State Machine Replication under Changing Network Conditions

Andreea B. Alexandru[1⋆][0000−0001−5396−1241], Erica Blum[1⋆][0000−0001−7497−7592], Jonathan Katz[1⋆][0000−0001−6084−9303], and Julian Loss[2⋆⋆][0000−0002−7979−3810]

[1] University of Maryland, College Park
{aandreea,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 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 the 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 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 maintained. We show that purely asynchronous proactive secret sharing is impossible without some form of synchronization between the parties, ruling out a natural approach to proactively secure network-agnostic SMR protocols. 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 even under arbitrarily changing network conditions.

Keywords: State Machine Replication, Consensus, Proactive Security

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

⋆ Work supported in part by NSF award #1837517.

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

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 considerable 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. SMR protocols in the synchronous model can tolerate $t < n/2$ corrupted parties (or $t < n$ corrupted parties if external validity is not required [34]), 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 [7] introduced the *network-agnostic* model in which a single protocol is required to be secure regardless of whether it is run in a synchronous or an asynchronous network, for different corruption thresholds. In subsequent work [8], 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 [7,9,8,29,5,17], 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 on only 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. [8] 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 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 all use the same input B to ACS which outputs this block even if there are t_s corrupted parties. On the other hand, if the network is asynchronous, t_a-security of ACS ensures that all parties can agree on B without relying on the synchronous protocol.

Problems with existing solutions. Blum et al. [8] present two SMR protocols, Tardigrade and Upgrade. Tardigrade is secure against an adaptive adversary and requires $O(n^4)$ bits of communication for n transactions. Upgrade gives a more efficient alternative against a static adversary that requires only $O(n^3)$ bits of communication for n^2 transactions. However, Upgrade relies on random subcommittees to execute the most expensive steps of the protocol. Such protocols are not adaptively secure and require very large committees 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 work only considers non-switching networks, i.e., the network is either synchronous or asynchronous for the entire duration of the protocol. Thus, if at any point in the lifetime of the protocol the adversary surpasses t_a corrupted parties, their protocols might be insecure if the network is ever asynchronous. We are interested in a more flexible model that tolerates repeated transitions of the network between synchronous and asynchronous behavior, and even in the presence of an adaptive, mobile adversary.

1.2 Our Contributions

We study protocols in a more realistic model where network conditions can arbitrarily change over time, and parties can also recover from corruptions. Such recovery is necessary if we want to allow more than t_a corruptions when the network is synchronous, but then restrict the adversary to fewer than t_a corruptions when the network becomes asynchronous.

Modeling recovery from key exposure. Modeling parties that are temporarily corrupted (sometimes referred to in the literature as *transient faults*) is non-trivial when parties have long-term keys. 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 the previously corrupted parties (including their long-term secret keys) remains known to the adversary. Proactive secret

sharing is the main technique to refresh parties' keys for threshold signatures and related primitives commonly used in communication-efficient randomized SMR protocols. We prove that without further restrictions, secure proactive secret sharing protocols in the pure asynchronous and network-agnostic setting are impossible. While this may seem to be a folklore result, modeling and proving such a result is non-trivial. One of our contributions is to formalize this result and provide a rigorous proof.

To address the above impossibility in the context of SMR protocols, we consider a model in which the attacker is limited to corrupting a set S of at most t_s parties *for the lifetime of the protocol*. (It may corrupt this entire set of parties when the network is synchronous, and must uncorrupt at least $t_s - t_a$ of them when the network becomes asynchronous.) 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.

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

Update is adaptively secure for optimal corruption thresholds and has $O(n^3)$ communication complexity for committing a block of $O(n)$. This is an $O(n)$ improvement over Tardigrade [8], which requires $O(n^4)$ communication to commit blocks of $O(n)$ transactions. We obtain the improvement by carefully applying error-correcting codes in a new ACS protocol.

Upstate is statically secure for near-optimal corruption thresholds and has $O(n^2)$ communication complexity to commit blocks of $O(n)$ transactions. Upstate achieves its improved communication complexity by using committees. Upstate compares favorably to Upgrade [8]: while Upgrade requires $O(n^3)$ communication to commit blocks of $O(n^2)$ transactions, Upstate commits blocks of $O(n)$ transactions and requires $O(n^2)$ communication.

