ABSTRACT

Title of Dissertation:	RESILIENT AND EFFICIENT CONSENSUS UNDER UNKNOWN NETWORK CONDITIONS
	Erica Blum Doctor of Philosophy, 2023
Dissertation Directed by:	Professor Jonathan Katz Department of Computer Science

Large-scale distributed services need to provide high throughput and low latency without sacrificing resilience. In particular, even if some servers crash or malfunction, the system as a whole should remain consistent. This challenge has been studied extensively in distributed computing and cryptography in the form of *consensus algorithms*. A consensus algorithm is an interactive protocol that allows honest (non-faulty) nodes to agree on a shared output in the presence of Byzantine (faulty) nodes, who may behave arbitrarily. Consensus algorithms have a long history in distributed computing, and are now receiving even more attention in the context of blockchain systems.

Consensus has frequently been studied in the context of two contrasting network models. In the *synchronous* network model, any message sent by an honest party will be delivered within a fixed bound; this bound is known to all parties and may be used as a protocol parameter. In the *asynchronous* network model, messages may be delayed for arbitrary lengths of time. For certain consensus problems and settings, the optimal fault tolerance is higher in the synchronous model than the asynchronous model (all else being equal). For example, assuming a public key infrastructure (PKI), the fundamental problem of *Byzantine agreement* (BA) for n parties is feasible for $t < n/2$ faults in the synchronous model, compared to only $t < n/3$ in the asynchronous model. On the other hand, synchronous consensus protocols can become stuck or even lose consistency if delays exceed the fixed bound.

In this dissertation, we consider a novel *network-agnostic* notion of security. Our central contribution is a suite of consensus protocols that achieve precisely defined security guarantees when run in either a synchronous or asynchronous network model, even when the parties are unaware of the network's true status. In addition, we provide matching impossibility results characterizing the best-possible security guarantees for this setting. We conclude by exploring a natural extension to network-agnostic security, in which protocols must remain secure in a setting where the underlying network status is not only unknown, but may switch between synchrony and asynchrony during a single protocol execution.

RESILIENT AND EFFICIENT CONSENSUS UNDER UNKNOWN NETWORK CONDITIONS

by

Erica Blum

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2023

Acknowledgments

First, I would like to thank my advisor, Jonathan Katz. Jon sets an exceptional example in every respect, including technical expertise, teaching ability, and near-instant email response time. Most of all, I hope that I have inherited some of Jon's unerring ability to ask exactly the right questions. In my research, whenever I've wandered off and gotten lost in the woods, Jon has been there to set me back on the right path.

I would also like to thank Julian Loss, who has been an invaluable mentor and constant collaborator during my PhD. Julian has devoted countless hours to helping me hammer out proofs—on whiteboards and Zoom calls, over Slack and email. Dozens of devious bugs were found and fixed thanks to his technical ability, patience, and attention to detail; any bugs that remain are entirely my own fault.

Sincere thanks to my committee members Jon, Julian, Dana Dachman-Soled, Dave Levin, and Ian Miers. I have always been able to count on them to give insightful feedback, suggest new perspectives, and inspire me to do my best work.

So many faculty and administrators at UMD supported me on this journey. I especially want to thank Leo Lampropoulos and José Manuel Calderón Trilla, whose doors were always open when I needed advice; and Michelle Mazurek, whose teaching and continued mentorship has deeply influenced the way I think about research and writing.

I was fortunate to intern with several incredible researchers, including Karim Eldefrawy, Dahlia Malkhi, Vipul Goyal, Elaine Shi, Shin'ichiro Matsuo, and Elette Boyle. Each of them welcomed me into their research group and generously offered their time and expertise.

A big thank you to all of my research collaborators over the years, a list that includes Alexander Russell, Andreea Alexandru, Chen-Da Liu Zhang, Tal Rabin, Kartik Nayak, Derek Leung, Ran Cohen, and Vinesh Sridhar in addition to those already mentioned. It is an amazing privilege to get to spend my time thinking interesting thoughts with such kind and talented people.

Lastly, none of this would have been possible without the help of my wonderful family and friends. I can't describe how thankful I am for their unconditional love and support.

Funding Acknowledgements

The material in this dissertation is based on work supported by the National Institute of Standards and Technology under financial assistance award #70NANB19H126; and by the National Science Foundation under grant #1837517.

Table of Contents

List of Figures

Chapter 1: Introduction

1.1 Building Resilient Distributed Systems with Asynchronous Fallback

In the 1980s, Pease, Shostak, and Lamport proposed the *Byzantine generals problem*, in which a set of devices must reach agreement, despite the traitorous plotting of some fraction of Byzantine (i.e., faulty) devices [1, 2]. They originally conceived of a setting in which a small number of devices (say, three or four) would be communicating over a local network; for example, a handful of sensors onboard an airplane might compare measurements to detect sudden changes in altitude or temperature. Even in this seemingly simple setting, reaching agreement is far from trivial – and as time has passed, the complexity of distributed systems has only increased. A typical application might require hundreds of servers to maintain a consistent state while processing requests from millions of users. Furthermore, many applications are run over the open Internet, making it difficulty to predict when (or even if) messages will arrive. In the best case, network connections might suffer from benign service interruptions or outages; in the worst case, the system could be deliberately targeted by denial-of-service attacks.

These challenges and more have motivated the study of *consensus problems*. Informally, a consensus problem is a theoretical problem in which honest (i.e., non-faulty) nodes must agree on a shared output in the presence of Byzantine (i.e., faulty) nodes, who may behave arbitrarily. Solutions usually take the form of *consensus protocols*. A consensus protocol, also called a *consensus algorithm*, is a set of rules or procedures that describes how an honest party should interact with its peers in order to arrive at a decision.

This dissertation focuses on two fundamental consensus problems: Byzantine agreement (BA) and state machine replication (SMR). In the Byzantine agreement problem, each device receives a local input value, and the honest devices must agree on a shared output value; in the state machine replication problem, each device receives a stream of unordered inputs and the honest devices must collectively agree on an ordered sequence of outputs. Informally, one can view BA as a "single-shot" form of consensus and SMR as its "multi-shot" counterpart.[1]

BA and SMR have been studied under a variety of *network models*. Informally, a network model is a collection of assumptions about the communication channels between parties. In the *synchronous* network model, any message sent by an honest party will be delivered within time Δ, where Δ is a fixed parameter known to all parties. In the *asynchronous* network model, messages may be delayed for arbitrary lengths of time. (Some work considers the *partially synchronous* model, in which messages are delivered within some time bound Δ that is unknown to the parties; we do not consider this model in our work.)

Assuming a public-key infrastructure (PKI), Byzantine agreement and SMR are known to be feasible for $t_s < n/2$ adversarial corruptions in a synchronous network, but only for $t_a < n/3$ faults in an asynchronous network. This leads us to ask whether it is possible to design a protocol that can withstand $n/3$ or more faults if the network happens to be synchronous, without entirely sacrificing security if the network happens to be asynchronous. More precisely, fix two thresholds t_a, t_s with $t_a \leq t_s$, and assume there is a trusted dealer who can distribute information (key

[1]State machine replication is a well-established technique in fault-tolerant distributed computing literature; for an in-depth tutorial, refer to Schneider's seminal work [3].

material, etc) to the parties in advance of the protocol execution. Is it possible to design *network-agnostic* protocols for BA and SMR that (1) tolerate t_s corruptions if they are run in a synchronous network and (2) tolerate t_a corruptions if they are run in an asynchronous network?

Depending on one's assumptions about the probabilities of different events, a network-agnostic protocol could be preferable to either a purely synchronous protocol (which loses security if the network is asynchronous) or a purely asynchronous one (which loses security if there are $n/3$ or more faults). To see why, consider the following concrete example. Fix t_a, t_s with $t_a < n/3 \leq t_s$ and $2t_s + t_a < n$, and let $f(t)$ be the probability that the number of faults is strictly greater than t. Suppose $f(t_a) = 1/10$, $f(\lfloor \frac{n-1}{3} \rfloor) = 1/20$, and $f(t_s) = 0$, and the probability that the network delay ever exceeds the assumed bound is $p_\Delta = 1/10$. Then a purely synchronous protocol fails to provide security with probability $p_\Delta = 1/10$, while a purely asynchronous protocol fails to provide security with probability $f(\lfloor \frac{n-1}{3} \rfloor) = 1/20$. But a network-agnostic protocol (with parameters t_a, t_s) fails to provide security only with probability $f(t_s) + p_\Delta \cdot f(t_a) = 1/100$.

This dissertation makes three key contributions. First, we propose a formal definition of network-agnostic security, and give complementary feasibility and impossibility results for network-agnostic BA and SMR, showing that network-agnostic security is achievable for corruption thresholds t_a, t_s if and only if $t_s + 2t_a < n$. This includes a network-agnostic BA protocol for optimal corruption thresholds (i.e., for any t_a, t_s such that $t_s + 2t_a < n$), and a simple proof-of-concept network-agnostic SMR protocol. The proof-of-concept SMR protocol, named TARDIGRADE,[2] supports optimal corruption thresholds and is secure against an adaptive

[2]Tardigrades, also called water bears, are microscopic animals known for their ability to survive in extreme environments.

adversary. When run among n parties, its per-block communication complexity is $O(n^4)$.[3]

Our second contribution is a pair of "next-generation" network-agnostic SMR protocols with improved communication complexity. The first of these protocols, UPDATE, has the same security guarantees as TARDIGRADE and improved per-block communication complexity of $O(n^3)$. The second, UPSTATE, further improves the per-block communication complexity to $O(n^2)$, at the cost of slightly weaker security guarantees (namely, its security is static rather than adaptive, and it supports a slightly restricted range of corruption thresholds).

As a third contribution, we extend our techniques to a challenging "variable" flavor of network-agnostic security, in which the network may switch arbitrarily between synchrony and asynchrony during a single protocol execution, and the adversary may hop between devices. We find that our protocols can be augmented with a "secure reboot" functionality in order to achieve a form of proactive security.

1.2 Related Work

The most closely related works are those that analyze security of a single protocol in multiple network models, or in a network model other than the one for which it was designed; however, we will begin with a brief overview of the most relevant work in the "pure" synchronous, partially synchronous, and asynchronous settings.

In the asynchronous setting, SMR protocols are often extensions of protocols for a related problem called *atomic broadcast* (cf. Definition 2.7). Canonical constructions for atomic broadcast in the asynchronous setting are based on multi-value validated asynchronous Byzan-

[3]For the purposes of this introduction, the communication complexity is stated as a function of the number of parties n. In later chapters, we present detailed analyses that account for all protocol parameters.

tine agreement or asynchronous common subset [4, 5, 6, 7, 8] and achieve cubic communication complexity for input sizes linear in n; on the synchronous side, more efficient constructions are known [9, 10].

Only a few existing protocols in the asynchronous setting tolerate *adaptive* corruptions: EPIC [11] and DAG-Rider [12] achieve adaptive security with cubic total communication complexity, and Dumbo2 [8] can be modified to achieve adaptive security by substituting a different MVBA protocol by the same authors [13]. Neither can be easily adapted to the network-agnostic setting.

Several prior works have demonstrated the (in)security of synchronous or partially synchronous protocols in an asynchronous network. For example, Miller et al. [6] describe an adversarial scheduler that causes PBFT [14] to lose liveness if the network is asynchronous rather than partially synchronous. Likewise, Nakamoto consensus (the protocol underlying the Bitcoin blockchain) was shown to be insecure if the network latency is too high or nodes become (temporarily) partitioned from the network [15, 16].

The work in this dissertation joins a relatively new line of work considering the security of a single protocol under different synchrony assumptions. In a 2016 paper, Liu et al. [17] proposed protocols that tolerate a minority of malicious faults in a synchronous network and a minority of *fail-stop* faults in an asynchronous network. Several other works on this topic were published concurrently with the work in this dissertation, including: Malkhi et al. [18] and Momose and Ren [19], who consider networks that may be either synchronous or *partially* synchronous; and Guo et al. [20] and Abraham et al. [10], who consider temporary disconnections between two synchronous network components.

