block: they jointly decrypt the output value of Π_{ACS}, populate the block with the unique transactions, assemble a validity certificate on the hash of the obtained block, and remove the posted transactions from their buffer.

We consider that each e starts for a party at time $T_e = \mu(e-1)$ as measured by the local clock. The parameter μ is a spacing parameter that should be heuristically tuned by the network designers in order to improve throughput, i.e., not have too much of an overlap or of a separation between epochs. If the network is synchronous, then epochs start at the same time for all parties, since clocks are synchronized. If the network is asynchronous, parties might start the epochs at different times and might not output a block until they have to start the next epoch. We implicitly assume parties can distinguish between messages from different epochs, e.g. by tagging messages with the epoch number.

```
$\Pi_{\mathsf{SMR}}$

Step 1. Proposal selection.
1.1 At time $T_e = \mu(e-1)$: Set $B_i^e := (\perp, \ldots, \perp)$ an empty pre-block of size
    $n$, and set $\mathsf{ready}_e = \mathsf{false}$.
1.2 Let $x_i$ be a threshold encryption of a random selection of $L/n$ transactions
    without replacement from the first $L$ transactions in the party's
    buffer. Multicast $x_i$.
1.3 Upon receiving a validly signed message $x_j$, if $B_i^e[j] = \perp$, set $B_i^e[j] := x_j$.
1.4 Upon assembling a $(n - t_s)$-quality pre-block $B_i^e$, set $\mathsf{ready}_e = \mathsf{true}$.

Step 2. Agreement.
2.1 At time $T_e + \Delta$: If $\mathsf{ready}_e = \mathsf{true}$, pass $B_i^e$ as input to $\Pi_{\mathsf{BLA}}^e$. If $\Pi_{\mathsf{BLA}}^e$
    terminates, let $B^*$ be the output.
2.2 At time $T_e + (5R+1)\Delta$: Terminate $\Pi_{\mathsf{BLA}}^e$ if not already terminated.
2.3 Pass $B^*$ or wait until $\mathsf{ready}_e = \mathsf{true}$ and pass $B_i^e$ as input to $\Pi_{\mathsf{ACS}}^e$.
2.4 Receive $S = \{B_j^*\}_{j \in \mathcal{S}}$, where $\mathcal{S} \subset \{1, \ldots, n\}$ from $\Pi_{\mathsf{ACS}}^e$.

Step 3. Output and public verification.
3.1 On input $S = \{B_j^*\}_{j \in \mathcal{S}}$, for each $j \in \mathcal{S}$, do:
  - Jointly decrypt the values in $S = \{x_j\}_{j \in \mathcal{S}}$.
  - Create a $\mathsf{block}$ by sorting $\bigcup_{j \in \mathcal{S}} x_j$ in canonical order.
  - Hash and sign $\mathsf{block}$, then multicast $\langle H(\mathsf{block}) \rangle_i$.
3.2 On receiving $t_s + 1$ distinct valid signatures $\langle h \rangle_j$ s.t. $h = H(\mathsf{block})$, do:
  - Assemble $\pi$ as $\langle h \rangle$ and proof of correct decryption of $S$.
  - Remove the transactions in $\mathsf{block}$ from the current buffer and output
    $(\mathsf{block}, \pi)$.
3.3 Update $e \leftarrow e + 1$.
```

Fig. 5. Update SMR protocol with adaptive security for party $P_{i \in \{1,\ldots,n\}}$.

Below we give our main results on Update. The proofs use the results on Π_{ACS} and Π_{BLA} discussed so far, and are provided in Appendix D.

Condition $(*)$**.** Assume $t_a \leq t_s$, $n > 2t_s + t_a$ and $t_a \leq n/3$, $t_s \leq n/2$.

Theorem 1. *Under condition* $(*)$*,* Π_{SMR} *is (1)* t_s*-consistent and* t_s*-complete if the network is synchronous and (2)* t_a*-consistent and* t_a*-complete if the network is asynchronous.*

Theorem 2. *Under condition* $(*)$*,* Π_{SMR} *is (1)* t_s*-externally valid if the network is synchronous and (2)* t_a*-externally valid if the network is asynchronous.*

Theorem 3. *Under condition* $(*)$*,* Π_{SMR} *is (1)* t_s*-live if the network is synchronous and (2)* t_a*-live if the network is asynchronous.*

Complexity In Π_{SMR}, the parties select a batch of L/n transactions, construct a pre-block of size $O(L|\mathsf{tx}|)$, and input the pre-block to Π_{BLA}. If Π_{BLA} outputs, it also outputs a pre-block of size $O(L|\mathsf{tx}|)$. The input to Π_{ACS} is of size $O(L|\mathsf{tx}|)$, and if the network is synchronous, the output is of size $O(L|\mathsf{tx}|)$. Conversely, if the network is asynchronous, the output is of size $O(nL|\mathsf{tx}|)$. Since the transactions were randomly selected from honest parties' buffers, with high probability there will be $O(nL)$ transactions in the output block after decryption, assuming that throughput is not limited by a lack of transactions.

Step 1 of Π_{SMR} incurs $O(nL|\mathsf{tx}| + n^2\kappa)$ total communication. In step 2, Π_{BLA} incurs $O(\kappa n^3 + \kappa n^2 L|\mathsf{tx}|)$ total communication and Π_{ACS} incurs $O(\kappa n^3 + n^2 L|\mathsf{tx}|)$ total communication. Finally, in step 3, the parties assemble an output block and then multicast the signatures of the hash of the block to construct a proof, incurring $O(\kappa n^2)$ communication.

Summing over all steps, we see that Update incurs a total communication of $O(\kappa n^3 + \kappa n^2 L|\mathsf{tx}|)$. Choosing a proposal sample size L that is $O(n)$ yields an asymptotic total communication of $O(\kappa n^3)$ per block of transactions and an amortized communication complexity of $O(\kappa n^2)$ per transaction.

6 Upstate SMR

We consider a static adversary that is able to corrupt up to $\hat{t}_a = (1-\epsilon)t_a$ parties in the asynchronous case and up to $\hat{t}_s = (1-\epsilon)t_s$ parties in the synchronous case, for a small $\epsilon > 0$. Informally, the ϵ slack in the corruption thresholds ensures that with high probability the fraction of corruptions in a smaller committee chosen at random is close to the fraction of corruptions in the pool of n parties.

Protocol 7 describes our construction for a network-agnostic committee-based SMR protocol. At the start of each epoch, parties choose a random sample of L/κ transactions from their buffers. The parties then run an input selection procedure, called INSE and described below, to select κ committee members. Inputs from committee members are gathered into pre-blocks, which are passed to committee-based versions of BLA and ACS in the same way as in Update. (Because the committee is only size κ, the pre-blocks are $(1 - t_s/n)\kappa$-quality.) The committee-based ACS and BLA protocols are described at the end of the section, with additional details in Appendix D. After running BLA and ACS, the parties construct the final block by jointly decrypting the output value of $\Pi^{\kappa}_{\mathsf{ACS}}$.

Figure 6 describes the input selection mechanism INSE^κ that handles input encoding and primary committee election. The input of size $\ell = L/\kappa$ is split as before into b blocks, which are then encoded into n codewords of size ℓ/b (Section 3). Each party sends the i-th codeword with a hash and a threshold signature over the epoch number to party P_i. Combining $\hat{t}_s + 1$ threshold signatures yields an unpredictable value that is used to select a committee of κ parties whose inputs will form the output.

```
$\mathsf{INSE}^\kappa(e, x_i)$

1. Encode $x_i$ using $\mathsf{ENC}$ into codewords $s_{i,1} \ldots, s_{i,n}$.
2. Compute $h_i := H(x_i)$ and signature $\sigma_i := \mathsf{TS.Sign}(\mathsf{PK}, \mathsf{sk}_i, e)$.
3. Set $v_{i,j} := (s_{i,j}, h_i, \sigma_i)$. For $j \in \{1, \ldots, n\}$, send $(v_{i,j}, \varphi_{i,j})$ to party $P_j$,
   where $\varphi_{i,j} := \mathsf{TS.Sign}(\mathsf{PK}, \mathsf{sk}_i, v_{i,j})$.
4. Upon receiving $n - \hat{t}_s$ messages $v_{j,i} = (s_{j,i}, h_j, \sigma_j)$, select $\hat{t}_s + 1$ signa-
   tures $\sigma_j$ and compute $\mathsf{coin}$ from them.
5. For each $j \in \{1, \ldots, n\}$, compute $\bar{h}_j := H(\mathsf{coin}, j)$ and select the first $\kappa$
   values to populate the primary committee index set $\mathcal{C}$.
6. For each $j \in \mathcal{C}$, multicast the codeword $s_{j,i}$ and $\varphi_{j,i}$ received from $P_j$.
7. For each member $j$ in $\mathcal{C}$, output the received $\{s_{j,k}, \varphi_{j,k}, h_j\}$, from $P_k$.
```

Fig. 6. Input selection—input encoding and primary committee election—from the perspective of party $P_{i \in \{1,\ldots,n\}}$ in epoch e.

Condition $(**)$**.** Assume $t_a \leq t_s$, $n > 2t_s + t_a$, $t_a \leq n/3$, $t_s \leq n/2$ and $\hat{t}_a := (1-\epsilon)t_a$, $\hat{t}_s := (1-\epsilon)t_s$ for $\epsilon > 0$.

Theorem 4. *Under condition* $(**)$ *except with negligible probability,* $\Pi^\kappa_{\mathsf{SMR}}$ *is (1)* $\hat{t}_s$*-consistent,* $\hat{t}_s$*-complete,* $\hat{t}_s$*-externally valid and* $\hat{t}_s$*-live if the network is synchronous and (2)* $\hat{t}_a$*-consistent,* $\hat{t}_a$*-complete,* $\hat{t}_a$*-externally valid and* $\hat{t}_a$*-live if the network is asynchronous.*

The proof follows along the same lines as the proofs of Theorems 1–3, using the properties of the committee-based protocols $\Pi^\kappa_{\mathsf{ACS}}$ and $\Pi^\kappa_{\mathsf{BLA}}$.

Committee-based Asynchronous Common Subset We now present an ACS protocol $\Pi^\kappa_{\mathsf{ACS}}$ in a network-agnostic setting with static corruptions. An instance of the protocol is parameterized by the committee size κ and an epoch number e, as used in $\Pi^\kappa_{\mathsf{SMR}}$.

An overview of the protocol appears in Figure 8. Inputs of size ℓ are passed to the input selection procedure INSE (Figure 6), which determines the *primary committee* $\mathcal{C}$. Next, each party multicasts the codewords they received from the members of the primary committee. To reduce communication, one *secondary* committee is elected for each member of the primary committee. The secondary committee is responsible for constructing certificates of correctness for the reconstructed values of the primary committee. The secondary committees are

$\Pi^{\kappa}_{\mathsf{SMR}}$

Step 1. Proposal selection.

1.1 At time $T_e = \mu(e-1)$: Set $B^e_i := (\bot, \ldots, \bot)$ an empty pre-block of size κ, and set $\mathsf{ready}_e = \mathsf{false}$.

1.2 Let x_i be a threshold encryption of a random selection of L/κ transactions without replacement from the first L in the party's buffer.

1.3 Run $\mathsf{INSE}(e, x_i)$ and store $\mathcal{C}$ and $\{s_{j,i}, \varphi_{j,i}, h_j\}_{j\in\mathcal{C}}$, as they are received.

1.4 Upon receiving $n - \hat{t}_s$ codewords of x_j, if (1) $h_j = H(x_j)$ and $B^e_i[j'] = \bot$, set $B^e_i[j'] := x_j$, where j' is the lexicographic order of P_j in $\mathcal{C}$.

1.5 Upon assembling a $(1 - t_s/n)\kappa$-quality pre-block B^e_i, set $\mathsf{ready}_e = \mathsf{true}$.

Step 2. Agreement.