SMR tolerating key exposure. We show that our protocols are also secure when the network can transition between synchronous and asynchronous behavior and the adversary can be mobile across epochs, but is limited to corrupting at most t_s unique parties. Adding reboots at the beginning of each protocol epoch to flush the adversary out helps Update and Upstate to withstand the 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. We leave open the question of designing an adaptively secure SMR protocol in our setting with quadratic communication complexity per committed block. We also leave open to explore communication-efficient proactive network-agnostic SMR protocols that bypass the impossibility result of network-agnostic proactive secret sharing. We remark that although our protocols use

threshold cryptosystems to boost efficiency and censorship resilience, these may not be necessary. Thus, it is plausible that a solution for key refresh could be achieved without limiting the adversary to corrupting a set of t_s parties. One could then hope to use a network-agnostic ACS protocol to agree on a new list of valid public keys obtained from distributed key generation.

1.3 Related work

Network-agnostic protocols were introduced by Blum et al. in the context of Byzantine agreement [7], and was later extended to multi-party computation [9] and SMR [8]. The latter presents two network-agnostic SMR protocols. Tardigrade achieves total communication $O(n^4+n^3\ell)$ against adaptive adversaries, for n the number of parties and ℓ the block size. Upgrade uses committees to achieve total communication $O(n^3 + n\ell)$ against static adversaries (but tolerates fewer corruptions). Appan et al. [5] proposed a protocol for network-agnostic perfectly secure multi-party computation; their protocol uses a novel network-agnostic perfectly secure verifiable secret sharing protocol.

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 [24,11,28,15,21] 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 the network-agnostic setting.

A final group of related works concerns secret sharing and distributed key generation (DKG) where parties may crash and then recover or where the set of participants may change. In the *proactive model* [31], the adversary can be mobile across the corrupted parties over time. *Proactive secret sharing* (PSS) was introduced by Herzberg et al. [22]. Canetti et al. [12] and Frankel et al. [16] gave solutions for synchronous DKG against adaptive proactive adversaries using verifiable secret sharing schemes. Benhamouda et al. [6] introduced a secret-sharing protocol for passing secrets from one anonymous committee to another, while Groth [20] proposed a DKG scheme based on publicly verifiable secret sharing that allows refreshing key shares to a new committee. In the asynchronous case, Cachin et al. [10] presented a proactive refresh protocol assuming clock ticks that define epochs, based on [13] which recovers state in an SMR protocol. Schulze et al. [33] proposed a mobile PSS protocol in a partially synchronous network. Recently, several works [27,35,32] have proposed more efficient dynamic/mobile PSS protocols assuming eventual synchrony, short periods of synchrony at the end of an epoch, or synchronized epochs. Subsequent to our work, Yurek et al. [36] constructed an asynchronous dynamic PSS protocol (circumventing our impossibility result) but with respect to different definitions than ours.

A related notion of security in the presence of exposed parties was considered in [19], which studied synchronous authenticated broadcast with both corrupted parties and parties who are honest but whose keys have been exposed.

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. 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 $P_1, \ldots, P_n$ 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, and the local clocks of the parties are synchronized. When the network is *asynchronous*, messages between parties are eventually delivered to their intended recipient, but may be adversarially delayed or reordered. 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 synchronous and asynchronous behaviors 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*. 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 $2t_s + t_a < n$ [8]). 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 times during the protocols, not necessarily simultaneously. 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. 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 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. For a parameter κ, a signature share and a full signature have length $O(\kappa)$. We implicitly assume that parties use domain separation when constructing signatures to ensure only local context validity. An encryption of a message m of length $|m|$ has length $|m| + O(\kappa)$, and a decryption share has length $O(\kappa)$; these criteria can be met using standard KEM/DEM mechanisms.

3 Preliminaries

State machine replication protocols enable a set of parties to emulate a single server by agreeing on an ever-growing, ordered log of transactions.[3] 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 $P_1, \ldots, P_n$. Let pp be some public parameters (e.g., PKI). Parties receive transactions as input, locally maintain arrays blocks, and output blocks and a publicly verifiable proof $\pi_i[e]$ for each $\mathsf{block}_i[e]$ in blocks. Π is a secure SMR 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 $\mathsf{Verify}(\mathsf{pp}, B, \pi) = 1$.*

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 BA protocol tolerating t corruptions if the following properties hold:*

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

- *(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 ACS 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 [8]) is a 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 with a valid signature attached. 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 BLA 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.