A separate body of work [21, 22, 23, 24] has considered consensus protocols with *respon-*

siveness. A responsive protocol terminates (or outputs, in the case of atomic broadcast/SMR) in time proportional to the actual message-delivery time δ rather than the upper bound on the network-delivery time Δ. Kursawe [25] gives a protocol for an asynchronous network that terminates more quickly if the network is synchronous (but does not tolerate more faults in that case). Responsiveness is not the same as network-agnostic security, but responsive protocols and network-agnostic protocols are related in the sense that they achieve something "extra" under certain good conditions.

1.3 Outline of the Dissertation

Chapter 2 formally defines the network-agnostic model, and reviews relevant background and definitions.

Chapter 3 presents a network-agnostic Byzantine agreement protocol for any thresholds t_a, t_s such that $t_s + 2t_a < n$, and a matching impossibility result when $t_s + 2t_a \geq n$.

Chapter 4 presents a simple "first-generation" network-agnostic SMR protocol, TARDIGRADE, and a complementary lower bound.

Chapter 5 presents a pair of "next-generation" network-agnostic state machine replication protocols, UPDATE and UPSTATE; these protocols improve on TARDIGRADE in terms of communication complexity.

Finally, in Chapter 6, we introduce a natural extension to the network-agnostic model in which the underlying network can switch between synchrony and asynchrony, and present alternative versions of TARDIGRADE, UPDATE, and UPSTATE that are secure in this new model.

This dissertation includes material from previously published co-authored papers. Chapter 3 is based on a paper co-authored with Jonathan Katz and Julian Loss and published in the proceedings of the *Theory of Cryptography Conference*, 2019 [26].[4] Chapter 4 is based on a paper co-authored with Jonathan Katz and Julian Loss and published in the proceedings of *Advances in Cryptology—ASIACRYPT*, 2021 [27].[5] Chapters 5 and 6 are based on a paper co-authored with Andreea Alexandru, Jonathan Katz, and Julian Loss and published in the proceedings of *Advances in Cryptology—ASIACRYPT*, 2022 [28].[6]

[4]`https://doi.org/10.1007/978-3-030-36030-6_6`
[5]`https://doi.org/10.1007/978-3-030-92075-3_19`
[6]`https://doi.org/10.1007/978-3-031-22963-3_23`

Chapter 2: Preliminaries

2.1 System Model

We consider a system of n parties connected by point-to-point authenticated channels. Up to a fixed fraction of the parties may deviate arbitrarily from a protocol; those parties are called *Byzantine*. (We also sometimes refer to Byzantine parties as *corrupted* or *faulty*.) Parties who are not Byzantine are called *honest*.

Our protocols rely on a number of cryptographic primitives (detailed in Section 2.2); unless otherwise noted, these primitives are assumed to be initialized by a trusted dealer as part of some offline setup.

Any parties that are corrupted during an execution are modeled as being controlled by a single adversary. We will consider adversaries with a wide range of capabilities, as described below.

We separately consider static, adaptive, and mobile adversaries. A static adversary must choose which parties to corrupt prior to the start of the protocol, whereas an adaptive adversary is allowed to adaptively corrupt parties as the protocol progresses (up to a fixed threshold). A mobile adversary can move between parties over the course of the protocol.

We further distinguish two types of mobile adversary. The first type, which we call an *epoch-wise mobile adversary*, can move freely among parties between epochs, as long as the number of distinct parties corrupted in any single epoch does not exceed a fixed threshold. The second type, which we refer to as an *constrained mobile adversary*, is limited to moving between a certain subset of parties. More formally, the constrained mobile adversary is constrained with respect to thresholds $t' \leq t$, so that at most t distinct parties are ever corrupted over the course of an execution, and at most $t' \leq t$ of those parties are corrupted at any particular moment in time.

2.1.1 Network Models

This work introduces a novel *network-agnostic* model, where the underlying network behaves according to one of the classical network models (synchronous or asynchronous), but the parties do not know which.

In the classical synchronous network model, there is a bound Δ known to all parties such that any message sent by an honest party at time T is guaranteed to be delivered by time $T + \Delta$. The adversary can reorder or delay messages arbitrarily subject to this constraint. In the classical asynchronous model, there is no upper bound on the message delay, and the adversary can reorder or delay messages in any way as long as each message is eventually delivered.

In the *network-agnostic model*, the network may be either synchronous (for fixed Δ) or asynchronous, but the parties do not know which. Throughout most of this work, we consider a *static* network-agnostic model, in which the mode (synchrony or asynchrony) is fixed prior to each protocol execution. (When we refer to "the network-agnostic model," we are referring to this static version.) In Section 6 we introduce a *variable* form of the network-agnostic model; all details related to that model can be found there.

A key feature of the network-agnostic model is that the corruption threshold depends on the underlying network model. We denote the corruption threshold in the asynchronous and synchronous case by t_a and t_s, respectively. Throughout, we assume $t_a \leq t_s$, because t-security in the synchronous case implies t-security in the asynchronous case. Furthermore, except where otherwise noted, we assume $t_a < n/3$ and $t_s < n/2$. This is because most of the protocols we consider have well-known impossibility results that preclude beyond these thresholds. Lastly, as we will show in our impossibility results, network-agnostic security is feasible for most of the protocols we consider only if $2t_s + t_a < n$. Because these conditions frequently appear together, we define the following shorthand:

Condition ★: thresholds t_a, t_s satisfy $t_a < n/3$, $t_s < n/2$, $t_a \leq t_s$, and $2t_s + t_a < n$.

Bounds on message delays are just one aspect of a network model. Other key features of our models include the following:

- No after-the-fact removal: In all models we consider, we assume the adversary cannot perform after-the-fact removal. That is, once an honest party P_i has sent a message, the message must eventually be delivered even if P_i is later corrupted by an adaptive adversary.
- Rushing adversaries: Unless otherwise specified, we assume the adversary is *rushing*, i.e., it can wait to see all messages that were sent to corrupted parties in a given time step before sending its own messages.
- Local clocks: Each honest party has a local clock, which can be used to measure local time during a protocol execution.
  - When working in the synchronous model (either by itself or within the network-agnostic model), we assume all honest parties' clocks are synchronized and all honest parties begin running a protocol at the same time.

  - When working in the asynchronous model (either by itself or within the network-agnostic model), honest parties' clocks may not be synchronized and may run at different rates; we assume only that each honest party's clock progresses at some positive rate.

- Authenticated channels: In all models we consider, all parties are connected by pairwise authenticated channels.

2.2 Cryptographic Primitives

For simplicity, we use a single symbol κ to denote a security parameter for all schemes; in practice, each scheme's security parameters could be set independently.

2.2.1 Threshold Digital Signatures

In an n-party threshold signature scheme with threshold t, there is a public key pk, a vector of private keys $sk_1, \ldots, sk_n$, and a vector of public signature verification keys $(pk_1, \ldots, pk_n)$. Each party P_i receives sk_i, pk, and $(pk_1, \ldots, pk_n)$, and can use its secret key sk_i to create a signature share σ_i on a message m. A signature share from party P_i on a message m can be verified using the verification key pk_i; for this reason, we can also view a signature share as a signature by P_i on m. (In some of our subprotocols, the ability to combine shares is irrelevant and so it would suffice to assume a plain digital signature scheme; however, to simplify descriptions, we assume the same threshold signature scheme is used for all signing operations.) We often write $\langle m \rangle_i$ as a shorthand for the tuple (i, m, σ_i), where σ_i is a valid signature share on m with respect to P_i's verification key, and implicitly assume that invalid signature shares are discarded.

A set of $t + 1$ valid signature shares on the same message can be used to compute a signature for that message, which can be verified using the public key pk; a signature σ (resp. signature share σ_i) on a message m is called *valid* if it verifies successfully with respect to pk (resp. pk_i).

For our protocols, we require a scheme that achieves standard notions of correctness, security (unforgeability under chosen-message attack) and robustness (any set of at least $t + 1$ signature shares can be combined to yield a signature) against a probabilistic polynomial-time adversary.

To simplify our protocol descriptions, we assume that honest parties implicitly ignore invalid signature shares, and furthermore use some form of domain separation to ensure that signature shares are valid only in the particular protocol execution, iteration, etc. in which they were generated.

We assume that signature shares and signatures have size $O(\kappa)$; this is easy to ensure using a collision-resistant hash function.

2.2.2 Threshold Encryption

In an n-party threshold encryption scheme with threshold t, there is a public encryption key ek, a vector of private decryption keys $\mathsf{dk}_1, \ldots, \mathsf{dk}_n$, and a vector of public verification keys $\mathsf{vk}_1, \ldots, \mathsf{vk}_n$. 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 $t + 1$ correct decryption shares can be used to obtain the decryption m of the ciphertext c.

We consider an idealized encryption scheme for simplicity, but it can be instantiated using any CPA-secure scheme. Additionally, for the purposes of our communication complexity analyses, we assume that encrypting a message m of length $|m|$ produces a ciphertext of length $|m| + O(\kappa)$, and that decryption shares have length $O(\kappa)$; these properties are easy to ensure using standard KEM/DEM mechanisms.

2.2.3 Error-Correcting Codes

We adopt from Nayak et al. [29] 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 elements 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 $n - b \geq 2c + d$.

2.2.4 Coin-Flip Mechanism

We assume a *coin-flip mechanism* available as an atomic primitive. The mechanism, denoted $\mathsf{CoinFlip}^{t,\kappa}$, is parameterized by a threshold t and an output length κ, and takes an input k from some domain specified by the protocol. The input k is used for domain separation in case there are multiple invocations during a single protocol execution. When the parameters t and κ are clear from context, we sometimes omit them and simply write $\mathsf{CoinFlip}$.

We treat this mechanism as ideal functionality with the following behavior: upon receiving input k from $t+1$ parties, it generates an unbiased value $\mathsf{coin}_k \in \{0,1\}^\kappa$ and sends (k, coin_k) to all parties. (When run in an asynchronous network, messages to and from the functionality can be arbitrarily delayed.) This functionality has the useful property that if at most t parties are corrupted, then at least one honest party must provide input to the functionality before the adversary can learn the output.

Several protocols for realizing such a coin flip[1] in an asynchronous network, based on general assumptions, are known [30, 31, 32, 33]. For our purposes, we need a protocol that is secure for $t < n/3$ faults, and that *terminates* for $t' < n/2$ faults. Such protocols can be constructed using a threshold unique signature scheme [23, 34, 35, 36].

2.2.5 Committee Election Mechanisms

Selecting a small committee from among a large set of participants is a common technique in consensus protocols; for completeness, we will briefly describe two specific mechanisms that are used in this work.

For our purposes, a *committee election mechanism* is a protocol or functionality that determines a *committee*, i.e., a subset of the set of all parties. In general, such a mechanism should ensure that each party is a member of the committee with uniform and independent probability.

The first mechanism uses an unpredictable value (such as the output of a coinflip protocol, as described in Section 2.2.4) and a collision-resistant hash function to determine a committee of fixed size. Let κ denote a committee size parameter. Say H is a collision-resistant hash

[1]Some of these realize a p-weak coin flip, where honest parties agree on the coin only with probability $p < 1$. We can also rely on such protocols, at an increase in the expected round complexity by a factor of $O(1/p)$.

function that is known to all parties in advance, and let coin denote the unpredictable input. The unpredictable input coin determines a unique committee, which parties can locally compute as follows. First, each party computes the hash $H(\mathsf{coin}, i)$ for each $i \in [1..n]$. Then, the parties simply take the first κ parties in increasing order of their corresponding hash values.