2.1 At time $T_e + 2\Delta$: If $\mathsf{ready}_e = \mathsf{true}$, pass B^e_i as input to $\Pi^{\kappa,e}_{\mathsf{BLA}}$. If $\Pi^{\kappa,e}_{\mathsf{BLA}}$ terminates, let B^* be the output.

2.2 At time $T_e + (7R+2)\Delta$: Terminate $\Pi^{\kappa,e}_{\mathsf{BLA}}$ if not already terminated.

2.3 Pass B^* or wait until $\mathsf{ready}_e = \mathsf{true}$ and pass B^e_i as input to $\Pi^{\kappa,e}_{\mathsf{ACS}}$.

2.4 Receive $S = \{B^*_j\}_{j\in\mathcal{S}}$, where $\mathcal{S} \subset \{1, \ldots, n\}$ from $\Pi^{\kappa,e}_{\mathsf{ACS}}$.

Step 3. Output and public verification.

3.1 Run Step 3 from $\mathsf{Update}\ \Pi_{\mathsf{SMR}}$.

Fig. 7. SMR protocol with adaptive security for party $P_{i\in\{1,\ldots,n\}}$.

self-elected as described in Section 3. Finally, parties agree on which primary committee members' values to output by running κ parallel BA instances.

For simplicity, in Step 2 in $\Pi^{\kappa}_{\mathsf{ACS}}$, we use $\varphi_{i,j}$ as both the signature of P_i over $s_{i,j}$ and over h_i, sent to P_j. Across the protocols, H denotes a collision-resistant hash function and b a bound ensuring committees of size κ in expectation.

Lemma 7. *$\Pi^{\kappa}_{\mathsf{ACS}}$ is $\hat{t}_a$-consistent, $\hat{t}_a$-terminating, has $\hat{t}_a$-set quality and $\hat{t}_s$-validity with termination except with negligible probability.*

Inputs are split into $b = \hat{t}_s$ blocks using an error correcting code that tolerates either $\hat{t}_s$ erasures or $\hat{t}_a$ errors and $\hat{t}_s - \hat{t}_a$ erasures.

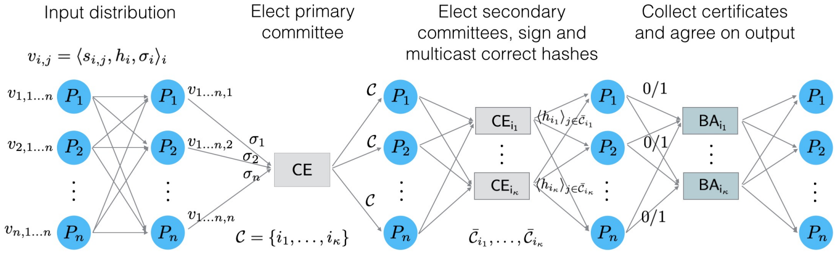

Input distribution
Elect primary committee
Elect secondary committees, sign and multicast correct hashes
Collect certificates and agree on output
$v_{i,j} = \langle s_{i,j}, h_i, \sigma_i\rangle_i$
$v_{1,1\ldots n}$
$v_{2,1\ldots n}$
$v_{n,1\ldots n}$
P_1
P_2
P_n
$v_{1\ldots n,1}$
$v_{1\ldots n,2}$
$v_{1\ldots n,n}$
σ_1
σ_2
σ_n
CE
$\mathcal{C} = \{i_1, \ldots, i_\kappa\}$
$\mathcal{C}$
CE_{i_1}
CE_{i_κ}
$\langle h_{i_1}\rangle_{j\in\bar{\mathcal{C}}_{i_1}}$
$\langle h_{i_\kappa}\rangle_{j\in\bar{\mathcal{C}}_{i_\kappa}}$
$\bar{\mathcal{C}}_{i_1}, \ldots, \bar{\mathcal{C}}_{i_\kappa}$
0/1
BA_{i_1}
BA_{i_κ}

Fig. 8. Diagram of the steps in the $\Pi^{\kappa}_{\mathsf{ACS}}$ protocol. CE stands for committee election and BA for Byzantine Agreement.

$\Pi^{\kappa,e}_{\mathsf{ACS}}(x_i)$

1. Run $\mathsf{INSE}^{\kappa}(e, x_i)$ and store $\mathcal{C}$ and $\{\{s_{j,k}\}, h_j, \{\varphi_{j,k}\}\}_{j\in\mathcal{C}}$, as they are received from P_k.
2. For each $j \in \mathcal{C}$, elect a secondary committee $\bar{\mathcal{C}}_j$ of size $O(\kappa)$ as follows:
   - Self-elect: if $\mathsf{VRF}_{\mathsf{sk}_i}(i, j, e, \mathsf{coin}) < \mathsf{b}$ then compute proof ξ_i for $P_i \in \bar{\mathcal{C}}_j$.
   - If $P_i \in \bar{\mathcal{C}}_j$, input $\{\langle v_{j,k}\rangle_k\}$ to RECON. If RECON outputs x_j, multicast a vote $\mathsf{vote}_i = (\mathsf{vote}, \langle h_j\rangle_i, \varphi_{j,i}, \xi_i)$.
3. For $j \in \mathcal{C}$, upon receiving $t_s\kappa/n + 1$ valid votes from distinct P_k in $\bar{\mathcal{C}}_j$, form a certificate $c_j := (\mathsf{commit}, \langle h_j\rangle)$ and send it to all parties.
4. Upon receiving the certificate c_j for the input of party P_j, forward it to all parties.
5. Upon receiving a certificate c_j on for party P_j, input 1 to BA_j. After outputting 1 in at least $(1 - t_a/n)\kappa$ BAs, input 0 for the rest.
6. Set $\mathcal{S}$ to be the set of indices of the BA instances that delivered 1.
7. Output according to step 7 in Π_{ACS}, where the set in OC 1 has size $(1 - t_s/n)\kappa$ and the sets in OC 2 and OC 3 have size $(1 - t_a/n)\kappa$.

Fig. 9. ACS protocol from the perspective of party $P_{i\in\{1,\ldots,n\}}$ in epoch e.

Committee-based block agreement protocol Throughout the remainder of the section, we consider a network that is synchronous with up to $\hat{t}_s = (1-\epsilon)t_s$ corruptions, such that with high probability a committee of size κ will have up to $t_s\kappa/n$ corrupted members. Honest parties are assumed to input $(1 - t_s/n)\kappa$-quality pre-blocks of total length κ to the block agreement protocol.

We give an adaptively secure protocol BLA^{κ}, based on the binary Byzantine Agreement protocol from [2,1] and the block agreement protocol from [7], with several changes to achieve security against adaptive adversaries at a quadratic communication per pre-block. The high-level approach involves electing a leader who proposes an input among the ones sent by the parties, such that honest parties will commit on the same value. In our protocol, the proposal of inputs is performed before the leader election. Due to the forward secure signatures, the adversary cannot later corrupt the leader and cause them to equivocate.

First, parties encode their pre-blocks into codewords and distribute them, along with the hash, for future reconstruction and verification. The protocol is run for multiple rounds, and a leader is elected at each round. The parties commit on a value when they receive sufficient votes on that value, prioritizing votes with higher round numbers. In each round, a (different) committee is tasked with assembling a certificate, in order to have the certificate list $|C| = o(n)$. In a given round r, only votes from the current $\mathcal{C}_r$ are considered valid. $\Pi^{\kappa}_{\mathsf{BLA}}$ makes calls to a graded consensus protocol $\Pi^{\kappa}_{\mathsf{GC}}$, which makes a call to a $\mathsf{Propose}$ protocol $\Pi^{\kappa}_{\mathsf{Propose}}$. The details are given in Appendix D.

Communication complexity. $\Pi^{\kappa}_{\mathsf{ACS}}$ has communication complexity $O(\kappa n\ell + \kappa^2 n^2)$ communication and $\Pi^{\kappa}_{\mathsf{BLA}}$ has communication complexity $O(R\kappa^2 n^2 + \kappa n\ell)$, per input of size ℓ.

In $\mathit{\Pi}^{\kappa}_{\mathsf{SMR}}$, $\mathit{\Pi}^{\kappa}_{\mathsf{BLA}}$ and $\mathit{\Pi}^{\kappa}_{\mathsf{ACS}}$ are run on pre-blocks of size $O(L|\mathsf{tx}|)$. If the network is synchronous, the output is of size $O(L|\mathsf{tx}|)$, while if the network is asynchronous, the output is of size $O(\kappa L|\mathsf{tx}|)$. After decryption, since the transactions were randomly selected from honest parties buffers, with high probability, there will be $O(\kappa L)$ transactions in the output block.

For simplicity, we omit the $|\mathsf{tx}|$ factor in the following paragraph. $\mathit{\Pi}^{\kappa}_{\mathsf{SMR}}$ incurs $O(n^2L/(\kappa b)+n^2\kappa)$ total communication for step 1.3 and $O(n^2\kappa L/b+n^2\kappa^2)$ total communication in step 1.4. In step 2, $\mathit{\Pi}^{\kappa}_{\mathsf{BLA}}$ incurs $O(R\kappa^2n^2+\kappa n^2L/b+\kappa nL)$ total communication and $\mathit{\Pi}^{\kappa}_{\mathsf{ACS}}$ incurs $O(\kappa n^2L/b+\kappa nL+\kappa^2n^2)$ total communication.

Since $b=\hat{t}_s=O(n)$, $\mathsf{Upstate}$ incurs a total communication of $O(R\kappa^2n^2+\kappa nL|\mathsf{tx}|)$. This allows us to select a proposal sample size of $L=O(R\kappa n)$ and obtain a total communication of $O(R\kappa^2n^2)$ per transaction and an amortized communication complexity of $O(\kappa n)$ per block L.

In Appendix D, we describe an attack on $\mathsf{Upstate}$ from an adaptive adversary. Designing an adaptively secure SMR protocol in the network-agnostic case that achieves $O(n^2)$ communication per $O(n)$-block is a challenging open problem.

7 SMR under arbitrary network changes

We now consider a network that can arbitrarily transition between a synchronous and an asynchronous behavior. The adversary considered in this section is a *constrained epoch-mobile adaptive adversary*, who can corrupt at most t_s unique parties over the duration of the protocol, and can move between those t_s parties from epoch to epoch, as long as it does not exceed the t_a or t_s limit in any epoch or at any moment in time. In this model, parties' local machines may reboot in a way that cannot be prevented by the adversary, flushing the adversary out from the machine. Importantly, the state of the parties is not removed from the adversary's view after uncorruptions.

We will show that adding a reboot step at the beginning of each epoch to the network-agnostic protocols discussed so far, Update and $\mathsf{Upstate}$, as well as $\mathsf{Tardigrade}$, results in protocols that are secure under arbitrary network changes, for $n>2t_s+t_a$, $t_a\le t_s$, with at most t_s-t_a exposed keys in the asynchronous case, in the restricted epoch-mobile model. (For simplicity, we assume the reboot is instantaneous; otherwise we can adjust the timings of the steps.)

Theorem 5. *Protocols* Update*,* $\mathsf{Upstate}$*, and* $\mathsf{Tardigrade}$ *[7] with reboots at the onset of every epoch are secure under arbitrary network changes against a constrained epoch-mobile adaptive adversary, where $n>2t_s+t_a$, $t_a\le t_s$.*

We prove the first part of Theorem 5 below, after some technical observations. Proofs of the rest of Theorem 5 and of the technical Lemmata below are given in Appendix E.

Throughout, we use threshold cryptographic primitives with a threshold of t_s+1. Although the adversary has access to up to t_s keys/key shares, it cannot create full signatures or certificates on its own because these require at least

t_s+1 valid contributions; likewise, it cannot decrypt independently of the honest parties. Moreover, while forming commit or output certificates, honest parties only sign messages that they locally verified, such as a hash value whose associate input was correctly reconstructed, or the output of the Π_{ACS} protocol.