Threshold signature schemes. A (t, n)-threshold signature scheme is a signature scheme allowing $t+1$ parties out of n to compute a signature on a message, with up to $t < n$ corruptions. It is *non-interactive* if parties can non-interactively compute signature shares that can be combined in the signature on a message, using protocols TS.Setup, TS.KeyGen, TS.Sign, TS.ShVer, TS.Verify for setup, key generation, partial signing, share verification and signature verification. The desired properties are correctness, security (unforgeability under chosen-message attack) and robustness (any number $\geq t+1$ of signature shares 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.

Given inputs $m_1, \ldots, m_b$, the encoding function ENC computes codewords $s_1, \ldots, s_n$. Knowledge of any b elements of the codeword uniquely determines the input message and the remaining of the codeword.

The decoding 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 to order 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.

Throughout the paper, we deal with several security parameters. The signature size and the hash output size depend on a parameter that ensures computational security. The committee sizes depend on a parameter that ensures a negligible failure probability. To streamline notation, we denote all these by κ.

4 Asynchronous Common Subset

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 a linear error correcting code. 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. INDI is performed before the agreement on whose messages to output, and ensures that all parties are eventually able to reconstruct the selected inputs despite an adaptive adversary.

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. Upon receiving $t_s + 1$ votes for P_j, parties assemble a certificate of validity for the reconstructed value of 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 recently, Das et al. [14] proposed an asynchronous reliable broadcast protocol using error correcting codes (but without digital signatures) that is related to this step. Finally, upon receiving a unique commit message for party P_j, parties input 1 to the corresponding BA_j instance. We implicitly assume that if honest parties receive conflicting commit messages, they do not input 1 to the respective BA.

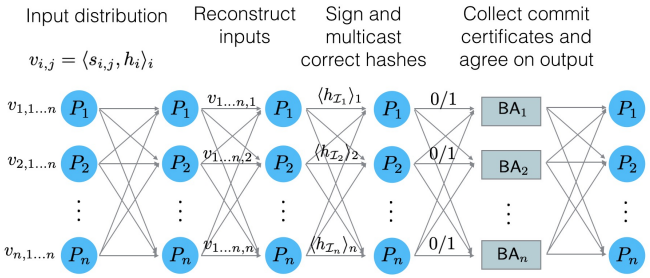

Fig. 1. Diagram of the steps in the Π_{ACS} protocol. BA stands for Byzantine Agreement. $\mathcal{I}_i$ is the set of indices j for which party P_i reconstructed the initial message of party P_j.

$\mathsf{INDI}(x)$

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.

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

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 $P_{i \in \{1,\ldots,n\}}$.

Protocol Π_{Term} (Figure 4) assembles an output certificate that allows parties 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 TS.KeyGen and sk_i the secret key associated to P_i. 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 over h_i, sent to party P_j. In this section (and all sections but Section 7), 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 $O(n^2)$.

Encoding and reconstruction. ENC and DEC are associated to a (n, b)-RS code (Section 3). 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. If an honest party has not managed to

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

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 a commit certificate c_j for the input of a party P_j, forward it to all parties.
5. Upon receiving a commit 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 certificate $(\mathsf{output}, \tilde{c}, x, h)$, multicast $(\mathsf{output}, \tilde{c}, x, h)$. Output x and terminate.

OC 1. Else if P_i (i) has obtained $n - t_s$ certificates $(\mathsf{commit}, \langle h_j\rangle)$ and (ii) 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 (i) $|\mathcal{S}| \geq n - t_a$, (ii) all n BA instances have terminated, (iii) P_i has obtained certificate $(\mathsf{commit}, \langle h_j\rangle)$ for $j \in \mathcal{S}$, (iv) 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 (i) $|\mathcal{S}| \geq n - t_a$, (ii) all n BA instances have terminated, (iii) P_i has obtained certificates $(\mathsf{commit}, \langle h_j\rangle)$ and (iv) 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 $P_{i\in\{1,\ldots,n\}}$.

reconstruct an input yet, it waits for more messages, then calls RECON again. Thus, each party feeds at least $n - t_s$ valid codewords in DEC. The (n, b)-RS code 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 be equal to 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 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.

If we let b 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 $(\mathsf{commit}, \langle h \rangle)$ for a party P, 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. To obtain a valid commit 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 stated above, 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 commit 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 this contradicts our assumption that $n > 2t_s + t_a$. □

Note that if the network is synchronous and $t_s = \lfloor n/2 \rfloor, t_a = 0$, different honest parties could receive commit certificates on different hashes of the same malicious party (honest parties always 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 equivocation is not necessarily 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 commit 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 despite t_s corruptions that could break security of the t_a-secure BA, it saw x' reconstructed in a strict majority of valid values associated with $n - t_a$ BA terminated instances. Any set of BA 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 $n \geq n - t_s + \lfloor \frac{n-t_a}{2} \rfloor + 1$, 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 P_i has terminated, all honest parties eventually terminate. If P_i output due to OC 0 (implying it received a valid output certificate from OC 1 or OC 2), then eventually all honest parties receive the certificate multicast by P_i and terminate (if they have not already).