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 [35, 37]. Let κ again denote a committee size parameter, and fix a VRF with κ-bit outputs. To elect a new committee, each party computes the output of the VRF on a prescribed input. A party is a member of this particular committee if the output of the VRF is less than $\mathsf{b} = \kappa 2^\kappa / n$. This mechanism can be idealized as determining a committee by flipping independent coins for each party. Throughout, we let $\chi_{s,n}$ denote the distribution for this idealized mechanism, i.e., the distribution that samples a subset of the n parties so that each party is included independently with probability s/n. Note that this mechanism produces committees of *expected* size κ, as opposed to committees of size exactly κ.

Several useful results about the size and composition of these committees can be derived from standard concentration bounds; for completeness, these are included in AppendixA.

2.3 Protocol Definitions

In this section, we provide property-based security definitions for various consensus protocols. Each definition specifies a set of standard properties that are required for security. Some definitions also include additional, weaker properties that will be relevant for our constructions.

In the following definitions (and throughout this work), we explicitly differentiate between liveness (i.e., generating output) and termination (i.e., exiting the protocol); in particular, we sometimes consider protocols that are live but not terminating.

We start by defining *broadcast*. A broadcast protocol allows a designated sender to share its input with a set of other parties called receivers. If the designated sender is honest, all honest receivers output the value input by the sender. If the designated sender is corrupted, we require only that the honest receivers output a consistent value.

Definition 2.1 (Broadcast)**.** Let Π be a protocol executed by parties P_1, $\ldots$, P_n, where a sender $P^* \in \{P_1, \ldots, P_n\}$ begins holding input $v^* \in \{0, 1\}$ and all parties are guaranteed to terminate.

- **Weak validity:** Π is *t-weakly valid* if the following holds whenever at most t of the parties are corrupted: if P^* is honest, then every honest party outputs either v^* or $\perp$.
- **Validity:** Π is *t-valid* if the following holds whenever at most t of the parties are corrupted: if P^* is honest, then every honest party outputs v^*.
- **Weak consistency:** Π is *t-weakly consistent* if the following holds whenever at most t of the parties are corrupted: there is a $v \in \{0, 1\}$ such that every honest party outputs either v or $\perp$.
- **Consistency:** Π is *t-consistent* if the following holds whenever at most t of the parties are corrupted: there is a $v \in \{0, 1, \perp\}$ such that every honest party outputs v.
- **Liveness:** Π is *t-live* if whenever at most t of the parties are corrupted, every honest party outputs a value in $\{0, 1\}$.

If Π is t-valid, t-consistent, and t-live, then we say Π is *t-secure*.

We reserve the term broadcast to mean a protocol with the security properties just defined, and use the term *multicast* to mean sending the same message to all parties. (In other words, we write "P broadcasts m" when P acts as the sender in an execution of a broadcast protocol with input m, and write "P multicasts m" when P simply sends m to all parties.)

Reliable broadcast is a slightly weaker form of broadcast. Reliable broadcast does not guarantee that receivers will output a value if the sender is corrupted; however, in this case it does guarantee that either all honest parties output the same value, or no honest party outputs a value.

Definition 2.2 (Reliable broadcast)**.** Let Π be a protocol executed by parties $P_1, \ldots, P_n$, where a designated sender $P^* \in \{P_1, \ldots, P_n\}$ begins holding input v^* and parties terminate upon generating output.

- **Validity:** Π is *t-valid* if the following holds whenever at most t parties are corrupted: if P^* is honest, then every honest party outputs v^*.

- **Consistency:** Π is *t-consistent* if the following holds whenever at most t parties are corrupted: either no honest party outputs a value, or all honest parties output the same value v.

If Π is t-valid and t-consistent, then we say it is *t-secure*.

Protocols for *Byzantine agreement* allow a set of parties to agree on a consistent output.

Definition 2.3 (Byzantine agreement)**.** Let Π be a protocol executed by parties $P_1, \ldots, P_n$, where each party P_i begins holding input $v_i \in \{0, 1\}$.

- **Weak validity:** Π is *t-weakly valid* if the following holds whenever at most t of the parties are corrupted: if every honest party's input is equal to the same value v, then every honest party outputs either v or $\perp$.

- **Validity:** Π is *t-valid* if the following holds whenever at most t of the parties are corrupted: if every honest party's input is equal to the same value v, then every honest party outputs v.
- **Validity with termination:** Π is *t-valid with termination* if the following holds whenever at most t of the parties are corrupted: if every honest party's input is equal to the same value v, then every honest party outputs v and terminates.
- **Weak consistency:** Π is *t-weakly consistent* if the following holds whenever at most t of the parties are corrupted: there is a $v \in \{0, 1\}$ such that every honest party outputs either v or $\perp$.
- **Consistency:** Π is *t-consistent* if the following holds whenever at most t of the parties are corrupted: there is a $v \in \{0, 1, \perp\}$ such that every honest party outputs v.
- **Liveness:** Π is *t-live* if whenever at most t of the parties are corrupted, every honest party outputs a value in $\{0, 1\}$.
- **Termination:** Π is *t-terminating* if whenever at most t of the parties are corrupted, every honest party terminates. Π has *guaranteed termination* if it is n-terminating.

If Π is t-valid, t-consistent, t-live, and t-terminating, then we say Π is *t-secure*.

While several of the above properties are not entirely standard, our notion of security matches the standard one. In particular, t-liveness and t-consistency imply that whenever at most t parties are corrupted, there is a $v \in \{0, 1\}$ such that every honest party outputs v. Note that t-validity with termination is weaker than t-validity plus t-termination, as the former does not require termination in case the inputs of the honest parties do not agree.

Next, we define *graded consensus* [38]. Graded consensus is a form of agreement in which each party outputs both a value $v \in \{0, 1, \perp\}$ and a *grade* $g \in \{0, 1, 2\}$. Protocols for graded consensus are sometimes used as a stepping stone to broadcast or Byzantine agreement.

Definition 2.4 (Graded consensus)**.** Let Π be a protocol executed by parties $P_1, \ldots, P_n$, where each party P_i begins holding input $v_i \in \{0, 1\}$.

- **Graded validity:** Π achieves t-*graded validity* if the following holds whenever at most t of the parties are corrupted: if every honest party's input is equal to the same value v, then all honest parties output $(v, 2)$.
- **Graded consistency:** Π achieves t-*graded consistency* if the following hold whenever at most t of the parties are corrupted: (1) If two honest parties output grades g, g', then $|g - g'| \leq 1$. (2) If two honest parties output (v, g) and (v', g') with $g, g' \geq 1$, then $v = v'$.
- **Liveness:** Π is t-*live* if whenever at most t of the parties are corrupted, every honest party outputs (v, g) with either $v \in \{0, 1\}$ and $g \geq 1$, or $v = \perp$ and $g = 0$.

If Π achieves t-graded validity, t-graded consistency, and t-liveness then we say Π is t-*secure*.

Protocols for *asynchronous common subset* (ACS) allow parties to agree on a set of values. We will often want our ACS protocols to satisfy a *set quality* property, specifically, that any output must contain at least some minimal number of honest parties' inputs; however, this property is not part of the core security definition.

Definition 2.5 (Asynchronous common subset (ACS))**.** Let Π be a protocol executed by parties $P_1, \ldots, P_n$, where each P_i begins holding input $v_i \in \{0, 1\}^*$, and parties output sets of at most n values.

- **Validity:** Π is *t-valid* if the following holds whenever at most t parties are corrupted: if every honest party's input is equal to the same value v, then every honest party outputs $\{v\}$.

- **Consistency:** Π is *t-consistent* if whenever at most t parties are corrupted, all honest parties output the same set S.

- **Liveness:** Π is *t-live* if whenever at most t parties are corrupted, every honest party generates output.

- **Set quality:** Π has *t-set quality* if the following holds whenever at most t parties are corrupted: if an honest party outputs a set S, then S contains the input of at least one honest party.

- **Validity with termination:** Π is *t-valid with termination* if, whenever at most t parties are corrupted and every honest party's input is equal to the same value v, then every honest party outputs $\{v\}$ and terminates.

- **Termination:** Π is *t-terminating* if whenever at most t parties are corrupted, every honest party generates output and terminates.

If Π is t-consistent, t-valid, and t-live, we say it is *t-secure*.

Our next definition is for *block agreement*. Block agreement is a useful abstraction of a building block that features prominently in several of our constructions, namely, agreement on values with a particular structure. Block agreement is related to existing notions of agreement (especially validated multivalued Byzantine agreement [39] and the synod protocol of Abraham et al. [40]) but is not identical.

Definition 2.6 (Block agreement)**.** Let Π be a protocol executed by parties $P_1, \ldots, P_n$, where parties terminate upon generating output. Define a pre-block to be a vector of length n, such that the i^{th} entry is either $\perp$ or a pair (m_i, σ_i), where m_i is an arbitrary value and σ_i is a signature on m_i by P_i, and define a *k-quality pre-block* to be a pre-block with at least k non-$\perp$ entries.

- **Validity:** Π is *t-valid* if whenever at most t of the parties are corrupted and every honest party's input is an $(n-t)$-quality pre-block, then every honest party outputs an $(n-t)$-quality pre-block.

- **Consistency:** Π is *t-consistent* if whenever at most t of the parties are corrupted, every honest party outputs the same pre-block B.

If Π is t-valid and t-consistent, then we say it is *t-secure*.

Finally, we define *atomic broadcast* (ABC) and *state machine replication* (SMR). At a high level, the purpose of both atomic broadcast and state machine replication is to allow parties to maintain agreement on an ever-growing, ordered log of values called *transactions*. Transactions are output in the form of batches called *blocks*. Party P_i's output is represented as a write-once array, denoted by $\mathsf{blocks}_i = \mathsf{blocks}_i[1], \mathsf{blocks}_i[2], \ldots$. The array's entries are initialized to a special character $\perp$. During the protocol, parties write to each index of the array in ascending order.

The two terms are sometimes used interchangeably to refer to any kind of repeated agreement (including blockchains); however, we will follow Momose et al. in making the distinction that an SMR protocol must achieve a property called *external validity* (also called *public verifiability*) [19]. Informally, a consensus protocol is externally valid if it is possible for an external observer to validate the output as authentic; this property will be formalized in the definition of state machine replication below.

Our protocols proceed in logical intervals or iterations called *epochs*. Informally, epoch j is the logical interval in which parties agree on block j. We say that P_i *outputs a block in epoch* j (or simply *outputs block* j) when P_i writes a value to $\mathsf{blocks}_i[j]$. For convenience, we let $\mathsf{blocks}_i[k:\ell]$ denote the contiguous subarray $\mathsf{blocks}_i[k], \ldots, \mathsf{blocks}_i[\ell]$ and let $\mathsf{blocks}_i[:\ell]$ denote the prefix $\mathsf{blocks}_i[1:\ell]$.

Each party P_i has a local buffer buf_i. Transactions are added to parties' local buffers by some mechanism external to the protocol (e.g., via a gossip protocol). Importantly, a particular transaction tx may be provided as input to different parties at arbitrary times, and may be provided as input to some honest parties but not others. Whenever P_i outputs a block, it clears those transactions from its buffer.

Definition 2.7 (Atomic broadcast (ABC))**.** Let Π be a protocol executed by parties $P_1, \ldots, P_n$ who receive transactions as input and locally maintain arrays blocks as described above.

- **Completeness:** Π is t-*complete* if the following holds whenever at most t parties are corrupted: for all $j > 0$, every honest party outputs a block in epoch j.

- **Consistency:** Π is t-*consistent* if the following holds whenever at most t parties are corrupted: if an honest party outputs a block B in epoch j then all honest parties output B in epoch j.

- **Liveness:** Π is t-*live* if the following holds whenever at most t parties are corrupted: if every honest party receives transaction tx as input, then every honest party eventually outputs a block that contains tx.

If Π is t-consistent, t-live, and t-complete, then we say it is t-*secure*.