In all protocols we use the binary BA protocol from [6], which is also designed for a network-agnostic setting with $n > 2t_s + t_a$. It is signature-free, apart from a threshold cryptosystem with high threshold of $t_s + 1$ to compute the common coin and ensure termination. This ensures that even with t_s key exposures (but only t_a active corruptions), the protocol remains t_a-valid, t_a-consistent and t_a-terminating against an adaptive adversary.

Lemma 8. *In an execution of Π_{ACS}, if there are at most t_a corruptions and $t_s - t_a$ exposed parties, then at least $n - t_a$ BA instances will terminate with output 1.*

Lemma 9. *Suppose there are at most t_a corruptions and $t_s - t_a$ exposed parties during an execution of Π_{ACS}. Given a certificate for a party P, $(\mathsf{commit}, \langle h \rangle)$, all honest parties eventually reconstruct the same output.*

Lemma 10. *If there are at most t_a corruptions, there cannot be two valid certificates $(\mathsf{commit}, \langle h \rangle)$, $(\mathsf{commit}, \langle h' \rangle)$ associated with P such that $h \neq h'$.*

Proof. (Theorem 5, Update) When the network is only synchronous or only asynchronous, or there is a single asynchronous to synchronous transition, the proof follows directly from the security proof of Update in Section 5.

Suppose the network has undergone a transition from synchronous to asynchronous. In this case the adversary actively controls at most t_a parties, but may have exposed up to t_s parties. This means that each pre-block created by an actively corrupted party may contain up to t_s validly signed adversarial ciphertexts. However, exposed parties still act honestly, and so each pre-block created by an honest party contains at most t_a malicious ciphertexts. Because pre-block entries are received directly from the corresponding party, an honest party's $(n - t_s)$-quality pre-block will have at least $n - t_s - t_a$ honestly created and signed ciphertexts.

ACS. In Π_{ACS}, parties need to be able to reconstruct all values corresponding to the at least $n - t_a$ BA instances that terminated with output 1. The use of codewords makes the analysis slightly subtler, since the adversary can forge valid but bad codewords and distribute them in the multicast round of INDI as if they originated from the exposed parties. By Lemma 8, at least $n - t_a$ BA instances will still terminate, despite exposures. Coupled with Lemmata 9 and 10, which show there cannot be conflicting certificates and all honest parties are able to eventually correctly reconstruct the same input, it follows that Π_{ACS} achieves t_a-termination, t_a-set quality and t_a-consistency. Finally, t_s-validity with termination has the same proof as in Lemma 3.

BLA. There is a Leader mechanism in Π_{BLA} [7], that is obtained using a strict majority of parties. Hence it is still unpredictable in the presence of t_s exposed parties. The property required of Π_{BLA} in the asynchronous case is the following: if an honest party does output in Π_{BLA}, its output is a $(n-t_s)$-quality pre-block. Honest parties only validate and multicast $(n-t_s)$-quality blocks, so this property still holds.

SMR. A corrupted party can forge the signature of an exposed party when assembling its own $(n-t_s)$-quality pre-block. Therefore, up to t_a pre-blocks input to Π_{BLA} could have only $n-2t_s$ entries originating from honest parties. If such a block is output by Π_{BLA}, then the same holds for the the output of Π_{ACS}.

By t_a-consistency and t_a-validity with termination of Π_{ACS}, all honest parties output the same set of pre-blocks. As a result, at least $n-t_a > t_s$ parties contribute valid decryption shares, and so every honest party is able to reconstruct the same block. Therefore, Update SMR is t_a-consistent and t_a-complete.

Next, we argue t_a-liveness holds. If an adversarial pre-block is output by ACS, only $n-2t_s$ honest parties are guaranteed to remove L/n transactions in a given epoch e. Thus, the presence of key exposures increases the number of epochs needed for tx to move to the front of sufficiently many honest parties' buffers (see Appendix B). However, tx will still eventually move to the front of enough parties' buffers to ensure that it is eventually output (with probability increasing with the number of epochs).

External validity follows from consistency of Π_{ACS}, since a threshold of t_s+1 is used in the validity certificates over the block hashes.

Finally, the adversary cannot break the liveness of the protocol by erasing threshold key shares of the corrupted parties: any t_s+1 shares can be used to reconstruct, so in order to prevent reconstruction, the adversary would need to erase at least $n-t_s-t_a$ shares. But this would require the adversary to corrupt more than t_s parties over the duration of the protocol, since $2t_s+t_a<n$. We conclude that security is preserved even across multiple network transitions. □

8 Asynchronous proactive secret sharing

Throughout most of this section we consider an asynchronous network in the presence of a mobile adaptive adversary. At the end of the section, we extend the analysis to the setting with changing network conditions.

In each epoch, the adversary is limited to t_a corruptions, but those t_a corruptions need not target the same parties in each epoch. Thus, over multiple epochs, the adversary could have controlled more than t_a+1 different parties. While a party is corrupted, its current epoch is considered to be undefined, since it can behave arbitrarily. Upon becoming uncorrupted, a party's local epoch number is considered to be the epoch in which it was originally corrupted. We refer to the parties that are not corrupted as *honest* (in that epoch).

Here, we use an additional assumption of secure (authenticated private) channels, implemented using a pairwise shared key inaccessible to the adversary (e.g., stored in secure hardware). We show that even with secure channels, it is impossible to have a proactive asynchronous protocol without making assumptions on epoch length (as in [9] where epochs are defined to take place between clock ticks) but with epochs determined by a successful reshare of the secret (as in [32] but there the network is partially synchronous). While Cachin *et al.* [9] briefly remark upon this impossibility before making the assumption of clock ticks and "asynchronous proactive channels", we fully model and prove this result.

Definition 5. *A* (t_a+1)*-out-of-n* proactive verifiable secret sharing scheme with reshare *is defined by an algorithm* Share *and protocols* $\mathsf{Reshare}, \mathsf{Reconstruct}$ *that satisfy the following:*

- Share *takes as input a secret* $s \in \mathbb{F}$ *and outputs shares* $(s_1^{(0)}, \ldots, s_n^{(0)})$*. Party* P_i*,* $i = 1, \ldots, n$ *is given* $s_i^{(0)}$ *and sets its epoch number to 0.*
- $\mathsf{Reshare}$ *is an interactive protocol run by a subset of parties* $\mathcal{S}$ *of size at least* $n - t_a$ *that takes as input an epoch number* τ*, a set of shares associated to that epoch number consisting of the share of each of the parties in* $\mathcal{S}$*:* $(s_{i_1}^{(\tau)}, \ldots, s_{i_{|\mathcal{S}|}}^{(\tau)})$ *and outputs to every party* P_i*,* $i \in [n]$ *a new share* $s_i^{(\tau+1)}$ *or an error symbol* $\perp$*. A party* P_i *that receives output from* $\mathsf{Reshare}$ *with associated epoch* τ *sets its epoch number to* $\tau + 1$*.*
- $\mathsf{Reconstruct}$ *is an interactive protocol run by a subset of parties* $\mathcal{S}$ *of size at least* $n{-}t_a$*, that takes as input a epoch number* τ*, a set of shares* $(s_{i_1}^{(\tau)}, \ldots, s_{i_{|\mathcal{S}|}}^{(\tau)})$ *and outputs to all parties either a value* $s' \in \mathbb{F}$ *or an error symbol* $\perp$*.*

An honest party is said to *complete* $\mathsf{Share}, \mathsf{Reshare}$*, or* $\mathsf{Reconstruct}$ *in epoch* τ when they generate the corresponding output from the algorithm in epoch τ.

For completeness, we give a standard privacy game between a challenger and an adversary $\mathcal{A}$ in Appendix F where the goal of the adversary is to learn the secret. The advantage of the adversary is denoted by $\mathsf{Adv}(\mathcal{A})$.

Definition 6. *A proactive verifiable secret sharing scheme with reshare is* secure *against a* t_a*-limited adversary if it satisfies the following:*

- *(Privacy):* $\mathsf{Adv}(\mathcal{A})$ *is negligible.*
- *(Correctness): For any* $s \in \mathbb{F}$*, conditioned on the adversary eventually delivering all messages between honest parties, it holds that: if during any epoch* τ*, a set* $\mathcal{S}$ *of least* $n - t_a$ *honest parties locally call* $\mathsf{Reconstruct}$ *on epoch number* τ *and local shares associated with* τ*, they obtain the initially shared secret:* $\mathsf{Reconstruct}(\tau, \{s_i^{(\tau)}\}_{i \in \mathcal{S}}) = \mathsf{Reconstruct}(\mathsf{Share}(s))$*. Furthermore, all parties in* $\mathcal{S}$ *proceed to epoch* $\tau + 1$*.*
- *(Liveness): For any epoch number* $\tau \geq 0$*, if an honest party has reached epoch* τ*, i.e., has obtained output from the* $\mathsf{Reshare}$ *protocol associated to epoch* $\tau - 1$*, then all honest parties will eventually reach a epoch number* $\tau' \geq \tau$*, provided the adversary delivers all messages sent between honest parties so far and the responses triggered by these messages.*

In verifiable secret sharing, in order to achieve correctness, $\mathsf{Share}, \mathsf{Reshare}$ and $\mathsf{Reconstruct}$ need to implicitly have validation procedures of the inputs.

Theorem 6. *There does not exist a secure asynchronous* (t_a+1)*-out-of-n proactive verifiable secret sharing scheme with reshare.*

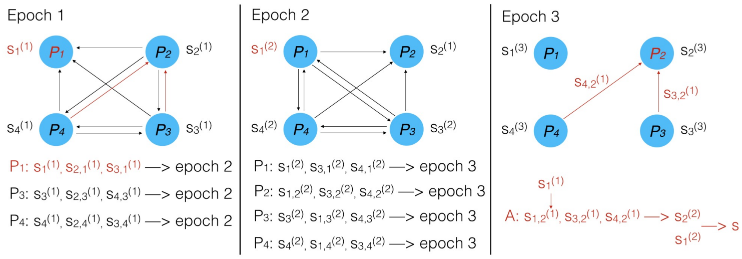

Epoch 1
$s_1^{(1)}$
P_1
P_2
$s_2^{(1)}$
$s_4^{(1)}$
P_4
P_3
$s_3^{(1)}$
P1: $s_1^{(1)}, s_{2,1}^{(1)}, s_{3,1}^{(1)}$ —> epoch 2
P3: $s_3^{(1)}, s_{2,3}^{(1)}, s_{4,3}^{(1)}$ —> epoch 2
P4: $s_4^{(1)}, s_{2,4}^{(1)}, s_{3,4}^{(1)}$ —> epoch 2
Epoch 2
$s_1^{(2)}$
P_1
P_2
$s_2^{(1)}$
$s_4^{(2)}$
P_4
P_3
$s_3^{(2)}$
P1: $s_1^{(2)}, s_{3,1}^{(2)}, s_{4,1}^{(2)}$ —> epoch 3
P2: $s_{1,2}^{(2)}, s_{3,2}^{(2)}, s_{4,2}^{(2)}$ —> epoch 3
P3: $s_3^{(2)}, s_{1,3}^{(2)}, s_{4,3}^{(2)}$ —> epoch 3
P4: $s_4^{(2)}, s_{1,4}^{(2)}, s_{3,4}^{(2)}$ —> epoch 3
Epoch 3
$s_1^{(3)}$
P_1
P_2
$s_2^{(3)}$
$s_{4,2}^{(1)}$
$s_{3,2}^{(1)}$
$s_4^{(3)}$
P_4
P_3
$s_3^{(3)}$
$s_1^{(1)}$
A: $s_{1,2}^{(1)}, s_{3,2}^{(1)}, s_{4,2}^{(1)}$ —> $s_2^{(2)}$
—> s
$s_1^{(2)}$

Fig. 10. We denote by $s_{j,i}^{(\tau)}$ the intermediate share obtained by party P_i from party P_j in epoch τ. Party P_j can construct its share for the next epoch $s_j^{(\tau+1)}$ from at least $n - t_a$ intermediate values $s_{j,i}^{(\tau)}$. The red quantities are in the view of the adversary. The red edges represent delayed messages from epoch 1 delivered in epoch 3.