If P_i output due to condition OC 3, then it must have terminated all BA instances, obtained commit 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 parts (i) and (ii) of OC 3 to be true. Furthermore, each honest party eventually reconstructs each $\{x_j\}_{j \in \mathcal{S}}$ and receives the certificates needed to 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 an output 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 $\lfloor \frac{n-t_a}{2} \rfloor + 1$ 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.

Third, 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. 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 agree on the set of BA instances $\mathcal{S}$ that output 1. By Lemma 1, we have $S' = S$. □

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

5 The Update SMR Protocol

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 ACS protocol described in Section 4, we also use a block agreement protocol (BLA), whose role is to make parties agree on the input to ACS if the network is synchronous. Honest parties input $(n - t_s)$-quality pre-blocks of length L to the BLA and ignore any pre-blocks with quality less than $n - t_s$.

We use the adaptively secure BLA protocol from [8], which we call 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 epoch 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 to improve throughput, i.e., not have too much overlap or separation between epochs. If the network is synchronous, then epochs start at the same time for all parties. 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 e.

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 the full version [4].

Condition $(*)$**.** Assume $t_a \leq t_s$, $2t_s + t_a < n$, 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.*

Communication 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

Π_{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 Π_{BLA}^e. If Π_{BLA}^e terminates, let B^* be the output.

2.2 At time $T_e + (5R+1)\Delta$: Terminate Π_{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 Π_{ACS}^e.

2.4 Receive $S = \{B_j^*\}_{j \in \mathcal{S}}$, where $\mathcal{S} \subset \{1, \ldots, n\}$ from Π_{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 block by sorting $\bigcup_{j \in \mathcal{S}} x_j$ in canonical order.
- Hash and sign 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 π as $\langle h \rangle$ and proof of correct decryption of S.
- Remove the transactions in block from the buffer and output (block, π).

3.3 Update $e \leftarrow e + 1$.

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

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 The Upstate SMR Protocol

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.

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

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

1. Encode x_i using 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 $\hat{t}_s + 1$ signatures σ_j and compute coin from them.
5. For each $j \in \{1, \ldots, n\}$, compute $\bar{h}_j := H(\mathsf{coin}, j)$ and select the first κ 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.

(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.

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, 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 of 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 the full version of the paper [4]. After running BLA and ACS, the parties construct the final block by jointly decrypting the output value of $\Pi^{\kappa}_{\mathsf{ACS}}$.

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

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

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 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 $B_i^e[j'] = \perp$, set $B_i^e[j'] := x_j$, 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_i^e, 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_i^e 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_i^e 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 Update Π_{SMR}.

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

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

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. For simplicity, 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 $\hat{t}_s$-validity with termination and $\hat{t}_a$-set quality except with negligible probability.*

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 construct a protocol BLA^{κ}, based on the BA protocol from [2,1] and the block agreement protocol from [8], with several changes to achieve security against adaptive adversaries at a quadratic communication per pre-block. The high-level idea is to elect a leader who proposes an input among the ones sent

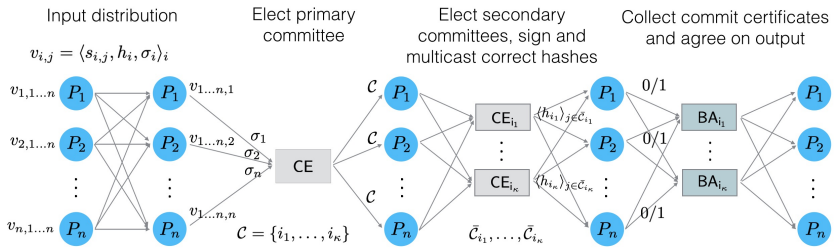

Fig. 8. Diagram of the steps in the $\Pi^{\kappa}_{\mathsf{ACS}}$ protocol. CE stands for committee election and BA for Byzantine Agreement.

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

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 a commit certificate c_j for the input of party P_j, forward it to all parties.
5. Upon receiving a commit 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.