The definition of state machine replication is almost identical to the definition of atomic broadcast; the key difference is the external validity property. Accordingly, we slightly modify the output syntax, so that parties output both a block B and proof π by writing (B, π) to $\mathsf{blocks}[e]$.

Definition 2.8 (State Machine Replication (SMR))**.** Let Π be a protocol executed by n parties $P_1, \ldots, P_n$, and let Verify be a predetermined Boolean verification function. Let params be some public parameters (e.g., for a PKI). Each P_i receives transactions as input and locally maintains an array blocks_i as described above.

- **Completeness:** Π is t-*complete* if the following holds whenever at most t parties are corrupted: for all $j > 0$, every honest party outputs a block in epoch j.
- **Consistency:** Π is t-*consistent* if the following holds whenever at most t parties are corrupted: if an honest party outputs a block B in epoch j then all honest parties output B in epoch j.
- **Liveness:** Π is t-*live* if the following holds whenever at most t parties are corrupted: if every honest party is provided a transaction tx as input, then every honest party eventually outputs a block that contains tx.
- **External validity:** Π is t-*externally consistent* if the following holds whenever at most t parties are corrupted: for all e, an honest party writes (B, π) to $\mathsf{blocks}_i[e]$ if and only if $\mathsf{Verify}(\mathsf{params}, \mathsf{blocks}_i, \pi, e) = 1$.

If Π is t-consistent, t-live, t-complete, and t-externally valid, then we say it is t-*secure*.

Chapter 3: Network-Agnostic Byzantine Agreement

The central result of this chapter is a network-agnostic Byzantine agreement protocol, HBA^{t_a,t_s} (abbrev. HBA), that is network-agnostically secure for any thresholds t_s, t_a satisfying condition $\bigstar$. HBA is constructed from a series of simpler building blocks.[1] The first major building block, $\mathsf{SBA}^{t_a,t_s}_{\text{HBA}}$ (abbrev. SBA), is a Byzantine agreement protocol that achieves validity and consistency if the network is *synchronous* and at most t_s parties are corrupted. Furthermore, in this case SBA is guaranteed to generate output within time $n \cdot \Delta$. On the other hand, if SBA is run in an asynchronous network with at most t_a corruptions, it only achieves a property we call weak validity, which states that if every honest party inputs the same value v then each honest party outputs either v or a special value $\perp$. The second major building block, $\mathsf{ABA}^{t_a,t_s}_{\text{HBA}}$ (abbrev. ABA), is a Byzantine agreement protocol that achieves the standard notion of security (with termination) if the network is *asynchronous* and at most t_a parties are corrupted. However, if ABA is run in a synchronous network with up to t_s corruptions, then it only achieves a weaker property, which we call validity with termination.

The full protocol HBA proceeds as follows. Each party P_i begins by running SBA on its initial input v_i for time at most $n \cdot \Delta$. At time $n \cdot \Delta$, if P_i has received an output b_i (other than $\perp$) from SBA, then it will input b_i to ABA; if P_i has not not received an output from SBA, it will

[1]Network-agnostic BA was sometimes called "hybrid BA" in earlier versions of this work, hence the abbreviation HBA.

abandon that subprotocol and input its original input v_i to ABA. Then, P_i simply waits for ABA to output a value v. When it does, P_i outputs v and terminates.

To see intuitively why combining SBA and ABA in this way yields the desired security properties, consider the two possible scenarios. If the network is synchronous and at most t_s parties are corrupted, then all honest parties receive the same output v from SBA by time $n \cdot \Delta$. Thus, even though ABA only guarantees validity with termination in this situation, the honest parties are already in agreement when they begin running ABA, and so this is enough to ensure the honest parties terminate with a consistent output. Next, suppose the network is asynchronous and at most t_a parties are corrupted. If all honest parties have the same initial input v, then the weak validity property of SBA ensures that all honest parties input v to ABA, and so validity of ABA guarantees all honest parties terminate with output v, as desired. Furthermore, even if not all honest parties had the same input, then security of ABA ensures that all honest parties eventually terminate with consistent output.

The rest of this chapter is organized as follows: in Section 3.1 and 3.2, we introduce a series of synchronous and asynchronous building blocks. In Section 3.3, we combine those building blocks to form the full network-agnostic BA protocol. Finally, in Section 3.4, we prove that network-agnostic BA is impossible when $t_s + 2t_a \geq n$, thus showing that our protocol achieves the best possible corruption tolerance.

3.1 Synchronous BA with Partial Asynchronous Fallback

In this section we show a BA protocol that is secure for some threshold t_s of corrupted parties when run in a synchronous network, and achieves weak validity (though liveness and weak consistency may not hold) for a lower threshold t_a even when run in an asynchronous network.

Our protocol relies on a variant of the Dolev-Strong broadcast protocol [41] as a subroutine. Since we use a slightly non-standard version of that protocol, we describe it in Figure 3.1 for completeness. In the protocol, we say that a message consisting of a value b and a set of signatures Σ is an *r-correct message* (from the point of view of a party P_i) if Σ contains valid signatures on b from the sender P^* and $r-1$ other distinct parties (not including P_i itself).

Protocol $\mathsf{DS}^{P^*}_{\mathbf{HBA}}$

Initialize $S_i = \emptyset$.

Round 1: If $P_i = P^*$: create a signature σ^* on input b_i and send $(b_i, \Sigma = \{\sigma^*\})$ to all parties.

Rounds $\mathbf{r} = \mathbf{1}, \ldots, \mathbf{n} - \mathbf{1}$: Upon receiving an r-correct message (b, Σ) in round r, add b to S_i. If $r < n-1$, then also compute a signature σ_i on b, and send $(b, S_i \cup \{\sigma_i\})$ to all parties in the following round. (This is done at most once for each (b, r) pair.)

Output determination: At time $(n-1) \cdot \Delta$, if S_i contains one value, then output that value and terminate. In any other case, output $\perp$ and terminate.

Figure 3.1: A broadcast protocol from the perspective of party P_i with designated sender P^*.

Lemma 3.1. DS is a secure broadcast protocol (as defined in Definition 2.1) that satisfies the following properties:

1. When run in a synchronous network, it is n-consistent and n-valid.
2. When run in an asynchronous network, it is n-weakly valid.

Proof. The standard analysis of the Dolev-Strong protocol shows that, when run in a synchronous network with any number of corrupted parties, $S_i = S_j$ for any honest parties P_i, P_j. This implies n-consistency. Since an honest P^* sends a 1-correct message to all honest parties, and the attacker cannot forge signatures of the honest sender, n-validity holds. The second claim follows because an attacker cannot forge the signature of an honest P^*. $\square$

We now define our synchronous BA protocol, SBA, using the broadcast protocol DS as a subprotocol. The protocol is parameterized by corruption thresholds t_a, t_s. (Here and in later protocols, we include t_s as a parameter even though the protocol does not explicitly depend on it, as it will be relevant to the security analysis.)

Protocol $\mathsf{SBA}^{t_a,t_s}_{\mathbf{HBA}}(b_i)$

- Run DS^{P_i} as the designated sender with input b_i, and for each $j \neq i$, run DS^{P_j} with P_j as the designated sender.
- For each $j \in [n]$, let b^i_j denote the output of DS^{P_j}. If there are at least $2t_a + 1$ values b^i_j that are in $\{0, 1\}$, output the majority of those values and terminate. (If there is a tie, output a fixed default bit.) Otherwise, output $\perp$ and terminate.

Figure 3.2: A Byzantine agreement protocol from the view of party P_i, parameterized by thresholds t_a, t_s.

Theorem 3.1. For any t_a, t_s satisfying condition $\bigstar$, SBA is a secure Byzantine agreement protocol (as defined in Definition 2.3) that satisfies the following properties:

1. When the protocol is run in a synchronous network, it is t_s-secure.
2. When the protocol is run in an asynchronous network, it is t_a-weakly valid.

Moreover, the protocol has guaranteed termination in both cases, and when run in a synchronous network every honest party terminates in time at most $n \cdot \Delta$.

Proof. The claim about termination is immediate.

When run in a synchronous network with t_s corrupted parties, at least $n - t_s > 2t_a$ of the executions of DS result in boolean output for all honest parties (by n-validity of DS) and so all honest parties generate boolean output in SBA; this proves t_s-liveness. By n-consistency of DS, all honest parties agree on the $\{b_j\}$ values they obtain and hence SBA is t_s-consistent (in fact, it is n-consistent). Finally, n-validity of DS implies that when all honest parties begin holding the same input $b \in \{0, 1\}$, then all honest parties will have b as their majority value. This proves t_s-validity (in fact, the protocol is t-valid for any $t < n/2$).

For the second claim, assume all honest parties begin holding the same input b, and t_a parties are corrupted. Any honest party P_i who generates boolean output must have at least $2t_a + 1$ boolean values $\{b^i_j\}$, of which at most t_a of these can be equal to $\bar{b}$. Hence, any honest party who generates boolean output will in fact output b. $\square$

3.2 Asynchronous Byzantine Agreement with Enhanced Validity

In this section, we present an asynchronous BA protocol that achieves validity in the presence of a higher number of corruptions. (Throughout this section, we always assume an asynchronous network model unless otherwise noted.) Formally, we will prove the following theorem:

Theorem 3.2. For any t_a, t_s satisfying condition ★, there is an n-party protocol for Byzantine agreement that, when run in an asynchronous network, is t_a-secure and also achieves t_s-validity with termination.

Our protocol design is inspired by Mostéfaoui et al. [33]. To simplify the presentation and proofs, we present the protocol in a modular fashion. First, in Section 3.2.1, we construct a subprotocol $\mathsf{Prop}_{\mathrm{HBA}}$ that allows a designated 'proposer' to propose a value. That protocol becomes the central building block of a graded consensus protocol $\mathsf{GC}_{\mathrm{HBA}}$. Finally, in Section 3.2.3, the graded consensus subprotocol becomes the foundation of the full Byzantine agreement protocol $\mathsf{ABA}^{t_a,t_s}_{\mathrm{HBA}}$.

3.2.1 A Value-Proposal Subprotocol

We begin by describing a subprotocol for proposing values $\mathsf{Prop}_{\mathrm{HBA}}^{t_a,t_s}$ (abbrev. $\mathsf{Prop}_{\mathrm{HBA}}$), shown in Figure 4.4. The rest of the section is devoted to proving properties of $\mathsf{Prop}_{\mathrm{HBA}}$ in preparation for later sections.

Protocol $\mathsf{Prop}_{\mathbf{HBA}}^{t_a,t_s}(b_i)$

1. Set $S := \emptyset$.
2. Send $(\mathsf{prepare}, b_i)$ to all parties.
3. Upon receiving $(\mathsf{prepare}, b)$ for the same $b \in \{0, 1, \lambda\}$ from strictly more than t_s parties: If $(\mathsf{prepare}, b)$ has not been sent, send $(\mathsf{prepare}, b)$ to all parties.
4. Upon receiving $(\mathsf{prepare}, b)$ for the same $b \in \{0, 1, \lambda\}$ from at least $n - t_s$ parties, add b to S.
5. Upon adding the first value $b \in \{0, 1, \lambda\}$ to S, send $(\mathsf{propose}, b)$ to all parties.
6. Once at least $n - t_s$ messages $(\mathsf{propose}, b)$ have been received on values $b \in S$: let $S^* \subseteq S$ be the set of values carried by those messages. Output S^* (but continue running).

Figure 3.3: A subprotocol for proposing values, parameterized by thresholds t_a, t_s.

Lemma 3.2. Assume $t_a < n - 2 \cdot t_s$ parties are corrupted in an execution of $\mathsf{Prop}_{\mathrm{HBA}}$. If two honest parties P_i, P_j output $\{b\}$, $\{b'\}$, respectively, then $b = b'$.