Proof. Our goal is to show that an adversary can break privacy by amassing shares corresponding to $t_a + 1$ parties in a single epoch τ. Then, we prove that protocols which avoid the prior attack do not satisfy liveness.

I. Non-interactive Reshare protocol. Consider $n = 4$ and $t_a = 1$. This counterexample is depicted in Figure 10 and can be extended to any arbitrary n and corruption threshold $t_a < n/3$.

The adversary corrupts party P_1 in epoch 1. At this point in time, the adversary knows the state of P_1, which includes the share $s_1^{(1)}$. Each honest party locally initiates the Reshare protocol at the onset of epoch 1. The adversary instructs P_1 not to deliver any message and delivers all the following messages: from P_2 to all other parties, from P_3 only to P_1 and P_4, and from P_4 only to P_1 and P_3. The parties P_3, P_4 thus obtain sufficient information to construct their shares $s_1^{(2)}$, $s_3^{(2)}$, $s_4^{(2)}$ and advance to epoch 2. However, P_2 remains in epoch 1. The adversary uncorrupts party P_1 after Reshare was completed. At this point in time, the view of the adversary includes $s_1^{(1)}$ and the intermediate shares for $s_1^{(2)}$. The adversary allows P_1 to also advance to epoch 2.

At the onset of epoch 2, each honest party locally initiates the Reshare protocol. The adversary delivers all messages between parties. This enables all parties to obtain their corresponding share $s_1^{(3)}$, $s_2^{(3)}$, $s_3^{(3)}$, $s_4^{(3)}$, and advance to epoch 3.

At the onset of epoch 3, the adversary corrupts party P_2 and delivers the messages originated in epoch 1 from P_3 and P_4 and destined to P_2. Using $s_1^{(1)}$, the adversary now has 3 messages and is able to obtain $s_2^{(2)}$. Hence, the adversary can reconstruct s from two correct shares in epoch 2: $s_1^{(2)}, s_2^{(2)}$, without corrupting more than $t_a = 1$ party per epoch.

Restarting and flushing the adversary out does not prevent the above attack, since there is no synchronizing signal in the secure co-processor of a corrupted party instructing it to restart before the first Reshare is completed. This specific issue can be addressed using erasures and/or interaction; however, we show that protocols that avoid this issue are necessarily not live.

II. Interactive Reshare protocol. Consider a generic interactive Reshare protocol where two parties, P_i and P_j, start an epoch with $s_i^{(\tau)}$ and $s_j^{(\tau)}$, respectively. After r rounds of communication, P_i obtains $s_{j,i}^{(\tau)}$ and P_j obtains $s_{i,j}^{(\tau)}$.

If only one of the r messages is useful for computing the new share, then the attack in I. can be modified to still break privacy. If more than one of the r messages are needed for computing the new share, and honest parties erase their previous state when transitioning to a new epoch (also implying they do not respond to messages originated from previous epochs), then the above attack does not break privacy. But such an interactive asynchronous protocol where parties can only advance to the next epoch after repeated interactions does not achieve liveness, as shown next.

Consider now that the adversary delays all messages destined to P_1, hence keeping it in epoch 1, while allowing the rest of the parties to progress an arbitrarily large number of epochs τ. At this point, the adversary delivers all messages that were sent so far, including the messages originated at P_1 as response to the received messages. However, since obtaining the output of any Reshare requires interaction and the other honest parties do not respond to messages originated in previous epochs in order to preserve privacy, a party P_1 cannot reach a subsequent epoch based only on the messages sent so far, breaking liveness. □

The attack above hinges on the fact that a party can still retrieve in epoch $\tau' > \tau$ the contents of a message sent to it in epoch τ. Both privacy and liveness would be maintained if parties had access to "setup-free asynchronous forward-secure channels" with the following properties:

1. A message sent in epoch τ can only be read in epoch τ;
2. At the onset of epoch $\tau + 1$, the sender and receiver on that channel have access to the new secret and public key, respectively, i.e., the adversary does not control the delivery of this information (it should not be interactive);
3. Messages in different epochs are encrypted with different keys.

Secure co-processors using forward secure encryption do not seem sufficient to implement this kind of channel. Say a party P_1 was delayed and is still in phase τ, and all other parties advanced to an epoch $\tau' > \tau$, so they updated their channel keys. But when honest parties start a new Reshare, they cannot use the key associated to P_1's phase τ, because an adversary corrupting P_1 in phase τ would learn shares from phase τ' and break privacy. These are points 1 and 3. So until the adversary delivers the messages from phase τ, P_1 is stuck, but this does not break liveness if the protocol is non-interactive. However, if point 2 is satisfied, the other parties need to already have the public key in the channel for phase $\tau + 1$, otherwise the impossibility proof for interactive protocols would apply. But a forward secure with unique public key alows a ciphertext encrypted at epoch $\tau + 1$ to be decrypted at epoch τ, so privacy is broken.

Note that in [9], the transition between epochs is *external*, triggered by a clock tick, and can happen even if a party did not complete the Reshare protocol in the current epoch. This allows parties to rely on the clock tick event to set new channel keys in a synchronized way.

Proactive secret sharing under network changes. We again consider a network that can arbitrarily switch between synchronous and asynchronous cases and $n > 2t_s + t_a$, $t_a \leq n/3$, $t_s \leq n/2$. Note that in this setting, the $\mathsf{Reconstruct}$ threshold is at least $t_s + 1$ and the $\mathsf{Reshare}$ threshold is $n - t_s$ in order to satisfy privacy in case the network is synchronous.

Corollary 1. *There does not exist a secure (t_s, t_a)-proactive verifiable secret sharing scheme with reshare under arbitrary network transitions.*

Proof. Assume the network in an asynchronous state. This allows the adversary to corrupt up to t_a parties in the same local epoch. The arguments in the proof of Theorem 6 still hold. For the privacy attack, the adversary delays the messages in epoch τ towards $t_s - t_a + 1$ honest parties, until the epoch(s) it corrupts these parties (if $t_s \geq 2t_a$, it needs more epochs to corrupt all $t_s - t_a + 1$ parties), while allowing the rest of the parties to complete the refresh in all epochs, i.e., deliver and receive at least $n - t_s$ share messages. For the interactive liveness attack, the adversary can still cause the parties to be arbitrarily far apart. □

We remark that the clock ticks used in Π_{SMR} (Section 5) to start an epoch are not the same as the ones assumed in [9]. In our model, the epoch started at T_e does not necessarily finish by T_{e+1}, and can continue in the background, so liveness could be lost if all parties would erase their key shares at T_{e+1}.

References

1. I. Abraham, T.-H. H. Chan, D. Dolev, K. Nayak, R. Pass, L. Ren, and E. Shi. Communication complexity of byzantine agreement, revisited. In P. Robinson and F. Ellen, editors, *38th ACM PODC*, pages 317–326. ACM, July / Aug. 2019.
2. 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 I. Goldberg and T. Moore, editors, *FC 2019*, volume 11598 of *LNCS*, pages 320–334. Springer, Heidelberg, Feb. 2019.
3. I. Abraham, D. Malkhi, K. Nayak, L. Ren, and M. Yin. Sync HotStuff: Simple and practical synchronous state machine replication. In *2020 IEEE Symposium on Security and Privacy*, pages 106–118. IEEE Computer Society Press, May 2020.
4. A. Appan, A. Chandramouli, and A. Choudhury. Perfectly-secure synchronous MPC with asynchronous fallback guarantees. Cryptology ePrint Archive, Report 2022/109, 2022. `https://eprint.iacr.org/2022/109`.
5. 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 R. Pass and K. Pietrzak, editors, *TCC 2020, Part I*, volume 12550 of *LNCS*, pages 260–290. Springer, Heidelberg, Nov. 2020.
6. E. Blum, J. Katz, and J. Loss. Synchronous consensus with optimal asynchronous fallback guarantees. In D. Hofheinz and A. Rosen, editors, *TCC 2019, Part I*, volume 11891 of *LNCS*, pages 131–150. Springer, Heidelberg, Dec. 2019.
7. E. Blum, J. Katz, and J. Loss. Tardigrade: An atomic broadcast protocol for arbitrary network conditions. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 547–572. Springer, 2021.

8. E. Blum, C.-D. L. Zhang, and J. Loss. Always have a backup plan: Fully secure synchronous MPC with asynchronous fallback. In D. Micciancio and T. Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 707–731. Springer, Heidelberg, Aug. 2020.
9. C. Cachin, K. Kursawe, A. Lysyanskaya, and R. Strobl. Asynchronous verifiable secret sharing and proactive cryptosystems. In V. Atluri, editor, *ACM CCS 2002*, pages 88–97. ACM Press, Nov. 2002.
10. C. Cachin and J. A. Poritz. Secure intrusion-tolerant replication on the internet. In *Proceedings International Conference on Dependable Systems and Networks*, pages 167–176. IEEE, 2002.
11. R. Canetti, R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Adaptive security for threshold cryptosystems. In M. J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 98–115. Springer, Heidelberg, Aug. 1999.
12. M. Castro and B. Liskov. Proactive recovery in a Byzantine-Fault-Tolerant system. In *Fourth Symposium on Operating Systems Design and Implementation (OSDI 2000)*, 2000.
13. M. Castro and B. Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems (TOCS)*, 20(4):398–461, 2002.
14. S. Das, Z. Xiang, and L. Ren. Balanced quadratic reliable broadcast and improved asynchronous verifiable information dispersal. Cryptology ePrint Archive, Report 2022/052, 2022. `https://eprint.iacr.org/2022/052`.
15. S. Duan, M. K. Reiter, and H. Zhang. BEAT: Asynchronous BFT made practical. In D. Lie, M. Mannan, M. Backes, and X. Wang, editors, *ACM CCS 2018*, pages 2028–2041. ACM Press, Oct. 2018.
16. Y. Frankel, P. D. MacKenzie, and M. Yung. Adaptively-secure optimal-resilience proactive RSA. In K.-Y. Lam, E. Okamoto, and C. Xing, editors, *ASIACRYPT'99*, volume 1716 of *LNCS*, pages 180–194. Springer, Heidelberg, Nov. 1999.
17. D. Ghinea, C.-D. Liu-Zhang, and R. Wattenhofer. Optimal synchronous approximate agreement with asynchronous fallback. Cryptology ePrint Archive, Report 2022/354, 2022. `https://eprint.iacr.org/2022/354`.
18. Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, pages 51–68, New York, NY, USA, 2017. Association for Computing Machinery.
19. S. D. Gordon, J. Katz, R. Kumaresan, and A. Yerukhimovich. Authenticated broadcast with a partially compromised public-key infrastructure. *Information and Computation*, 234:17–25, 2014.
20. J. Groth. Non-interactive distributed key generation and key resharing. Cryptology ePrint Archive, Report 2021/339, 2021. `https://ia.cr/2021/339`.
21. B. Guo, Z. Lu, Q. Tang, J. Xu, and Z. Zhang. Dumbo: Faster asynchronous BFT protocols. In J. Ligatti, X. Ou, J. Katz, and G. Vigna, editors, *ACM CCS 2020*, pages 803–818. ACM Press, Nov. 2020.
22. A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung. Proactive secret sharing or: How to cope with perpetual leakage. In D. Coppersmith, editor, *CRYPTO'95*, volume 963 of *LNCS*, pages 339–352. Springer, Heidelberg, Aug. 1995.
23. I. Keidar, E. Kokoris-Kogias, O. Naor, and A. Spiegelman. All you need is dag. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, pages 165–175, 2021.
24. K. Kursawe and V. Shoup. Optimistic asynchronous atomic broadcast. In L. Caires, G. F. Italiano, L. Monteiro, C. Palamidessi, and M. Yung, editors, *ICALP 2005*, volume 3580 of *LNCS*, pages 204–215. Springer, Heidelberg, July 2005.