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. The construction is given in the full version [4].

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 a given round , only votes from the current committee 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 $\Pi^{\kappa}_{\mathsf{Propose}}$.

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 $\Pi^{\kappa}_{\mathsf{SMR}}$, $\Pi^{\kappa}_{\mathsf{BLA}}$ and $\Pi^{\kappa}_{\mathsf{ACS}}$ 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. $\Pi^{\kappa}_{\mathsf{SMR}}$ incurs $O(n^2 L/(\kappa b) + n^2\kappa)$ 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, $\Pi^{\kappa}_{\mathsf{BLA}}$ incurs $O(R\kappa^2 n^2 + \kappa n^2 L/b + \kappa nL)$ total communication and $\Pi^{\kappa}_{\mathsf{ACS}}$ incurs $O(\kappa n^2 L/b + \kappa nL + \kappa^2 n^2)$ total communication.

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

7 SMR under Arbitrary Network Changes

We now consider a network that can arbitrarily transition between synchronous and asynchronous behaviors and 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 to flush the adversary out. Importantly, the state of the parties is not removed from the adversary's view after uncorruptions.

Adding a reboot step at the beginning of each epoch to the network-agnostic protocols discussed so far, Update and Upstate, as well as Tardigrade, results in protocols that are secure under arbitrary network changes, as long as rebooting ensures that $n > 2t_s + t_a$, $t_a \leq t_s$, 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, Upstate, *and* Tardigrade *[8] with reboots are secure under arbitrary network changes against a constrained epoch-mobile adaptive adversary, where* $n > 2t_s + t_a$, $t_a \leq t_s$.

We prove the first part of Theorem 5 below, after some technical observations. Proofs of the rest of Theorem 5 and of the Lemmata are given the full version [4].

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 in this section, we use the binary BA protocol from [7], 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 a Π_{ACS} execution, 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. 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, 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.

In the following, we first examine the security of the building blocks and then the security of the overall protocol.

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} [8], 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 that 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. 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 [4]). However, this still happens and ensures that tx is eventually output, but the probability increases 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

We first consider an asynchronous network in the presence of a mobile adaptive adversary. At the end, we extend the analysis to 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 any assumption on epoch length (as in [10] where epochs are defined to take place between clock ticks) but with epochs determined by a successful reshare of the secret (as in [33] but where the network is partially synchronous). While Cachin et al. [10] 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* Reshare, 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.*
- 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* Reshare *with associated epoch* τ *sets its epoch number to* $\tau + 1$*.*
- 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* Share, Reshare, *or* Reconstruct *in epoch* τ when they generate the corresponding output from the algorithm in epoch τ.

We give a standard privacy game between a challenger and an adversary $\mathcal{A}$ where the goal of the adversary is to learn the secret in the full version of the paper [4]. 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* 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* 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, Share, Reshare and Reconstruct need to implicitly have validation procedures of the inputs. We asked for at least $n - t_a$ instead of $t_a + 1$ parties to participate in Reconstruct to guarantee success against t_a malicious parties who could submit t_a invalid shares. Nevertheless, $t_a + 1$ valid shares are sufficient to reconstruct the secret.

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

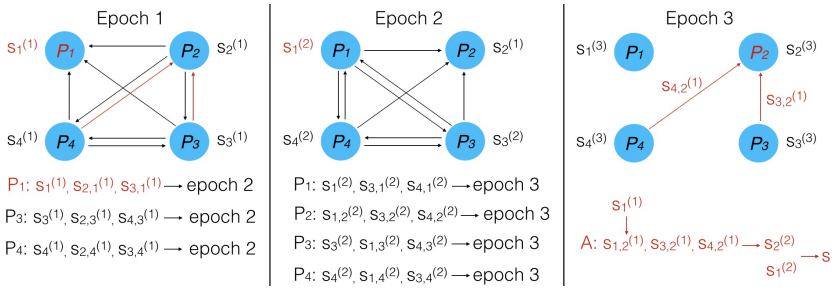

Fig. 10. We denote by $s_{j,i}^{(\tau)}$ the intermediate share obtained by party P_i from party P_j in epoch τ. P_j can construct its share for the next epoch $s_j^{(\tau+1)}$ from $n - t_a$ values $s_{j,i}^{(\tau)}$. The red quantities are in the view of the adversary. The red edges represent delayed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

Proof. We show that an adversary can break privacy by amassing shares corresponding to $t_a + 1$ parties in a single epoch. Then, we prove that protocols which avoid the prior attack do not satisfy liveness. For simplicity we first consider the case of non-interactive reshare protocols, and then handle the general case.