Proof. Since P_i outputs $\{b\}$, it must have received at least $n - t_s$ messages $(\mathsf{propose}, b)$, of which at least $n - t_s - t_a$ of those were sent by honest parties. Similarly, P_j must have received at least $n - t_s - t_a$ messages $(\mathsf{propose}, b')$ that were sent by honest parties. If $b \neq b'$, then because $2 \cdot (n - t_s - t_a)$ is strictly greater than the number of honest parties $n - t_a$, this would mean that some honest party sent propose messages on two different values, which is impossible. □

Lemma 3.3. Assume $t_a \leq t_s$ parties are corrupted in an execution of $\mathsf{Prop}_{\mathrm{HBA}}$. If no honest party inputs b, then no honest party outputs a set containing b.

Proof. If b was not input by any honest party, then at most $t_a \leq t_s$ messages $(\mathsf{prepare}, b)$ are sent in step 2. Thus, no honest party ever sends a message $(\mathsf{prepare}, b)$, and consequently no honest party ever sends a message $(\mathsf{propose}, b)$. It follows that no honest party ever adds b to its set S, and so no honest party outputs a set S^* containing b. $\square$

Lemma 3.4. Assume t_a parties are corrupted in an execution of $\mathsf{Prop}_{\mathrm{HBA}}$, where $t_a < n - 2 \cdot t_s$ and $t_a \leq t_s$. If an honest party sends a message $(\mathsf{propose}, b)$, all honest parties add b to S.

Proof. Suppose some honest party P_i sends $(\mathsf{propose}, b)$. Then P_i must have received at least $n - t_s$ messages $(\mathsf{prepare}, b)$. At least $n - t_s - t_a > t_s$ of these must have been sent by honest parties, and so eventually all other honest parties also receive strictly more than t_s messages $(\mathsf{prepare}, b)$. We thus see that every honest party will eventually send $(\mathsf{prepare}, b)$. Therefore, every honest party will eventually receive at least $n - t_a \geq n - t_s$ messages $(\mathsf{prepare}, b)$, and consequently every honest party will add b to S. $\square$

Lemma 3.5. Assume t_a parties are corrupted in an execution of $\mathsf{Prop}_{\mathrm{HBA}}$, where $t_a < n - 2 \cdot t_s$ and $t_a \leq t_s$. If all honest parties hold one of two different inputs, then all honest parties output.

Proof. We first argue that every honest party sends a $\mathsf{propose}$ message. There are $n - t_a$ honest parties, so at least $\frac{1}{2}(n - t_a) > t_s$ honest parties must have the same input b. Therefore, all honest parties receive strictly more than t_s messages $(\mathsf{prepare}, b)$. Consequently, all honest parties will eventually send $(\mathsf{prepare}, b)$. Thus, every honest party receives $n - t_a \geq n - t_s$ messages $(\mathsf{prepare}, b)$ and adds b to S. In particular, S is nonempty and so every honest party sends a $\mathsf{propose}$ message.

Each honest party thus receives at least $n - t_a \geq n - t_s$ $\mathsf{propose}$ messages sent by honest parties. By Lemma 3.4, for any b proposed by an honest party, all honest parties eventually have

$b \in S$. Thus, every honest party eventually receives at least $n - t_s$ propose messages for values in their set S, and therefore all honest parties terminate. □

Unfortunately, the full security properties of $\mathsf{Prop}_{\mathsf{HBA}}$ are not guaranteed to hold if the number of corruptions exceeds t_a. However, $\mathsf{Prop}_{\mathsf{HBA}}$ does achieve a notion of validity in the presence of up to $t_s < n/2$ corrupted parties; this fact will be useful later on.

Lemma 3.6. Assume $t_s < n/2$ parties are corrupted in an execution of $\mathsf{Prop}_{\mathsf{HBA}}$. If all honest parties hold the same input b, then all honest parties output $S^* = \{b\}$.

Proof. Suppose t_s parties are corrupted, and all honest parties hold the same input b. In step 2, all $n - t_s$ honest parties send $(\mathsf{prepare}, b)$, and so all honest parties add b to S. Any prepare messages on other values in step 2 are sent by the $t_s < n - t_s$ corrupted parties, and so no honest party ever adds a value other than b to S. Thus, all $n - t_s$ honest parties send their (single) propose message $(\mathsf{propose}, b)$ in step 5. It follows that every honest party outputs $S^* = \{b\}$ in step 6. □

3.2.2 Asynchronous Graded Consensus with Enhanced Validity

We now present our graded consensus protocol, $\mathsf{GC}_{\mathsf{HBA}}^{t_a,t_s}$ (abbrev. $\mathsf{GC}_{\mathsf{HBA}}$). In the graded consensus protocol, parties participate in two consecutive instances of the propose subprotocol $\mathsf{Prop}_{\mathsf{HBA}}$ before outputting a value v and grade g. Informally, the grade $g \in \{0, 1, 2\}$ represents a party's confidence in their output,with 0 indicating low confidence and 2 indicating high confidence. Complete pseudocode for the protocol appears in Figure 3.4.

For the moment, we will analyze $\mathsf{GC}_{\mathsf{HBA}}$ assuming that parties continue participating indefinitely. (Later, when $\mathsf{GC}_{\mathsf{HBA}}$ is used as a subprotocol, the calling protocol will be responsible for ensuring that parties "safely" abandon all instances of $\mathsf{GC}_{\mathsf{HBA}}$.)

Protocol $\mathsf{GC}^{t_a,t_s}_{\mathbf{HBA}}(b_i)$

- Set $b_1 := b_i$.
- Run $\mathsf{Prop}_{\mathsf{HBA}}$ on input b_1, and let S_1 denote the output.
- If S_1 contains a single value b^*, then set $b_2 := b^*$. Otherwise (if S_1 contains more than one value) set $b_2 := \lambda$.
- Run $\mathsf{Prop}_{\mathsf{HBA}}$ using input b_2, and let S_2 denote the output.
- If S_2 contains a single value $b_{\mathsf{out}} \neq \lambda$, output $(b_{\mathsf{out}}, 2)$. Else if $S_2 = \{b_{\mathsf{out}}, \lambda\}$ for some $b_{\mathsf{out}} \neq \lambda$, output $(b_{\mathsf{out}}, 1)$. Else if $S_2 = \{\lambda\}$, output $(\perp, 0)$.

Figure 3.4: A protocol for graded consensus from the perspective of party P_i, parameterized by thresholds t_a, t_s.

The following lemmas prove that $\mathsf{GC}_{\mathsf{HBA}}$ achieves partial security (specifically, graded validity) for up to t_s corruptions, and full security for up to t_a corruptions.

Lemma 3.7. If $t_s < n/2$, then $\mathsf{GC}_{\mathsf{HBA}}$ achieves t_s-graded validity (as defined in Definition 2.4).

Proof. Suppose t_s parties are corrupted, and every honest party's input is equal to the same value b. By Lemma 3.6, all honest parties have $S_1 = \{b\}$ following the first execution of $\mathsf{Prop}_{\mathsf{HBA}}$, and so use b as the input for the second execution of $\mathsf{Prop}_{\mathsf{HBA}}$. By the same reasoning, all honest parties have $S_2 = \{b\}$ after the second execution of $\mathsf{Prop}_{\mathsf{HBA}}$. Thus, all honest parties output $(b, 2)$. □

Lemma 3.8. Assume t_a, t_s satisfy condition ★. Then $\mathsf{GC}_{\mathsf{HBA}}$ achieves t_a-graded consistency (as defined in Definition 2.4).

Proof. Suppose t_a parties are corrupted. First, we show that the grades output by two honest parties P_i, P_j differ by at most 1. The only way this can possibly fail is if one of the parties (say, P_i) outputs a grade of 2. P_i must then have received $S_2 = \{b\}$, for some $b \in \{0, 1\}$, as its output from the second execution of $\mathsf{Prop}_{\mathsf{HBA}}$. It follows from Lemma 3.2 that another party P_j could not have received $S_2 = \{\lambda\}$. Therefore, it is not possible for P_j to output grade 0.

Next, we show that any two honest parties that output nonzero grades must output the same value. Observe first that there is a bit b such that the inputs of all the honest parties to the second execution of $\mathsf{Prop}_{\mathrm{HBA}}$ lie in $\{b, \lambda\}$. (Indeed, if all honest parties set $b_2 := \lambda$ this claim is immediate. On the other hand, if some honest party sets $b_2 := b \in \{0, 1\}$ then they must have $S^* = \{b\}$; but then Lemma 3.2 implies that any other honest party who sets b_2 to anything other than λ will set it equal to b as well.) Lemma 3.3 thus implies that no honest party outputs a set S^* after the second execution of $\mathsf{Prop}_{\mathrm{HBA}}$ that contains a value other than b or λ. Thus, any two honest parties that output a nonzero grade must output the same value b. $\square$

Lemma 3.9. Assume t_a, t_s satisfy condition $\bigstar$. Then $\mathsf{GC}_{\mathrm{HBA}}$ achieves t_a-liveness (as defined in Definition 2.4).

Proof. All honest parties hold input in $\{0, 1\}$ in the first execution of $\mathsf{Prop}_{\mathrm{HBA}}$, so Lemma 3.5 shows that all honest parties generate output in that execution. As in the proof of Lemma 3.8, there is a bit b such that the inputs of all the honest parties to the second execution of $\mathsf{Prop}_{\mathrm{HBA}}$ lie in $\{b, \lambda\}$; so, applying Lemma 3.5 again, that execution also produces output for all honest parties. Moreover, by Lemma 3.3, the set S^* output by any honest party is a nonempty subset of $\{b, \lambda\}$, i.e., is either $\{b\}$, $\{b, \lambda\}$, or $\{\lambda\}$. Thus, $\mathsf{GC}_{\mathrm{HBA}}$ is t_s-live. $\square$

3.2.3 Asynchronous Byzantine Agreement with Enhanced Validity

We now present our asynchronous Byzantine agreement protocol, $\mathsf{ABA}^{t_a,t_s}_{\mathrm{HBA}}$ (abbrev. ABA). Like its constituent building blocks, ABA guarantees full security for up to t_a corruptions and limited security (namely, validity with termination) for up to t_s corruptions. For the construction, we assume an atomic primitive $\mathsf{CoinFlip}$ that allows all parties to generate and learn an unbiased value $\mathsf{coin}_k \in \{0,1\}$ for $k = 1, 2, \ldots$; for an explanation of how this primitive can be realized, we refer back to Chapter 2. Pseudocode for the protocol appears in Figure 3.5.

To simplify the presentation, we begin by describing ABA as a non-terminating BA protocol; we conclude by briefly discussing how to add termination using existing techniques.

```
Protocol $\mathsf{ABA}^{t_a,t_s}_{\mathbf{HBA}}(b_i)$

Set $b^* := b_i$, $\mathsf{committed} = \mathsf{false}$, and $k := 1$. Then do:

    1. Run $\mathsf{GC}_{\mathrm{HBA}}$ on input $b^*$, and let $(b_1, g_1)$ denote the output.
    2. $\mathsf{coin}_k \leftarrow \mathsf{CoinFlip}(k)$.
    3. If $g_1 = 2$, set $b^* := b_1$; else set $b^* := \mathsf{coin}_k$.
    4. Run $\mathsf{GC}_{\mathrm{HBA}}$ on input $b^*$, and let $(b_2, g_2)$ denote the output.
    5. If $g_2 = 2$ and $\mathsf{committed} = \mathsf{false}$, set $\mathsf{committed} := \mathsf{true}$ and send $\langle \mathsf{commit}, b_2 \rangle_i$ to
       all parties.
    6. Set $k := k + 1$ and repeat from (1).

To terminate:

    • Upon receiving valid signature shares $\sigma_{i_1}, \ldots, \sigma_{i_{t_s+1}}$ on $(\mathsf{commit}, b_{\mathsf{out}})$ (for the same
      $b_{\mathsf{out}}$) from at least $t_s + 1$ distinct parties, form a combined signatures $\sigma$, send
      $(\mathsf{notify}, b_{\mathsf{out}}, \sigma)$ to all parties, output $b_{\mathsf{out}}$, and terminate.
    • Upon receiving $(\mathsf{notify}, b_{\mathsf{out}}, \sigma)$ such that $\sigma$ is a valid combined signature on
      $(\mathsf{commit}, b_{\mathsf{out}})$, forward $(\mathsf{notify}, b_{\mathsf{out}}, \sigma)$ to all parties, output $b_{\mathsf{out}}$ and terminate.
```

Figure 3.5: A Byzantine agreement protocol from the view of party P_i, parameterized by t_a, t_s.