25. C. Liu, S. Duan, and H. Zhang. Epic: Efficient asynchronous bft with adaptive security. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 437–451. IEEE, 2020.
26. Y. Lu, Z. Lu, Q. Tang, and G. Wang. Dumbo-MVBA: Optimal multi-valued validated asynchronous byzantine agreement, revisited. In Y. Emek and C. Cachin, editors, *39th ACM PODC*, pages 129–138. ACM, Aug. 2020.
27. S. K. D. Maram, F. Zhang, L. Wang, A. Low, Y. Zhang, A. Juels, and D. Song. CHURP: Dynamic-committee proactive secret sharing. In L. Cavallaro, J. Kinder, X. Wang, and J. Katz, editors, *ACM CCS 2019*, pages 2369–2386. ACM Press, Nov. 2019.
28. A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song. The honey badger of BFT protocols. In E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, editors, *ACM CCS 2016*, pages 31–42. ACM Press, Oct. 2016.
29. A. Momose and L. Ren. Multi-threshold byzantine fault tolerance. In G. Vigna and E. Shi, editors, *ACM CCS 2021*, pages 1686–1699. ACM Press, Nov. 2021.
30. K. Nayak, L. Ren, E. Shi, N. H. Vaidya, and Z. Xiang. Improved extension protocols for byzantine broadcast and agreement. In *34th International Symposium on Distributed Computing (DISC 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.
31. M. Rambaud and A. Urban. Asynchronous dynamic proactive secret sharing under honest majority: Refreshing without a consistent view on shares. Cryptology ePrint Archive, Paper 2022/619, 2022. `https://eprint.iacr.org/2022/619`.
32. D. A. Schultz, B. Liskov, and M. Liskov. Mobile proactive secret sharing. In *Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing*, pages 458–458, 2008.
33. R. Vassantlal, E. Alchieri, B. Ferreira, and A. Bessani. Cobra: Dynamic proactive secret sharing for confidential bft services. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1528–1528. IEEE Computer Society, 2022.

A Preliminaries

Threshold signature scheme Setup: $\mathsf{pp} \leftarrow \mathsf{TS.Setup}(1^\kappa, n, t)$. Given the security parameter, the number of participants and threshold, output public parameters pp.

Key generation: $(\mathsf{pk}, \mathsf{SK}, \mathsf{VK}) \leftarrow \mathsf{TS.KeyGen}(\mathsf{pp})$. This is an interactive protocol where the outcome is a public key pk, a vector of private keys $\mathsf{SK} = (\mathsf{sk}_1, \dots, \mathsf{sk}_n)$, where party P_i obtains only sk_i and a public vector of verification keys $\mathsf{VK} = (\mathsf{vk}_1, \dots, \mathsf{vk}_n)$. Denote by $\mathsf{PK} = (\mathsf{pk}, \mathsf{VK})$.

Partial signing: $\sigma_i \leftarrow \mathsf{TS.Sign}(\mathsf{PK}, \mathsf{sk}_i, M)$. This algorithm takes in the public keys PK, a message M and a private key share sk_i and outputs a signature share σ_i.

Signature share verification: $b_i \leftarrow \mathsf{TS.ShVer}(\mathsf{PK}, M, i, \sigma_i)$. This algorithm takes in the public keys PK, a message M and a partial signature on that message and outputs 1 is the partial signature is valid and 0 otherwise.

Share combination: $\sigma \leftarrow \mathsf{TS.Combine}(\mathsf{PK}, M, (i, \sigma_i)_{i \in S})$. On input the public keys PK, the message M and a set of shares of size $|S| = t + 1$, this algorithm outputs a full signature σ if $\mathsf{TS.ShVer}(\mathsf{pk}, \mathsf{VK}, m, i, \sigma_i)$ returned 1 for all shares and outputs $\perp$ otherwise.

Verification: $b \leftarrow \mathsf{TS.Verify}(\mathsf{PK}, M, \sigma)$. Anyone can verify a signature σ for on message M under public keys PK by running the verification algorithm, which returns 1 to indicate that the signature is valid and 0 otherwise.

Cryptographic sortition Key generation: $(vk, sk) \leftarrow \mathsf{VRF.KeyGen}(\mathsf{pp})$. Given input pp, the key generation algorithm outputs a verification key vk and a secret key sk.

Evaluation: $(\rho, \pi) \leftarrow \mathsf{VRF.Eval}(\mathsf{pp}, sk, M)$. Given a secret key sk and message M, the evaluation algorithm outputs a value ρ and proof π.

Verification: $b \leftarrow \mathsf{VRF.Verify}(\mathsf{pp}, vk, x, \rho, \pi)$. Given a verification key vk, input M, output ρ and proof π, the verification algorithm outputs 1 if the output ρ and proof π are valid with respect to vk and x and outputs 0 otherwise.

B Liveness bound

The following lemma is based on the pigeon-hole principle.

Lemma 11. *Assume there are at most t_y corrupted parties. If there are $n-t_s-t_x$ honest entries in a pre-block output in any given epoch, there exists at least*

$$M_\alpha < 1 + (n - t_s - t_x)\frac{1 - \alpha + \frac{n-t_y}{e(n-t_x-t_s)}}{1 - \alpha\frac{n-t_s-t_x}{n-t_y} + \frac{1}{r}},$$

honest parties who output more than $\alpha r\frac{n-t_s-t_x}{n-t_y}$ times over r epochs for $0 < \alpha \leq 1$, and at least

$$M_0 \leq n - t_x - t_s$$

honest parties who output more than once over r epochs.

If the network is synchronous, we have $t_x = 0$ and $t_y = t_s$. If the network is asynchronous, we have $t_x = t_y = t_a$ if there are no key exposures, and $t_x = t_y = t_s$ if there are key exposures.

Proof. Assume this does not hold, i.e., for $0 < \alpha \leq 1$ and M_α in the statement, there do not exist M_α honest parties that output at least $\alpha r\frac{n-t_s-t_x}{n-t_y}$ times over r epochs. Then, in expectation, there exist at least $n-t_y-M_\alpha+1$ honest parties that output less than $\alpha r\frac{n-t_s-t_x}{n-t_y}$ times over r epochs. Then the number of honest entries output over r epochs will be bounded below and above by:

$$\begin{aligned} r(n - t_s - t_x) &\leq (n - t_y - M_\alpha + 1)\left(\alpha r\frac{n - t_s - t_x}{n - t_y} - 1\right) + r(M_\alpha - 1) \\ M_\alpha &\geq 1 + (n - t_s - t_x)\frac{1 - \alpha + \frac{n-t_y}{e(n-t_x-t_s)}}{1 - \alpha\frac{n-t_s-t_x}{n-t_y} + \frac{1}{r}}, \end{aligned} \tag{1}$$

which contradicts the assumption at the beginning of the proof.

Consider $\alpha = 0$ and there are only M_0-1 parties that ever output. This means that there can be at most $r(M_0-1)$ outputs. Therefore $r(M_\alpha-1) \geq r(n-t_s-t_x)$, which contradicts the statement of the lemma. □

Lemma 11 implies that at least $n - t_x - t_s$ honest parties output at least once over r epochs. Moreover, we always have at least one honest party consistently outputting over r epochs, since

$$\alpha \leq \min \left\{ 1, \frac{1 + (n - t_y - 1)/r(n - t_s - t_x)}{1 - 1/(n - t_y)} \right\} = 1, \tag{2}$$

as long as $n > \max\{t_s + t_x, t_y\}$, which always happens in our settings.

A similar argument follows for the committee-based protocols with $\hat{t}_s = (1 - \epsilon)t_s$ and $\hat{t}_a = (1 - \epsilon)t_a$ corruptions.

C Probability bounds for the committees

Lemma 12. *[Chernoff's inequalities] Let $X_1, X_2, \ldots, X_n$ be independent random binary variables such that, for $1 \leq i \leq n$, $\mathbb{P}[X_i = 1] =: p_i$. Then, for $X := \sum_{i=1}^{n} X_i$ and $\mu := \mathbb{E}[X] = \sum_{i=1}^{n} p_i$:*

$$\mathbb{P}[X \geq (1 + \delta)\mu] \leq e^{-\frac{\delta^2 \mu}{\delta + 2}}, \qquad 0 < \delta, \tag{3}$$

$$\mathbb{P}[X \leq (1 - \delta)\mu] \leq e^{-\frac{\delta^2 \mu}{2}}, \qquad 0 < \delta < 1. \tag{4}$$

Lemma 13. *[Hoeffding's inequality for binomial distribution] Let X be a variable sampled from a binomial distribution with n independent trials and probability of success p. Then:*

$$\mathbb{P}[X \geq k + np] \leq e^{-2k^2/n}. \tag{5}$$

Let $X_1, \ldots, X_n$ be random binary variables sampled with probability t/n and $Y_1, \ldots, Y_n$ be random binary variables sampled with probability $1 - t/n$. We want to bound the probability that $X = \sum_{i=1}^{\kappa} X_i$ is greater than a value s, and the probability that $Y = \sum_{i=1}^{\kappa} Y_i$ is less than a value s.

$$\mathbb{P}[X \geq s] \leq e^{-\frac{(s - t\kappa/n)^2}{s + 2 - t\kappa/n}}, \text{if } s > t\kappa/n, \text{ by (3)}. \tag{6}$$

$$\mathbb{P}[Y \leq s] \leq e^{-\frac{\kappa(1 - t/n - s/\kappa)^2}{1 - t/n}}, \text{if } 0 < s < (1 - t/n)\kappa, \text{ by (4)}. \tag{7}$$

C.1 Unique committee

In the ACS and BLA protocol, the committee $\mathcal{C}$ of size κ is selected by using a collision-resistant hash function on unpredictable inputs. Hence, the event of choosing a particular party to be part of the committee can be thought of as an independent uniform at random sampling procedure.

Let Y_s denote the number of honest parties among the κ randomly elected committee members when the number of corrupted parties is $(1 - \epsilon)t_s$. The probability that there are fewer than $t_s\kappa/n$ honest parties in the committee is bounded by

$$\mathbb{P}[Y_s \leq \kappa t_s/n] \leq e^{-\frac{\kappa(1 - (2 - \epsilon)t_s/n)^2}{1 - (1 - \epsilon)t_s/n}}. \tag{8}$$

Since $t_s/n \leq 1/2$, we get $\mathbb{P}[Y_s \leq \kappa/2] \leq e^{-\frac{\epsilon^2\kappa}{1+\epsilon}}$.

The probability that there are fewer than $(1 - t_s/n)\kappa$ honest parties in the committee is bounded by

$$\mathbb{P}[Y_s \leq (1 - t_s/n)\kappa] \leq e^{-\frac{\kappa(\epsilon t_s/n)^2}{1-(1-\epsilon)t_s/n}}. \tag{9}$$

Since $t_s/n \leq 1/2$, we get $\mathbb{P}[Y_s \leq \kappa/2] \leq e^{-\frac{\epsilon^2\kappa}{2+2\epsilon}}$.

The probability that there are more than $t_a\kappa/n$ actively corrupted parties in the committee when the threshold of corruptions is $(1 - \epsilon)t_a$ is bounded by

$$\mathbb{P}[X_a \geq \kappa t_a/n] \leq e^{-\frac{(\epsilon\kappa t_a/n)^2}{2-\epsilon\kappa t_a/n}}. \tag{10}$$

Since $t_a/n \leq 1/3$, we get $\mathbb{P}[X_a \geq \kappa/3] \leq e^{-\frac{\epsilon^2\kappa^2}{3(6-\epsilon\kappa)}}$.