Non-interactive Reshare **protocol.** Consider $n = 4$ and $t_a = 1$. This counterexample is depicted in Figure 10 and can be extended to 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 destined to P_2. The adversary now has 3 messages, counting $s_1^{(1)}$, and is able to obtain $s_2^{(2)}$. Hence, it reconstructs 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 this attack, since there is no synchronizing signal instructing a corrupted party to restart before the first Reshare is completed. This could be addressed using erasures and/or interaction; however, we show that protocols that avoid this attack are not live.

Interactive Reshare protocols. 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 previous attack still applies. 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 (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 are not sufficient to implement this kind of channel. Say a party P_1 was delayed and is still in epoch τ, and all other parties advanced to epoch $\tau' > \tau$, updating their channel keys. But when honest parties start a new Reshare, they cannot use the key associated to P_1's epoch τ, because an adversary corrupting P_1 in epoch τ would learn shares from epoch τ' and break privacy. These are points (1) and (3). So until the adversary delivers the messages from epoch τ, P_1 is stuck, but this does not break liveness if the protocol is non-interactive. If point (2) is satisfied, the other parties need to already have the public key in the channel for epoch $\tau + 1$, 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 [10], 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.

To circumvent the result in Theorem 6, Yurek et al. [36] considered high reconstruction thresholds and defined local epochs such that a party can decide to not pass to a subsequent epoch even if it has all shares to do so, unlike our

definition based on completing a Reshare. Briefly, the impossibility does not hold because (i) a party decides to progress to the next epoch after receiving at least $n - t_a$ epoch τ messages (while in epoch τ), forcing the adversary to deliver at least these many messages to every party per epoch; (ii) combined with a high reconstruction threshold of $n - t_a$, the t_a shares held by the adversary in epoch τ and the at most t_a messages it could have delayed are not sufficient to reconstruct $s^{(\tau)}$, as $n - t_a > 2t_a$. We also mention that the constructions in [36] assume every party has Paillier key pairs that are not refreshed after corruptions.

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 Reconstruct threshold is at least $t_s + 1$ and the 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 is in an asynchronous state, so the adversary can 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 [10]. 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. B. Alexandru, E. Blum, J. Katz, and J. Loss. State machine replication under changing network conditions. Cryptology ePrint Archive, Report 2022/698, 2022. `https://eprint.iacr.org/2022/698`.

5. 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.
6. 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.
7. 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.
8. E. Blum, J. Katz, and J. Loss. Tardigrade: An atomic broadcast protocol for arbitrary network conditions. In M. Tibouchi and H. Wang, editors, *ASIACRYPT 2021, Part II*, volume 13091 of *LNCS*, pages 547–572. Springer, Heidelberg, Dec. 2021.
9. 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.
10. 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.
11. 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.
12. 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.
13. M. Castro and B. Liskov. Proactive recovery in a Byzantine-Fault-Tolerant system. In *4th Symposium on Operating Systems Design and Implementation*, 2000.
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*, pages 51–68, New York, NY, USA, 2017. ACM.
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://eprint.iacr.org/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 *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*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.
31. R. Ostrovsky and M. Yung. How to withstand mobile virus attacks (extended abstract). In L. Logrippo, editor, *10th ACM PODC*, pages 51–59. ACM, Aug. 1991.
32. M. Rambaud and A. Urban. Asynchronous dynamic proactive secret sharing under honest majority: Refreshing without a consistent view on shares. Cryptology ePrint Archive, Report 2022/619, 2022. `https://eprint.iacr.org/2022/619`.
33. D. A. Schultz, B. Liskov, and M. Liskov. Mobile proactive secret sharing. In *Proceedings of the 27th ACM symposium on Principles of Distributed Computing*, pages 458–458, 2008.
34. E. Shi. Foundations of distributed consensus and blockchains. Book manuscript, 2020.
35. R. Vassantlal, E. Alchieri, B. Ferreira, and A. Bessani. Cobra: Dynamic proactive secret sharing for confidential bft services. In *Symposium on Security and Privacy (SP)*, pages 1528–1528. IEEE Computer Society, 2022.
36. T. Yurek, Z. Xiang, Y. Xia, and A. Miller. Long live the honey badger: Robust asynchronous DPSS and its applications. Cryptology ePrint Archive, Report 2022/971, 2022. `https://eprint.iacr.org/2022/971`.