When a party terminates in ABA, they will also stop participating in any $\mathsf{Prop}_{\mathrm{HBA}}$ or $\mathsf{GC}_{\mathrm{HBA}}$ executions. Because the security properties of $\mathsf{Prop}_{\mathrm{HBA}}$ and $\mathsf{GC}_{\mathrm{HBA}}$ hold for an asynchronous network, honest parties dropping out may cause the executions to lose liveness, but cannot break security otherwise. We argue below that if any honest party terminates ABA with output b, then all honest parties terminate; furthermore, by the security properties of the subprotocols, all honest parties output the same value b.

In the following, we say a message $(\mathsf{commit}, b, \sigma)$ from party P_i is *valid* if $b \in \{0,1\}$ and

σ is a valid signature from P_i on (commit, b). Furthermore, we say that a set of signatures is a *certificate for* b if it contains valid signatures on (commit, b) from at least $t_s + 1$ distinct parties.

Lemma 3.10. If $t_s < n/2$, then protocol ABA satisfies t_s-validity with termination (as defined in Definition 2.3).

Proof. Suppose there are at most t_s corrupted parties and all honest parties input $b \in \{0, 1\}$. All honest parties use input b in the first execution of GC_{HBA} in the first iteration; t_s-graded validity of GC_{HBA} (cf. Lemma 3.7) implies they all output $(b, 2)$ from that execution. Thus, all honest parties ignore the result of the coin flip and run a second instance of GC_{HBA} using input b, again unanimously obtaining $(b, 2)$ as output. Therefore, all honest parties either send a commit message on b in step 5 of the first iteration of ABA, or have already received (and forwarded) a certificate Σ containing at least t_s signatures on b. Furthermore, honest parties will receive at most $t_s < t_s+1$ commit messages on $b' \neq b$. Therefore, all honest parties eventually receive valid commit messages on b from $n - t_s \geq t_s + 1$ distinct parties (via either a single notify message or individual commit messages), output b, and terminate. $\square$

Lemma 3.11. Let t_a, t_s satisfy condition $\bigstar$. Then ABA satisfies t_a-liveness and t_a-consistency (as defined in Definition 2.3). Moreover: (1) some honest party sends a commit message within an expected constant number of iterations, and (2) if some honest party sends a commit message on a bit b, then all honest parties terminate with output b.

Proof. Assume t_a parties are corrupted. Consider an iteration k of the protocol by which no honest party has yet sent a commit message. Let Agree be the event that all honest parties use the same input to the second execution of GC_{HBA} in that iteration. If Agree occurs, then t_s-graded validity of GC_{HBA} implies that all honest parties will obtain a grade of 2 in that execution and hence at least one honest party will send a commit message in iteration k. We show that Agree occurs with probability at least $1/2$. We distinguish two cases:

- If some honest party outputs $(b, 2)$ in the first execution of GC_{HBA} in iteration k: by t_a-graded consistency of GC_{HBA}, all honest parties output either $(b, 2)$ or $(b, 1)$ in that execution of GC_{HBA}. Since coin_k is not revealed until after the first honest party generates output for that execution of GC_{HBA}, this means b is chosen independently of coin_k. If $\mathsf{coin}_k = b$, which occurs with probability $1/2$, then all honest parties will use the same input in the second execution of GC_{HBA} in iteration k.

- If no honest party outputs $(b, 2)$ after the first execution of GC_{HBA}: then all honest parties will use coin_k as their input in the second execution of GC_{HBA} in iteration k.

Thus, in expected constant rounds *some* honest party sends a commit message. We next show that if some honest party P_i sends a commit message on b, then all other honest parties terminate with output b.

Let k be the first iteration in which some honest party sends a commit message, and let P_i be the first honest party to have sent a commit message on b in iteration k. P_i must have seen $(b, 2)$ as the output of the second execution of GC_{HBA} in iteration k. By t_a-graded consistency of GC_{HBA}, every honest party who obtains output from that execution of GC_{HBA} will receive either $(b, 1)$ or $(b, 2)$. This means that all honest parties who continue running will input b to the next iteration of GC_{HBA}. By t_s-graded validity of GC_{HBA}, honest parties will continue to input b to each iteration of GC_{HBA}, ignoring the coin, for as long as they continue running. Thus, we see that no honest party will ever send a commit message on $\bar{b}$. Hence, there will never exist a set of commit

messages on $\bar{b}$ from at least $t_s + 1$ distinct parties, and so no honest party will ever output $\bar{b}$.[2]

We now consider two cases: either some honest party terminates, or (by t_a-liveness and t_s-validity of $\mathsf{GC}_{\mathsf{HBA}}$) all honest parties reach step 5 of iteration $k + 1$ and receive output $(b, 2)$. In the latter case, all honest parties will send a commit message on b. Therefore, eventually all honest parties will receive at least $n - t_a \geq t_s + 1$ commit messages on b and can terminate with output b. In the case that some honest party terminates, liveness of $\mathsf{GC}_{\mathsf{HBA}}$ may be lost. However, in order for that party to terminate, they must have already received sufficient signatures on b, and forwarded those signatures as a certificate Σ to all other parties. Therefore, all honest parties eventually receive Σ and terminate with output b. $\square$

Lemma 3.12. For any t_a, t_s satisfying condition $\bigstar$, there is an n-party protocol for Byzantine agreement that, when run in an asynchronous network, achieves t_s-validity, t_a-consistency, and t_a-termination (as defined in Definition 2.3).

Proof. If $t_s < t_a$, then we parameterize ABA with $t_s = t_a$. Lemma 3.11 and Lemma 3.10 immediately show that ABA is t_a-secure (and therefore also t_s-valid if $t_s < t_a$).

Otherwise, if $t_s \geq t_a$, we can apply Lemma 3.11 directly. The claim thus follows from Lemmas 3.10 and 3.11. $\square$

3.3 A Network-Agnostic Byzantine Agreement Protocol

In this section, we present our full network-agnostic Byzantine agreement protocol, HBA. The protocol is parameterized by thresholds t_a, t_s (and the number of parties, n); we will assume throughout that t_a, t_s satisfy condition $\bigstar$. The protocol description assumes subprotocols SBA

[2]It is important to note that $\mathsf{GC}_{\mathsf{HBA}}$ cannot lose *safety* if honest parties drop out, only liveness. Because it is an asynchronous protocol, it necessarily tolerates parties dropping out.

and ABA with the following properties:

- SBA is an n-party BA protocol that is t_s-secure and terminates by time $n \cdot \Delta$ when run in a synchronous network, and is t_a-weakly valid and eventually terminates when run in an asynchronous network.

- ABA is an n-party BA protocol that is t_a-secure and t_s-valid with termination in either a synchronous or asynchronous network.

We proved that there exist protocols with precisely these properties in Theorems 3.1 and 3.2, respectively.

Protocol $\mathsf{HBA}^{t_a,t_s}(b_i)$

- Run SBA on input b_i for time $n \cdot \Delta$.
- At time $n \cdot \Delta$, let b denote the output of SBA (if SBA has not output, let $b = \perp$). If $b \neq \perp$, set $b^* := b$; otherwise set $b^* := b_i$.
- Run $\Pi_{\mathsf{ABA}}^{t_s}$ using input b^*. Once $\Pi_{\mathsf{ABA}}^{t_s}$ outputs a value b_{out}, output b_{out} and terminate.

Figure 3.6: A Byzantine agreement protocol from the perspective of party P_i, parameterized by thresholds t_a, t_s.

Theorem 3.3. Let n, t_a, t_s be as above. Then HBA is t_s-secure when run in a synchronous network and t_a-secure when run in an asynchronous network (defining t-security as in Definition 2.3).

Proof. First consider the case when HBA is run in a synchronous network, and at most t_s parties are corrupted. By t_s-security of SBA, after running SBA there is a value $b \neq \perp$ such that every honest P_i's intermediate value b^* is equal to b. Moreover, if every honest party input the same value b', then $b^* = b'$. By t_s-validity with termination of ABA, all honest parties terminate and agree on their output from HBA, proving t_s-consistency, t_s-liveness, and t_s-termination. Moreover, if every honest party's original input was equal to the same value b', then the output of ABA (and thus of HBA) is equal to b'. This proves t_s-validity.

Next consider the case when HBA is run in an asynchronous network, and at most t_a parties are corrupted. The protocol inherits t_a-consistency, t_a-liveness, and t_a-termination from t_a-security of ABA, and so it only remains to argue t_a-validity. Assume every honest party's initial input is equal to the same value b'. Then for each honest party P_i, the output of SBA must be in $\{b', \perp\}$ due to t_a-weak validity and termination of SBA. Thus, each honest party sets the intermediate value b^* to b'. It follows from t_s-validity with termination (note $t_a \leq t_s$) of ABA that all honest parties output b' and terminate. $\square$

3.4 Optimal Thresholds for Network-Agnostic Byzantine Agreement

We show here that our positive result from the previous section is tight. That is:

Theorem 3.4. For any n, if $t_a \geq n/3$ or $t_a + 2 \cdot t_s \geq n$ there is no n-party protocol for Byzantine agreement that is t_s-secure in a synchronous network and t_a-secure in an asynchronous network (defining t-security as in 2.3).

The case of $t_a \geq n/3$ follows from Toueg's classical impossibility result for asynchronous consensus [42], so we can focus on the case where $t_a < n/3$ but $t_a + 2 \cdot t_s \geq n$. Using a similar line of reasoning to Toueg's original proof, we show that secure BA is also impossible in this case. In fact, we prove a stronger claim, namely that a BA protocol cannot be both t_a-weakly consistent in an asynchronous network and t_s-valid in a synchronous network; these properties are a subset of the properties required for secure BA, and so Theorem 3.4 follows immediately.

Lemma 3.13. Fix n, t_a, t_s with $t_a + 2t_s \geq n$. If an n-party Byzantine agreement protocol is t_s-valid in a synchronous network, then it cannot also be t_a-weakly consistent in an asynchronous network (as defined in Definition 2.3).

Proof. Assume $t_a + 2t_s = n$ and fix a BA protocol Π. Partition the n parties into sets S_0, S_1, S_a where $|S_0| = |S_1| = t_s$ and $|S_a| = t_a$, and consider the following experiment:

- Parties in S_0 run Π using input 0, and parties in S_1 run Π using input 1. All communication between parties in S_0 and parties in S_1 is blocked (but all other messages are delivered within time Δ).

- Create virtual copies of each party in S_a, call them S_a^0 and S_a^1. Parties in S_a^0 run Π using input 0, and communicate only with each other and parties in S_0. Parties in S_a^1 run Π using input 1, and communicate only with each other and parties in S_1.

Consider an execution of Π in a synchronous network where parties in S_1 are corrupted and simply abort, and all remaining (honest) parties use input 0. The views of the honest parties in this execution are distributed identically to the views of $S_0 \cup S_a^0$ in the above experiment. In particular, t_s-validity of Π implies that all parties in S_0 output 0. Analogously, all parties in S_1 output 1.