The probability that there are more than $t_s\kappa/n$ exposed parties in the committee when the threshold of corruptions is $(1 - \epsilon)t_s$ is bounded by

$$\mathbb{P}[X_s \geq \kappa t_s/n] \leq e^{-\frac{(\epsilon\kappa t_s/n)^2}{2+\epsilon\kappa t_s/n}}. \tag{11}$$

Since $t_s/n \leq 1/2$, we get $\mathbb{P}[X_s \geq \kappa/2] \leq e^{-\frac{\epsilon^2\kappa^2}{2(4+\epsilon\kappa)}}$.

C.2 Self-elected committees

The committees obtained through self-election have in expectation κ members, obtained using a VRF with output length of κ bits and bound $\mathsf{b} = \kappa 2^\kappa/n$. Applying the VRF can be idealized as flipping coins for each party to determine whether they (κ of them) are in the committee or not.

Let Z denote the number of parties selected in the secondary committee $\bar{\mathcal{C}}$. The expected value of Z is $\mathbb{E}[Z] = \kappa$. By (4), the probability that committee $\bar{\mathcal{C}}$ has strictly fewer than $(1 - \epsilon)\kappa + 1$ members is:

$$\mathbb{P}[Z \leq (1 - \epsilon)\kappa] \leq e^{-\epsilon^2\kappa/2}. \tag{12}$$

By (3), the probability that committee $\bar{\mathcal{C}}$ has more than $(1 + \epsilon)\kappa$ members is:

$$\mathbb{P}[Z \geq (1 + \epsilon)\kappa] \leq e^{-\epsilon^2\kappa/(2+\epsilon)}. \tag{13}$$

Let Y_s denote the number of honest parties among the randomly elected secondary committee members. In expectation, the number of honest parties selected will be greater than the initial fraction of honest parties times the committee size: $\mathbb{E}[Y_s] \geq (1 - (1 - \epsilon)t_s/n)\kappa$, so equation (8) holds for the secondary committee as well. Analogously, equation (11) holds for X_a, the number of corrupted parties among the randomly elected secondary committee members when the initial corruption threshold is t_a.

Denote by W the number of committees that have more than $\kappa t_s/n + 1$ corrupted/exposed members. Note that because the selection of the secondary committees is independent (also of the corruption selection), W is a binomial variable with probability of success $p := P[X \geq \kappa t_s/n]$ out of κ independent trials. We are interested in bounding the probability of W being more than $t_a\kappa/n$, which is the cumulative distribution function of a binomial random variable with parameters (p, κ). Using the Hoeffding inequality (5), we obtain:

$$\mathbb{P}[W \geq t_a\kappa/n] \leq e^{-2\kappa(t_a/n-p)^2}. \tag{14}$$

For $t_a/n \leq 1/3$ we get $\mathbb{P}[W \geq \kappa/3] \approx e^{-2\kappa(1/3-e^{-\epsilon^2\kappa^2/2(4+\epsilon\kappa)})^2}$.

D Technical details on protocols

D.1 Update SMR proofs

Proof. (Theorem 1) We start with (1). Say a honest party P output a valid block in epoch e. Then P must have generated output in Π_{ACS} in epoch e, call it B, and at least $t_s + 1$ decryption shares on B were gathered. By t_s-validity with termination of Π_{ACS}, all honest parties will output B if they started Π_{ACS} with a valid pre-block B, so to prove t_s-consistency of Π_{SMR}, it remains to show that all honest parties input the same B to Π_{ACS}. Since the network is synchronous, by time $T_e + \Delta$, all honest parties have managed to assemble a $(n - t_s)$-quality pre-block B_i^e and input it to Π_{BLA}, which is t_s-terminating, t_s-valid and t_s-consistent, so it terminates by time $T_e + (5R + 1)\Delta$ with the same valid output B. Finally, t_s-completeness follows from t_s-validity with termination of Π_{ACS}.

We now address (2). Say a honest party P output a valid block in epoch e. P must have generated output in Π_{ACS} in epoch e, call it B, and gathered at least $t_s + 1$ decryption shares on B. By t_a-consistency of Π_{ACS}, all honest parties should have generated B in epoch e, so this proves t_a-consistency of Π_{SMR}. Every honest party will eventually assemble a valid $n - t_s$-quality pre-block B_i^e, either as an output of Π_{BLA} if it terminates, or by waiting until $n - t_s$ codewords multicast by honest parties are delivered for at least $n - t_s$ parties. By t_a-consistency and t_a-termination of Π_{ACS}, all honest parties will output the same pre-block B in epoch e, and therefore there are at least $t_s + 1 \leq n - t_a$ valid decryption shares (for the same B). This ensures each honest party successfully recovers a block, proving t_a-completeness of Π_{SMR}. □

Proof. (Theorem 2) By Theorem 1, all honest parties will output the same valid block, obtained by decrypting the output of Π_{ACS}, which means that they have valid certificates of correct decryption. External validity follows from the fact that the adversary cannot generate invalid certificates because it controls fewer than $t_s + 1$ parties. □

Proof. (Theorem 3) Assume all honest parties (at least $n - t_s$ in (1) and at least $n - t_a$ in (2)) have received a transaction tx. If by some epoch e, tx is not in an honest party's buffer anymore it means it was output in $\mathsf{blocks}[e']$ for $e' < e$. Then, by consistency of Π_{SMR} proven in Theorem 1, tx will not be in any honest party's buffer after epoch e'. Otherwise, suppose tx is still in an honest party P's buffer at epoch e. By completeness of Π_{SMR} proven in Theorem 1, each party outputs a block in every epoch. This block is obtained by decrypting a pre-block of $(n - t_s)$-set quality to which at least $n - t_s - t_a$ honest parties contributed L/n transactions, by Lemma 14 proved below. Thus, each honest party that contributes removes in expectation at least L/n transactions from their buffer in each epoch. Assuming parties cannot receive an infinite amount of transactions in a finite number of epochs, there will be a finite number of transactions in P's buffer alongside tx. By the lower bound in Lemma 11 (Appendix B), honest parties continue to clear transactions from their buffers so that eventually tx appears among the first L transactions of their buffers. Once this has occurred, the probability that tx fails to appear in the output block at the e'th epoch if at least one of the honest parties that contributes its input to the block has tx among the first L transactions of its buffer is at most $1-1/n$. Thus, a transaction tx is included in $\mathsf{blocks}[e : e+r]$ with probability at least $1-(1-1/n)^{r+1}$, which approaches 1 as r goes to infinity. [2] □

Lemma 14. *Under condition $(*)$, at least $n - t_s - t_a$ honest parties have contributed transactions in any block output by a honest party in Π_{SMR}.*

Proof. All honest parties input valid pre-blocks in Π_{BLA} and Π_{ACS}, meaning that they wait to receive at least $n - t_s$ validly signed encrypted entries. By the t_s-security of Π_{BLA}, if Π_{BLA} outputs, it outputs a $(n - t_s)$-quality pre-block; even if the network is asynchronous, an honest party would not output an invalid pre-block. Therefore, honest parties' inputs to Π_{ACS} are also $(n - t_s)$-quality.

In case the network is asynchronous and there are at most t_a corrupted parties, $n - t_s - t_a$ entries in the pre-block originate from honest parties. By t_a-set quality of Π_{ACS} (Lemma 4), the output of Π_{ACS} contains at least a pre-block of $(n - t_s)$-quality, therefore with $(n - t_s - t_a)$ honest entries.

In case the network is synchronous, each honest party has received all messages from all other honest parties upon reaching Step 2 of Π_{SMR}, so the number of honest entries in their pre-blocks is at least $(n - t_s)$. Moreover, all honest parties complete Π_{BLA} with the same output pre-block B containing $(n - t_s)$ honest entries. By the t_s-validity with termination of Π_{ACS} (Lemma 3), the output pre-block of Π_{ACS} is also B. □

D.2 Committee-based ACS

Throughout this section, we consider a network that is asynchronous with up to $\hat{t}_a = (1 - \epsilon)t_a$ corruptions.

In $\Pi^{\kappa}_{\mathsf{ACS}}$, the termination conditions OC 1 and OC 2 make calls to $\Pi^{\kappa,e}_{\mathsf{Term}}$.

[2] The notion of eventual liveness considered here is standard under asynchrony. In the synchronous case, a more detailed analysis of liveness can be made, e.g. following the approach used in [7].

$\Pi^{\kappa,e}_{\mathsf{Term}}(x, h, \mathsf{coin})$

1. Self-elect based on the current signature key: if $\mathsf{VRF}_{\mathsf{sk}_i}(i, e, \mathsf{coin}) < b$, then compute proof ξ_i for $P_i \in \tilde{\mathcal{C}}$.
2. If $P_i \in \tilde{\mathcal{C}}$, multicast $\langle x, h, \xi_i \rangle_i$.
3. Upon receiving at least $t_s\kappa/n + 1$ valid signature shares and proofs $\langle x, H(x), \xi_j \rangle_i$ from distinct $P_j \in \tilde{\mathcal{C}}$, aggregate an output certificate $\tilde{c}$ for x and multicast $(\mathsf{output}, \tilde{c}, H(x))$. Output x and terminate.
4. Upon receiving a valid output certificate $\tilde{c}$ for x, multicast $(\mathsf{output}, \tilde{c}, h)$. Wait until reconstructed or received x, output x and terminate.

Fig. 11. Termination helper protocol from the perspective of party $P_{i\in\{1,\ldots,n\}}$.

Sketch of proof. (Lemma 7) We discuss the changes arising from the use of committees in the proofs for the properties of the $\Pi^{\kappa}_{\mathsf{ACS}}$ protocol. The proof will then follow from the proofs of Lemmata 1– 6.

The static adversary cannot tamper with the election of the primary committee because it can corrupt only up to $\hat{t}_s$ parties, while the signature aggregation requires $\hat{t}_s + 1$ signatures. The election of the secondary committees is done independently and in parallel, based on the coin computed this epoch. An adversary cannot tamper with these elections because of the unforgeability of the signature scheme and cannot predict the membership from previous epochs.

A committee election is unpredictable and modeled as a uniformly random sampling of κ parties (in the primary committee) or $O(\kappa)$ parties (in the other committees) from the pool of n parties. In expectation, the fraction of corrupted parties over all parties will be reflected in the committee. We select parameters κ and ϵ such that with high probability, there are at most $t_x\kappa/n$ corrupted parties in the committees and at most $t_x\kappa/n$ secondary committees contain more than $t_x\kappa/n$ corrupted members, where $t_x = t_a$ in the asynchronous case and $t_x = t_s$ in the synchronous case. The failure probabilities are given in Appendix C, using standard arguments based on the Chernoff and Hoeffding bounds. □

D.3 Committee-based BLA

Throughout this section, we consider a network that is synchronous with up to $\hat{t}_s = (1-\epsilon)t_s$ corruptions. We use the following validity definition that depends on the committee size:

- *(t-Validity) If every honest party has input an $(1 - t/n)\kappa$-quality pre-block, then every honest party outputs an $(1 - t/n)\kappa$-quality pre-block.*

Lemma 15. *$\Pi^{\kappa}_{\mathsf{BLA}}$ achieves $\hat{t}_s$-termination, $\hat{t}_s$-validity and $\hat{t}_s$-consistency except with negligible probability.*

In each call of $\Pi^{\kappa}_{\mathsf{Propose}}$, parties encode their inputs and send the codewords and hashes to the other parties, such that honest parties are able to reconstruct

```
$\mathit{\Pi}^{\kappa,e}_{\mathsf{BLA}}(B_i)$

1. At time 0: Set $B'_i = B_i$, $r = 0$, $C' = \emptyset$.
2. While $r < R$ do:
 - At time $7 \cdot r \cdot \mathit{\Delta}$: Run $\mathit{\Pi}^{\kappa,e}_{\mathsf{GC}}(r, B'_i, C')$.
 - At time $7 \cdot (r+1) \cdot \mathit{\Delta}$: Update $(B^*_i, C^*, g)$ as the output of $\mathit{\Pi}^{\kappa,e}_{\mathsf{GC}}$.
 - If $g > 0$ set $B'_i = B^*_i$, $C' = C^*$. Additionally, if $g = 2$, output $B^*_i$.
 - Set $r = r + 1$.
```

Fig. 12. BLA protocol from the perspective of party $P_{i \in \{1,\dots,n\}}$ in epoch e.

the proposed pre-block of the leader. Since parties might output a different pre-block than the one they started with in the current round, they have to send the hash and the codewords of their new input during the next round.

```
$\mathit{\Pi}^{\kappa,e}_{\mathsf{GC}}(r, B, C)$

1. At time 0: Set $C' = \emptyset$. Run $\mathit{\Pi}^{\kappa,e}_{\mathsf{Propose}}$ using $(r, B, C)$.
2. At time $5\mathit{\Delta}$: Let $B^*$ denote the output of $\mathit{\Pi}^{\kappa,e}_{\mathsf{Propose}}$. If $B^* \neq \perp$, try to
   self-elect in committee $\mathcal{C}_r$ and compute proof $\xi_i$ if $\mathsf{VRF}_{\mathsf{sk}_i}(i, r, e) < \mathsf{b}$. If
   $P_i \in \mathcal{C}_r$, multicast $\langle \mathsf{commit}, r, B^*, \xi_i \rangle$.
3. Until time $6\mathit{\Delta}$: Upon receiving a valid commit message, $c_j =$
   $\langle \mathsf{commit}, r_j, B_j, \xi_j \rangle_j$ from $P_j \in \mathcal{C}_r$, if this is the first such message re-
   ceived from $P_j$, add $c_j$ to $C'$.
4. At time $6\mathit{\Delta}$: If there is a subset $C'' \subseteq C'$ of commit messages on the
   same hash value $h'$ of a block $B'$ that $P_i$ can reconstruct, such that (a)
   $|C''| \geq \kappa t_s/n + 1$ and (b) for each $c_j = \langle \mathsf{commit}, r_j, B', \xi_j \rangle_j$ it holds
   that $r_j \geq r$, then multicast $\langle \mathsf{notify}, r, H(B'), C'' \rangle$, output $(B^*, C^*, 2)$
   and terminate.
5. At time $7\mathit{\Delta}$: If a valid notify message $\langle \mathsf{notify}, r, h^*, C^* \rangle$ and $P_i$ can re-
   construct $B^*$ such that $h^* = H(B^*)$, output $(B^*, C^*, 1)$ and terminate.
   (Choose arbitrarily is here are more than one such notify messages.)
   Otherwise, output $(\perp, \perp, 0)$ and terminate.
```

Fig. 13. GC protocol from the perspective of party $P_{i \in \{1,\dots,n\}}$ in round r.

We sketch the main ideas of the proof below. The full proofs can be obtained by expanding the BLA proofs in [7].

Sketch of proof. (Lemma 15) Parties terminate after participating in all R rounds of the protocol (even if they generated output earlier), so $\hat{t}_s$-termination is ensured by design.

We now argue $\hat{t}_s$-validity. Suppose all honest parties start $\mathit{\Pi}^{\kappa}_{\mathsf{BLA}}$ with input B of quality $t_s\kappa/n$. If the leader is honest, all honest parties terminate $\mathit{\Pi}^{\kappa}_{\mathsf{Propose}}$ with a pre-block B^* of quality $t_s\kappa/n$. This is because honest parties agree on the committee and can reconstruct their pre-blocks after INSE^{κ} since the code tolerates t_s erasures (honest parties distinguish invalid signatures and ignore

$\Pi^{\kappa,e}_{\mathsf{Propose}}(r, B, C)$

1. At time 0: Set $C' = \emptyset, V = \emptyset$. Input B to INSE^{κ}. Additionally, multicast $v_i = \langle \mathsf{vote}, r, H(B), C\rangle_i$.
2. Until time 2Δ: Upon receiving a vote $v_j = \langle \mathsf{vote}, r, h_j, C\rangle$ from a party $P_j \in \mathcal{C}$, add v_j to V. Store the committee $\mathcal{C}$ and the codewords and signatures $\{s_{j,k}, \varphi_{j,k}\}$. Denote by $V_{\mathcal{C}}$ the set of votes in V originating from members of $\mathcal{C}$.
3. At time 2Δ: Each party computes $z_i = \mathsf{VRF}_{\mathsf{sk}_i}(i, e)$ and a proof ξ_i and multicasts them.
4. At time 3Δ: Parties elect the leader P^* as the party having the smallest valid value z_i. If $P_i = P^*$ and $V_{\mathcal{C}} \geq t_s\kappa/n$, find the vote $v = \langle \mathsf{vote}, r^*, h^*, C^*\rangle$ in $V_{\mathcal{C}}$ such that (a) P^* knows B^* such that $h = H(B^*)$ and (b) r^* is greater than all other round numbers in $V_{\mathcal{C}}$, breaking the ties by lowest party index, and multicast $p^* = \langle \mathsf{propose}, r^*, h^*, C^*, V_{\mathcal{C}}\rangle_i$.
5. At time 4Δ: If a valid proposal p has been received from P^* (i.e., all signatures verify, the votes come from members of $\mathcal{C}$, the reconstructed B^* is such that $h^* = H(B^*)$), multicast p. Otherwise, output $\perp$.
6. At time 5Δ: Let p_j denote a valid proposal forwarded by P_j, if any. If there exists p_j such that $p_j \neq p^*$, output $\perp$. Otherwise, output B^*.

Fig. 14. $\mathsf{Propose}$ protocol from the perspective of party $P_{i\in\{1,\dots,n\}}$ in round r.

codewords with invalid signatures). Except with negligible probability, there will be at most $t_s\kappa/n$ corrupted parties in the primary committee, so $V_{\mathcal{C}}$ cannot contain $t_s\kappa/n + 1$ votes of corrupted parties. Thus, the proposal will be valid, ensuring all honest parties receive only valid proposals by time 5Δ and will output B^*. On the other hand, if the leader is dishonest, it can refuse to send a valid proposal by the required time. However, it cannot force honest parties to accept an invalid proposal since the adversary cannot tamper with the election of the primary committee (the network is synchronous and at most) and the corrupted leader cannot forge the signatures of the honest parties that are in the primary committee. In this case, the honest parties may terminate $\Pi^{\kappa}_{\mathsf{Propose}}$ with output $\perp$.

In $\Pi^{\kappa}_{\mathsf{GC}}$, with high probability, at least $(1 - t_s/n)\kappa$ members of $\mathcal{C}_r$ will be honest and send B^* in step 2 of $\Pi^{\kappa}_{\mathsf{GC}}$, which will reach all honest parties by the beginning of the next round. Therefore, in step 5 of $\Pi^{\kappa}_{\mathsf{GC}}$, a party receiving a valid notify message will be able to determine to what block B^* the hash h^* corresponds and to output B^*. Moreover, fewer than $t_s\kappa/n$ parties in $\mathcal{C}_r$ are corrupted, so there cannot be sufficient votes in C'' if no honest party participates. (Recall that honest parties do not output invalid B^* from $\Pi^{\kappa}_{\mathsf{Propose}}$.) Therefore, if the leader is honest in r^*, all honest parties output a valid B^* with grade 2 and thus output in $\Pi^{\kappa}_{\mathsf{BLA}}$. Since $\hat{t}_s/n < 1/2$, a dishonest leader is elected with probability smaller than 1/2, so the probability no honest leader is elected in R rounds is negligible. This proves $\hat{t}_s$-validity.

To prove $\hat{t}_s$-consistency, suppose r^* is the first round in which some honest party P_i has output a pre-block B. P_i must have generated a notify message and output with grade 2 in $\Pi^{\kappa}_{\mathsf{GC}}$. Then no honest party can output with grade 0, and all honest parties must have received that notify message in the same round. Therefore, all honest parties will use B as input in iterations greater than $r^* + 1$. Moreover, no honest party could have sent a commit message on a different pre-block $B' \neq B$ in the same execution of $\Pi^{\kappa}_{\mathsf{GC}}$, and so a corrupted leader cannot construct a valid vote on β' in a subsequent round number. Inductively, we can argue that honest parties will keep inputting B and not voting on other blocks in all subsequent rounds until R, so all honest parties will output B at the end of $\Pi^{\kappa}_{\mathsf{BLA}}$. □

If the network is asynchronous, note that we cannot guarantee any of the termination, consistency and validity properties. However, in the SMR protocol, the BLA step is followed by ACS, so that ACS compensates for BLA if the network is asynchronous (and likewise BLA compensates for ACS if the network is synchronous). This is the reason it suffices to show that BLA achieves termination, consistency and validity in a synchronous network.

D.4 Attack on network-agnostic committee-based SMR from an adaptive adversary

If the network is purely synchronous, Upstate SMR is adaptively-secure. The reason is that all parties start each part of the protocol at the same time and the individual subprotocols are designed such that an adaptive adversary cannot learn information about committee membership before the honest parties have taken their actions.

In a multi-epoch SMR protocol run in a asynchronous network, the adversary can delay a party from proceeding to epoch $e + 1$. Since $\hat{t}_a$ delayed parties are indistinguishable from $\hat{t}_a$ corrupted parties, the protocols should be designed to proceed and terminate while requiring at most $n - \hat{t}_a$ messages. Therefore, the adversary can delay up to $\hat{t}_a$ parties from starting, e.g., the INSE part in $\Pi^{\kappa}_{\mathsf{ACS}}$. The protocol proceeds and generates the coin, and the adversary can learn the primary committee membership before the delayed members sent their messages.

Say the adversary has corrupted $f < \hat{t}_a$ parties so far in this epoch, and has a budget of $\hat{t}_a - f$ remaining corruptions. In expectation, out of the corrupted f parties, $f\kappa/n$ of them will be members, and out of the delayed $\hat{t}_a$ parties, $t_a\kappa/n$ will be members. However, $\hat{t}_a - f \geq t_a\kappa/n$, for a variety of values of f, so the adversary can corrupt $(t_a + f)\kappa/n$ members of the committee. This is more than the $t_a\kappa/n$ allowed committee corruptions and breaks the $\hat{t}_a$-security of $\Pi^{\kappa}_{\mathsf{ACS}}$.

We note that a self-elected primary committee resolves the attack in the sense that the adversary is not able to corrupt and control the inputs of more than $t_a\kappa/n$ committee members. However, there will be no agreement between honest parties on the membership in the primary committee in the asynchronous case, so we cannot simultaneously achieve $\hat{t}_a$-consistency and $\hat{t}_s$-validity in the $\Pi^{\kappa}_{\mathsf{ACS}}$. Designing an adaptively secure SMR protocol in the network-agnostic case that achieves $O(n^2)$ communication per $O(n)$-block is a challenging open problem.

D.5 Communication complexity of the protocols

Let ℓ be the input length and let b be the number of blocks into which the input is divided, so that each codeword has size ℓ/b. Also note that we consider a regime where $\kappa < n$.

Protocol Π_{ACS} from Section 4 Step 2 of INDI involves n^2 messages of $O(\kappa + l/b)$ bits, so the total communication is $O(\kappa n^2 + n^2 l/b)$. Resending the received codewords and signatures corresponding in step 3 of INDI takes $O(n^3 l/b + \kappa n^3)$ total communication. Step 2 of Π_{ACS} where parties send their votes takes $O(\kappa n^3)$. A certificate contains a full signature of size κ, so forwarding the received certificates takes $O(\kappa n^3)$ communication in step 3. Using an instantiation of asynchronous binary BA with quadratic communication, the communication for all n BA instances takes $O(\kappa n^3)$ communication. Π_{Term} is run only on inputs of size ℓ, and requires $O(n^2 l + \kappa n^2)$ communication. We consider a regime where $\kappa < n$. Therefore, the asymptotic communication of Π_{ACS} is $O(n^2 l + n^3 l/b + \kappa n^3)$ per input of size ℓ.