Next consider an execution of Π in an asynchronous network where parties in S_a are corrupted, and run Π using input 0 when interacting with S_0 while running Π using input 1 when interacting with S_1. Moreover, all communication between the (honest) parties in S_0 and S_1 is delayed indefinitely. The views of the honest parties in this execution are distributed identically to the views of $S_0 \cup S_1$ in the above experiment, yet the conclusion of the preceding paragraph shows that weak consistency is violated. □

Chapter 4: Network-Agnostic State Machine Replication

We now turn our attention from Byzantine agreement to *state machine replication* (SMR). At a high level, the goal of SMR is to allow a group of parties to combine individual unordered input streams into a single, unified output stream, so that all parties agree on a growing sequence of outputs.

This chapter presents a network-agnostic SMR protocol, TARDIGRADE. In later chapters, we will build on the techniques used in TARDIGRADE to construct two additional SMR protocols, UPDATE and UPSTATE.

Network-agnostic SMR can be approached similarly to network-agnostic BA: we will pair a synchronous building block with an asynchronous building block, connecting the two halves in such a way that at least one will succeed and the other, at worst, neither helps or hinders. However, the various building blocks become significantly more complex as we move from one-shot agreement on binary values to multi-shot agreement on arbitrary values.[1]

A central piece of our construction is a novel protocol for the fundamental problem of *asynchronous common subset* (ACS). Our ACS protocol achieves non-standard security properties that turn out to be generally useful for constructing protocols in a network-agnostic setting; it has already served as a key ingredient in follow-up work [44] on network-agnostic secure com-

[1]Generic transformations from BA to SMR are known in other settings [43], but we are not aware of any such transformations that immediately translate to our setting.

putation.

Even though we are ultimately interested in protocols for SMR, most of this chapter is actually devoted to building network-agnostic *atomic broadcast* (ABC).[2] Toward that end, in Section 4.1 and Section 4.2 we construct network-agnostic subprotocols for asynchronous common subset (ACS) and block agreement (BLA), respectively. These subprotocols play a central role in the construction of the network-agnostic atomic broadcast protocol in Section 4.3. Then, in Section 4.4, we briefly discuss how to extend the atomic broadcast protocol to a full state machine replication protocol. Finally, in Section 4.5, we prove an impossibility result showing that the corruption thresholds tolerated by TARDIGRADE are optimal for network-agnostic state machine replication.

4.1 Asynchronous Common Subset with Enhanced Validity

In this section, we construct an asynchronous common subset (ACS) protocol that achieves full security for a lower threshold (t_a) and achieves validity for a higher threshold (t_s). More precisely, for any t_a, t_s satisfying condition $\bigstar$, we show a t_a-secure ACS protocol that achieves t_a-termination, t_s-validity with termination, and t_a-set quality (as defined in Definition 2.5). Throughout this section, we assume an asynchronous network,[3] though the protocol naturally achieves the same guarantees in a synchronous network.

Our protocol is adapted from the ACS protocol of Ben-Or et al. [45] (later adapted by Miller et al. [6]). We present our construction in three steps: first, we introduce a reliable broad-

[2]SMR is closely related to atomic broadcast, to the extent that the terms are sometimes used interchangeably in the literature. In this dissertation, we will use the convention that SMR protocols must achieve an additional external validity property that is not required for ABC protocols.

[3]The ‘asynchronous’ in ACS is historical; in principle one can design ACS protocols for any network model.

cast protocol with separate security guarantees for each of the dual thresholds t_s, t_a. The reliable broadcast protocol then forms the basis of a novel ACS protocol ($\mathsf{ACS}^*_{\mathsf{tdg}}$, described in Figure 4.2) that is t_a-secure and has t_a-set quality, but is non-terminating. Finally, we construct a terminating protocol ($\mathsf{ACS}_{\mathsf{tdg}}$, described in Figure 4.3) using the non-terminating ACS as a subprotocol. The terminating protocol inherits security and set quality from the non-terminating protocol, and additionally achieves t_a-termination and t_s-validity with termination.

4.1.1 Reliable Broadcast with Higher Validity

This section presents $\mathsf{BB}^{t_a,t_s}_{\mathsf{tdg}}$ (abbrev. $\mathsf{BB}_{\mathsf{tdg}}$), an asynchronous reliable broadcast protocol that is t_s-valid and t_a-consistent with $O(n^2\,|v|)$ communication complexity. The protocol design, described in Figure 4.1, is based on Bracha's asynchronous reliable broadcast protocol [46]; the key difference is that the corruption thresholds for consistency and validity can be set separately via the parameters t_a and t_s.

$\mathsf{BB}^{t_a,t_s}_{\mathbf{tdg}}$

- Set $\mathsf{ready} = \mathsf{false}$.
- If $P_i = P^*$: multicast input v^*.
- Upon receiving initial value v^* from P^*, multicast (echo, v^*).
- Upon receiving (echo, v^*) messages on the same value v^* from $n - t_s$ distinct parties: if $\mathsf{ready} = \mathsf{false}$, set $\mathsf{ready} = \mathsf{true}$ and multicast (ready, v^*).
- Upon receiving (ready, v^*) messages on the same value v^* from $t_s + 1$ distinct parties: if $\mathsf{ready} = \mathsf{false}$, set $\mathsf{readied} = \mathsf{true}$ and multicast (ready, v^*).
- Upon receiving (ready, v^*) messages on the same value v^* from $n - t_s$ distinct parties: output v^* and terminate.

Figure 4.1: A reliable broadcast protocol with designated sender P^*, from the perspective of party P_i.

Lemma 4.1. If $t_s < n/2$ then $\mathsf{BB}_{\mathsf{tdg}}$ is t_s-valid (as defined in Definition 2.2).

Proof. Assume there are at most t_s corrupted parties, and the sender is honest. All honest parties receive the same value v^* from the sender, and consequently send $(\textsf{echo}, v^*)$ to all other parties. Since there are at least $n-t_s$ honest parties, all honest parties receive $(\textsf{echo}, v^*)$ from at least $n-t_s$ different parties, and as a result send $(\textsf{ready}, v^*)$ to all other parties. By the same argument, all honest parties receive $(\textsf{ready}, v^*)$ from at least $n-t_s$ parties, and so can output v^* (and terminate). Fix any $v \neq v^*$; to complete the proof, we argue that no honest party will output v. Note first that no honest party will send $(\textsf{echo}, v)$. Thus, any honest party receives $(\textsf{echo}, v)$ from at most t_s other parties. Since $t_s < n - t_s$, no honest party will ever send $(\textsf{ready}, v)$. By the same argument, this shows that any honest party receives $(\textsf{ready}, v)$ from at most t_s other parties, and hence will not output v. □

Lemma 4.2. Fix $t_a \leq t_s$ with $t_a + 2 \cdot t_s < n$. Then $\textsf{BB}_{\textsf{tdg}}$ is t_a-consistent (as defined in Definition 2.2).

Proof. Suppose at most t_a parties are corrupted, and that an honest party P_i outputs v. Then P_i must have received $(\textsf{ready}, v)$ from at least $n - t_s$ distinct parties, at least $n - t_s - t_a \geq t_s + 1$ of whom are honest. Thus, all honest parties receive $(\textsf{ready}, v)$ from at least $t_s + 1$ distinct parties, and so all honest parties send $(\textsf{ready}, v)$ to everyone. It follows that all honest parties receive $(\textsf{ready}, v)$ from at least $n - t_a \geq n - t_s$ parties, and so can output v as well. To complete the proof, we argue that an honest party cannot output $v' \neq v$. We argued above that every honest party sends $(\textsf{ready}, v)$ to everyone. Since $t_a < t_s + 1$, each honest party must have sent $(\textsf{ready}, v)$ in response to receiving $(\textsf{echo}, v)$ from at least $n - t_s$ distinct parties. If some honest party outputs v' then, arguing similarly, every honest party must have received $(\textsf{echo}, v')$ from at least $n - t_s$ distinct parties. But this is a contradiction, since an honest party sends only a single echo message but $2 \cdot (n - t_s) - t_a > n$. $\square$

4.1.2 A Non-Terminating ACS Protocol

The non-terminating ACS protocol $\mathsf{ACS}^{*t_a,t_s}_{\text{tdg}}$ (usually abbreviated $\mathsf{ACS}^{*}_{\text{tdg}}$) is based on two building blocks. The first building block is the asynchronous reliable broadcast protocol that was just constructed, BB_{tdg}; in this section, we will denote it as BB for short. The second building block is a standard asynchronous Byzantine agreement (ABA) protocol, which we will denote as BA. This building block can be instantiated using any ABA protocol that is secure for $t_a < n/3$ corruptions and has $O(n^2)$ communication complexity, such as the ABA protocol of Mostéfaoui et al. [33].

Each execution of $\mathsf{ACS}^{*}_{\text{tdg}}$ contains n parallel instances of reliable broadcast ($\mathsf{BB}_1, \ldots, \mathsf{BB}_n$) and n parallel instance of Byzantine agreement ($\mathsf{BA}_1, \ldots, \mathsf{BA}_n$). Instance BB_i is used to broadcast P_i's input v_i, while instance BA_i is used to agree on whether to include P_i's input in the final output. "Accepted" indices (i.e., indices i s.t. BA_i output 1) are recorded in a local variable S^*. At the end of the protocol, if a party observes a majority value v in the set of values $\{v'_i\}_{i \in S^*}$, it outputs the singleton set $\{v\}$; otherwise, it outputs $\{v'_i\}_{i \in S^*}$, i.e., the set of all values broadcast by parties in S^*.

Lemma 4.3. Fix t_a, t_s satisfying condition $\bigstar$, and assume there are at most t_s corrupted parties during some execution of $\mathsf{ACS}^{*}_{\text{tdg}}$. If an honest party P_i outputs a set S_i, then $\exists v_j \in S_i$ such that v_j was input by an honest party P_j.

Proof. We show that P_i's output S_i always includes a value that was output from an execution of BB where the corresponding sender is honest. The lemma then follows from t_s-validity of BB.

$\mathsf{ACS}_{\mathbf{tdg}}^{*t_a,t_s}(v_j)$

- Set $\mathsf{commit} := \mathsf{false}$ and $S^* := \emptyset$.
- Broadcast v_j by running BB as the sender, and for each $i \neq j$ run an execution of BB with P_i as the sender.
- Upon BB_i terminating with output v_i': if P_j has not yet begun running BA_i then begin running it with input 1.
- Upon BA_i terminating with output 1: add i to S^*.
- Upon setting $|S^*|$ to $n - t_a$: for any BA_i that P_j has not yet begun running, begin running BA_i with input 0.

Predicates:

$C_1(v)$:	At least $n - t_s$ executions $\{\mathsf{BB}_i\}_{i\in[n]}$ have output v.
C_1:	There exists v for which $C_1(v)$ is true.
$C_2(v)$:	$\lvert S^*\rvert \geq n - t_a$, all executions $\{\mathsf{BA}_i\}_{i\in[n]}$ have terminated, and a strict majority of the executions $\{\mathsf{BB}_i\}_{i\in S^*}$ have output v.
C_2:	There exists v for which $C_2(v)$ is true.
C_3:	$\lvert S^*\rvert \geq n - t_a$, all executions $\{\mathsf{BA}_i\}_{i\in[n]}$ have terminated, and all executions $\{\mathsf{BB}_i\}_{i\in S^*}$ have terminated.

Output conditions:

(Event 1) If $C_1(v) = \mathsf{true}$ for some v and $\mathsf{commit} = \mathsf{false}$ then:
set $\mathsf{commit} := \mathsf{true}$ and output $\{v\}$.

(Event 2) If $C_1 = \mathsf{false}$, $C_2(v) = \mathsf{true}$ for some v, and $\mathsf{commit} = \mathsf{false}$ then:
set $\mathsf{commit} := \mathsf{true}$ and output $\{v\}$.

(Event 3) If $C_1 = C_2 = \mathsf{false}$, $C_3 = \mathsf{true}$, and $\mathsf{commit} = \mathsf{false}$ then:
set $\mathsf{commit} := \mathsf{true}$ and output $\{v_i'\}_{i\in S^*}$.

Figure 4.2: An ACS protocol, from the perspective of party P_j with input v_j.