The parameter b of the error correcting code is chosen to be $b = t_s = O(n)$, leading to a communication of $O(n^2 l + \kappa n^3)$. This yields an amortized communication of $O(n^2)$ per input of size $\ell = O(\kappa n)$.

Protocol $\Pi^{\kappa}_{\mathsf{ACS}}$ from Section 6 Step 3 of INSE^{κ} involves n^2 messages of $O(\kappa + l/b)$ bits, so the total communication is $O(\kappa n^2 + n^2 l/b)$. Resending the received codewords and signatures corresponding to the κ parties in the primary committee in step 4 of INSE^{κ} takes $O(\kappa n^2 l/b + \kappa^2 n^2)$ total communication. Step 2 of $\Pi^{\kappa}_{\mathsf{ACS}}$ where members of κ secondary committees send their signatures on hashes to all other parties takes $O(\kappa^3 n)$. A certificate contains $O(\kappa)$ signatures of size κ, so forwarding the received certificates corresponding to the proposals of the primary committee takes only $O(\kappa^3 n)$ communication in steps 3 and 4. Using an instantiation of asynchronous binary BA with $O(\kappa n^2)$ communication, the communication for all κ BA instances takes $O(\kappa^2 n^2)$ communication. Termination conditions OC 1 and OC 2 incur another $O(\kappa n l + \kappa^2 n^2)$ total communication. Overall, the total communication of $\Pi^{\kappa}_{\mathsf{ACS}}$ is $O(\kappa n^2 l/b + \kappa n l + \kappa^2 n^2)$ communication per block.

The parameter of the error correcting code $b = \hat{t}_s = O(n)$. To have total communication $O(\kappa^2 n^2)$ per block, we can set $\ell \leq O(\kappa n)$ so $\ell/b \leq O(\kappa)$. This yields an amortized communication of $O(\kappa n)$.

Protocol $\Pi^{\kappa}_{\mathsf{BLA}}$ from Section 6 The commit list can have $R\kappa^2$ size. $\Pi^{\kappa}_{\mathsf{Propose}}$ incurs $O(\kappa^2 n^2 + \kappa n^2 l/b)$ total communication from INSE^{κ}, $O(n^2 \kappa^2 R)$ from the vote and propose multicasts, and $n^2\kappa$ for the leader election, for a total of $O(\kappa^2 n^2 R + \kappa n^2 l/b)$. Additionally, $\Pi^{\kappa}_{\mathsf{GC}}$ contributes $O(\kappa n l + \kappa^2 n^2 R)$ to the communication cost. These steps are run for $R < \kappa$ rounds, so that in total $\Pi^{\kappa}_{\mathsf{BLA}}$ incurs $O(R\kappa^2 n^2 + \kappa n^2 l/b + \kappa n l)$ communication complexity.

To obtain $O(R\kappa^2n^2)$ communication complexity, we can set $\ell = O(\kappa n R)$ so that, yielding an amortized communication per block of $O(\kappa n)$.

$\mathsf{Upgrade}$ [7] achieves a total communication complexity of $O(\kappa^2n^3+\kappa nL|\mathsf{tx}|+\kappa^3n+L|\mathsf{tx}|\kappa^2)$ per block (the number of rounds R is absorbed), which enables a linear amortized cost for a block size of $L = O(\kappa n^2)$. Such a block size implies a communication complexity of $O(\kappa^2n^3)$ only to distribute them to a small committee of size κ, which already exceeds the quadratic communication limit we imposed for $\mathsf{Upstate}$. Nevertheless, both $\mathsf{Upgrade}$ and $\mathsf{Upstate}$ need to be run over hundreds of thousands of parties in order to achieve asymptotic communication improvements over $\mathsf{Tardigrade}$ and Update. In this large-scale context, assuming that parties have $O(n^2)$ transactions in their buffers at every epoch is quite restrictive. Therefore, we believe that providing a protocol with linear amortized communication per $O(n)$ transactions has more relevance in practice.

E Network changes

Proof. (Lemma 8) By t_a consistency and validity of BA, at least one honest party needs to input 1 in a BA instance in order for it to output 1. This means honest parties need to be able to construct at least $n - t_a$ certificates. Clearly, certificates corresponding to the $n - t_s$ honest and unexposed parties can be eventually reconstructed. Therefore, we focus on the case of building a certificate for an exposed party P.

When multicasting messages in step 3 of INDI, corrupted parties can send erroneous codewords on behalf of P. Therefore, in RECON, up to t_a of the at least $n-t_a$ codewords can have a valid signature but are erroneous (but need to have the same hash h in order to be taken into consideration). While the code cannot tolerate at the same time t_a errors and t_s erasures, with overwhelming probability, the value x output by DEC on erroneous codewords will not satisfy $h = H(x)$. Therefore, honest parties wait for more correct codewords, which are guaranteed to eventually arrive, since $n - t_a$ parties behave honestly, so honest parties can assemble certificates for exposed parties as well.

Proof. (Lemma 9) Since the adversary cannot act on behalf of the exposed parties directly, the arguments for when P is honest and unexposed, and for when P is dishonest are the same as the arguments in the proof of Lemma 1.

Assume P is exposed. The same argument as for a dishonest P that sends codewords in the first round of INDI applies for an exposed P. Assuming H is a collision-resistant hash function, there do not exist values $x \neq x'$ reconstructed by different sets of codewords such that $h = H(x) = H(x')$. Therefore, if after inputting $n - t_s$ codewords to RECON and not obtaining a valid output with respect to h, the honest parties wait until they receive $n - t_s + t_a$ codewords in order to be able to correctly reconstruct. □

Proof. (Lemma 10) Since the adversary cannot act directly on behalf of the exposed parties, the arguments for when P is honest and unexposed and when P is dishonest can be taken directly from the proof of Lemma 2.

Suppose P is exposed. The same argument as for an exposed P ensures that because $n > 2t_s + t_a$, there needs to be one honest party that would sign both certificates, implying $h = h'$. □

The adversary can include more malicious entries in the block by forging the signatures of t_s instead of t_a parties. However, asymptotically, the honestly generated throughput remains the same as in the network-agnostic case.

Upstate **SMR under network changes**

Sketch of proof. (Theorem 5, Upstate) Despite knowing $\hat{t}_s$ keys, a static adversary cannot actively corrupt more than $t_a\kappa/n$ parties in any of the committees with high probability (see Appendix C), because it selects the corrupted parties before the epoch starts and committee membership is unpredictable.

The arguments for t_a-security of Upstate under arbitrary network changes follow from the arguments in the proof of Theorem 5. □

As a side note, an adaptive adversary knowing $\hat{t}_s$ keys would be able to corrupt up to $t_s\kappa/n$ parties in the secondary committees. Nevertheless, since the secondary committees are only used to construct certificates of $\hat{t}_s + 1$ keys, this would not be a problem.

Tardigrade **SMR under network changes**

Sketch of proof. (Theorem 5, Tardigrade) Let $\overline{\Pi_{\mathsf{ACS}}}$ denote the asynchronous common subset from Tardigrade.

ACS. The t_s-valid and t_a-consistent reliable broadcast protocol used in $\overline{\Pi_{\mathsf{ACS}}}$ is signature-free, and furthermore, the non-terminating ACS protocol is signature-free (apart from the BA components). Thus, the proofs for t_a-consistency, t_s-validity, t_a-liveness, and t_a-set quality of the ACS protocol from [7, Sec. 4] hold.

The terminating wrapper of the ACS protocol, $\overline{\Pi_{\mathsf{ACS}}}$, uses threshold digital signatures. Thus, the adversary can forge the threshold signatures of up to t_s parties. However, it cannot create acceptable certificates on its own. An honest party will sign an output if and only if it has already terminated ACS with that output. Thus, there can never be a valid certificate for an invalid output. Therefore, $\overline{\Pi_{\mathsf{ACS}}}$ is t_a-consistent, t_a-live, t_a-terminating, t_s-valid with termination and has t_a-set quality [7, Sec. 4].

The arguments for Π_{BLA} and Π_{SMR} are similar to the ones in the proof of Theorem 5. □

We note that we recover a similar result as [19] concerning synchronous authenticated broadcast for $n > 2t_s + \min(t_s, t_e - t_s)$, where t_s is the number of corrupted parties and t_e is the number of exposed parties.

Discussion on the model

Mobile corruptions. Without even mild synchrony assumptions, such as a time signal from an external clock, a mobile adaptive adversary that can corrupt any of the n parties can break either privacy or liveness of a proactive threshold protocol, as we describe in Section 8. This motivates a limiting assumption over the adversary's powers: the overall number of parties that can be corrupted over the duration of the protocol is below the total number of parties, and the adversary can only be mobile in between that smaller set of parties. In order to further characterize the adversarial mobility, we need to granulate the time. We do so by defining local epochs, characterized by the parties' actions, not by a clock. We assume that the adversary can only corrupt a limited number of parties at any given moment in time, and in any given epoch. Without this assumption, an adversary could corrupt the maximum number of parties at an onset of the epoch, take some actions and deliver only those associated messages, then immediately move into a different maximal set of parties, which in the asynchronous case would be equivalent to corrupting twice the allowed threshold.

Channels. An initially agreed upon public key infrastructure (initially) ensures agreement and confidentiality. However, as the adversary corrupts the parties and learns their secret keys, authentication and confidentiality are not ensured anymore, even if the adversary is flushed out from the party, since it can use the learned keys to decrypt communication, forge signatures, and create messages that appear to have been created by the corresponding party. Without an assumption of authenticated channels for all the duration of the protocol, the adversary can spawn other parties in the network to impersonate the uncorrupted parties. This assumption is standard in other proactive protocols [9,32].

F Privacy game for asynchronous proactive secret sharing

The game proceeds as follows between a t_a-limited adversary and a challenger.

1. The adversary chooses two secrets $s_0, s_1 \in \mathbb{F}$ and gives them to the challenger.
2. The challenger chooses $b \leftarrow \{0,1\}$ and runs $(s_1^{(0)}, \dots, s_n^{(0)}) \leftarrow \mathsf{Share}(s_b)$.
3. For an epoch number $\tau \geq 0$ the adversary specifies a set of parties $\mathcal{C}^\tau$, where $|\mathcal{C}^\tau| \leq t_a$ and sends it to the challenger. Denote by $\mathcal{H}^\tau$ the set of honest parties in epoch τ. The challenger sends to the adversary the corresponding state of the parties, which includes the shares $\{s_i^{(\tau)}\}_{i \in \mathcal{C}^\tau}$.
4. The adversary sends replacing shares $\{s_i^{(\tau)'}\}_{i \in \mathcal{C}^\tau}$ to the challenger, which sets $s_i^{(\tau)} = s_i^{(\tau)'}$ for $i \in \mathcal{C}^\tau$ and specifies the actions of the parties in $\mathcal{C}^\tau$. The adversary specifies the set of messages $\mathcal{M}$ (including other messages than the ones associated to the proactive VSS protocol) to be delivered to the honest parties from the set C^τ. It also specifies to the challenger which messages between the remaining honest parties are to be delivered over the network; note that this means that the adversary can refuse to deliver a message originated in epoch $\tau' < \tau$ from a party in $\mathcal{C}^\tau$ to a party in $\mathcal{H}^\tau$.

5. For a epoch τ for which the challenger has a set $\mathcal{S}$ of at least $n - t_a$ associated shares, the challenger runs $(s_1^{(\tau+1)}, \ldots, s_n^{(\tau+1)}) \leftarrow \mathsf{Reshare}(\tau, \{s_i^{(\tau)}\}_{i \in \mathcal{S}})$.
6. If τ is as large as desired by the adversary, go to step 7. Otherwise, increment the epoch number by 1 and go to step 3.
7. Eventually, the adversary outputs a guess $b' \in \{0, 1\}$ of b.

Denote the adversary's advantage by $\mathsf{Adv}(\mathcal{A}) = \left|\Pr[b' = b] - \frac{1}{2}\right|$.