Suppose P_i outputs in response to Event 1, so S_i is a singleton set $\{v\}$. P_i must have received v as output from at least $n - t_s$ broadcasts. Because $n - 2t_s > t_a \geq 0$, at least one of those corresponds to an honest sender.

Next, suppose P_i outputs in response to Event 2. Again, S_i is a singleton set $\{v\}$. P_i must have seen at least $\lfloor \frac{|S^*|}{2} \rfloor + 1$ broadcast instances terminate with output v, and furthermore $|S^*| \geq n - t_a$. Therefore, P_i has seen at least $\left\lfloor \frac{n-t_a}{2} \right\rfloor + 1 \geq \left\lfloor \frac{2t_s}{2} \right\rfloor + 1 > t_s$ broadcasts terminate with output v. Since there are at most t_s corrupted parties, at least one of those executions must correspond to an honest sender.

Finally, suppose P_i outputs in response to Event 3, so $S_i = \{v_i'\}_{i \in S^*}$. Since there are at most t_s corrupted parties and $|S^*| - t_s \geq n - t_a - t_s > t_s \geq 0$, at least one party in S^* is honest. $\square$

Lemma 4.4. For any t_a, t_s satisfying condition $\bigstar$, ACS^* is t_s-valid (as defined in Definition 2.5).

Proof. Assume at most t_s parties are corrupted, and all honest parties have the same input v. By t_s-validity of BB, at least $n - t_s$ broadcast instances (namely, the instances with an honest sender) will eventually output v. It follows that all honest parties eventually set $C_1(v) = \mathsf{true}$, at which point they will output $\{v\}$ if they have not already generated output. It only remains to show that there is no other set an honest party can output.

If an honest party outputs S in response to Events 1 or 2, then S is a singleton set. Since all honest parties have input v, Lemma 4.3 implies $S = \{v\}$.

To conclude, we show that no honest party outputs in response to Event 3. Assume toward a contradiction that some honest party P outputs in response to Event 3. Then P must have seen BB_i terminate (say, with output v_i) for all $i \in S^*$. Since also $|S^*| \geq n - t_a > 2t_s$, a majority of those executions $\{\mathsf{BB}_i\}_{i \in S^*}$ correspond to honest senders and so (by t_s-validity of BB) resulted in output v. But then $C_2(v)$ would be true for P, and P would not generate output due to Event 3. $\square$

Lemma 4.5. Fix t_a, t_s satisfying condition $\bigstar$, and assume at most t_a parties are corrupted during an execution of ACS^*. If honest parties P_1, P_2 output sets S_1, S_2, respectively, then $S_1 = S_2$.

Proof. Say P_1 outputs in response to Event i and P_2 outputs in response to Event j, and assume without loss of generality that $i \leq j$. We consider the different possibilities.

First, assume $i = 1$, so Event 1 occurs for P_1 and $S_1 = \{v_1\}$ for some value v_1. We have the following sub-cases:

- If $j = 1$, then $S_2 = \{v_2\}$ for some value v_2. P_1 and P_2 must have each seen some set of at least $n - t_s > n/2$ executions of $\{\mathsf{BB}_i\}$ output v_1 and v_2, respectively. The intersection of these sets is non-empty; thus, t_a-consistency of BB implies that $v_1 = v_2$ and hence $S_1 = S_2$.

- If $j = 2$, then once again $S_2 = \{v_2\}$ for some v_2. P_2 must have $|S^*| \geq n - t_a$, and must have seen at least $\left\lfloor \frac{|S^*|}{2} \right\rfloor + 1 \geq \left\lfloor \frac{n-t_a}{2} \right\rfloor + 1$ executions of $\{\mathsf{BB}_i\}$ output v_2. Moreover, P_1 must have seen at least $n - t_s$ executions of $\{\mathsf{BB}_i\}$ output v_1. Since

$$n - t_s + \left\lfloor \frac{n - t_a}{2} \right\rfloor + 1 \geq n - t_s + \left\lfloor \frac{2t_s}{2} \right\rfloor + 1 > n, \tag{4.1}$$

  these two sets of executions must have a non-empty intersection. But then t_a-consistency of BB implies that $v_1 = v_2$ and hence $S_1 = S_2$.

- If $j = 3$, then P_2 must have seen all executions $\{\mathsf{BB}_i\}_{i \in S^*}$ terminate, where $|S^*| \geq n - t_a$. We know P_1 has seen at least $n - t_s$ executions $\{\mathsf{BB}_i\}_{i \in [n]}$ output v_1, and so (by t_a-consistency of BB) there are at most t_s executions $\{\mathsf{BB}_i\}_{i \in [n]}$ that P_2 has seen terminate with a value other than v_1. The number of executions of $\{\mathsf{BB}_i\}_{i \in S^*}$ that P_2 has seen terminate with output v_1 (which is at least $(n - t_a) - t_s > t_s$) is thus strictly greater than the number of executions $\{\mathsf{BB}_i\}_{i \in S^*}$ that P_2 has seen terminate with a value other than v_1 (which is at most t_s). But then $C_2(v_1)$ would be true for P_2. We conclude that Event 3 cannot occur for P_2.

Next, assume $i = j = 2$, so Event 2 occurs for P_1 and P_2. Then $S_1 = \{v_1\}$ and $S_2 = \{v_2\}$ for some v_1, v_2. Both P_1 and P_2 must have seen all executions $\{\mathsf{BA}_i\}_{i\in[n]}$ terminate. By t_a-consistency of BA, they must therefore agree on S^*. P_1 must have seen a majority of the executions $\{\mathsf{BB}_i\}_{i\in S^*}$ output v_1; similarly, P_2 must have seen a majority of the executions $\{\mathsf{BB}_i\}_{i\in S^*}$ output v_2. Then t_a-consistency of BB implies $v_1 = v_2$.

Finally, consider the case where P_2 outputs in response to Event 3 and P_1 outputs in response to Event 2 or 3. As above, t_a-consistency of BA ensures that P_1 and P_2 agree on S^*. Moreover, P_2 must have seen all executions $\{\mathsf{BB}_i\}_{i\in S^*}$ terminate, but without any value being output by a majority of those executions. But then t_a-consistency of BB implies that P_1 also does not see any value being output by a majority of those executions, and so Event 2 cannot occur for P_1; thus, Event 3 must have occurred for P_1. Therefore, t_a-consistency of BB implies that P_1 outputs the same set as P_2. □

Lemma 4.6. For any t_a, t_s satisfying condition ★, ACS^* is t_a-live (as defined in Definition 2.5).

Proof. It follows easily from t_a-security of BB and BA that if any honest party generates output then all honest parties generate output, so consider the case where no honest parties have (yet) generated output. Let H denote the indices of the honest parties. By t_s-validity of BB, all honest parties eventually see the executions $\{\mathsf{BB}_i\}_{i\in H}$ terminate, and so all honest parties input a value to the executions $\{\mathsf{BA}_i\}_{i\in H}$. By t_a-security of BA, all honest parties eventually see those executions terminate and agree on their outputs. There are now two cases:

- If all executions $\{\mathsf{BA}_i\}_{i\in H}$ output 1, then it is immediate that all honest parties have $|S^*| \geq n - t_a$.

- If BA_i outputs 0 for some $i \in H$, then (by t_a-validity of BA) some honest party P must have used input 0 when running BA_i. But then P must have seen at least $n - t_a$ other executions $\{\mathsf{BA}_i\}$ output 1. By t_a-consistency of BA, this implies that all honest parties see at least $n - t_a$ executions $\{\mathsf{BA}_i\}$ output 1, and hence have $|S^*| \geq n - t_a$.

Since all honest parties have $|S^*| \geq n - t_a$, they all execute $\{\mathsf{BA}_i\}_{i \in [n]}$. Once again, t_a-termination of BA implies that all those executions will eventually terminate. Finally, if $i \in S^*$ for some honest party P then P must have seen BA_i terminate with output 1; then t_a-validity of BA implies that some honest party used input 1 when running BA_i and hence has seen BB_i terminate. It follows that P will see BB_i terminate. As a result, we see that every honest party can (at least) generate output due to Event 3. □

Lemma 4.7. For any t_a, t_s satisfying condition ★, ACS^* has t_a-set quality (as defined in Definition 2.5).

Proof. If an honest party P outputs $S = \{v\}$ due to Event 1, then P has seen at least $n - t_s$ executions $\{\mathsf{BB}_i\}$ terminate with output v. Of these, at least $n - t_s - t_a > 0$ must correspond to honest senders. By t_s-validity of BB, those honest parties must have all had input v, and so set quality holds. Alternatively, say P outputs a set $\{v\}$ due to Event 2. Then P must have $|S^*| \geq n - t_a$, and at least $\lfloor \frac{|S^*|}{2} \rfloor + 1 \geq \lfloor \frac{n-t_a}{2} \rfloor + 1 > t_a$ of the executions $\{\mathsf{BB}_i\}_{i \in S^*}$ output v. At least one of those executions must correspond to an honest party, and that honest party must have had input v (by t_s-validity of BB); thus, set quality holds. Finally, if P output a set S due to Event 3, then S contains every value output by $\{\mathsf{BB}_i\}_{i \in S^*}$ with $|S^*| \geq n - t_a$. Since S^* must contain at least one honest party, set quality follows as before. □

Theorem 4.1. For any t_a, t_s satisfying condition ★, ACS^* is t_a-secure and t_s-valid, and has t_a-set quality (as defined in Definition 2.5).

Proof. Lemma 4.4 proves t_s-validity. Lemmas 4.5 and 4.6 together prove t_a-liveness and t_a-consistency, and Lemma 4.7 proves t_a-set quality. □

4.1.3 A Terminating ACS Protocol

For our eventual atomic broadcast protocol, we will need an ACS protocol with guaranteed termination. Fortunately, the non-terminating protocol in Section 4.1.2 can be transformed into a terminating protocol with only a few modifications.

Our terminating ACS protocol, $\mathsf{ACS}_{\text{tdg}}^{t_a,t_s}$, is described in Figure 4.3. The protocol consists of two phases. In the first phase, the parties simply run the non-terminating ACS protocol $\mathsf{ACS}_{\text{tdg}}^{*}$. In the second phase, parties send commit messages containing the output of the first phase and a threshold signature. Upon receiving a full signature (or enough signature shares to form a full signature), parties can safely abandon the non-terminating ACS protocol.

In the rest of this section, we abbreviate $\mathsf{ACS}_{\text{tdg}}^{t_a,t_s}$ and $\mathsf{ACS}_{\text{tdg}}^{*}$ as simply ACS and ACS^*, respectively.

$\mathsf{ACS}_{\textbf{tdg}}^{t_a,t_s}(v_j)$

- Run ACS^* using input v_j.
- Upon receiving output S_j from ACS^*, multicast $\langle \mathsf{commit}, S_j\rangle_j$.
- Upon receiving $t_s + 1$ signature shares on (commit, S), form a signature σ on (commit, S), multicast $(\mathsf{commit}, S, \sigma)$, output S, and terminate.
- Upon receiving a valid signature σ on (commit, S), multicast $(\mathsf{commit}, S, \sigma)$, output S, and terminate.

Figure 4.3: A terminating ACS protocol, from the perspective of party P_j with input v_j.

Lemma 4.8. $\mathsf{ACS}_{\text{tdg}}^{t_a,t_s}$ is t_a-terminating (as defined in Definition 2.5).

Proof. If one honest party terminates ACS then all honest parties will eventually receive a valid signature and thus terminate ACS. While no honest party has yet terminated, t_a-liveness of ACS* implies that all honest parties will eventually generate output from ACS*; moreover, t_a-consistency of ACS* implies that all those outputs will be equal to the same set S. So the $n - t_a \geq t_s + 1$ honest parties will send signature shares on S to all parties, which means that all honest parties will terminate. □

Lemma 4.9. Fix t_a, t_s with $t_a \leq t_s$ and $t_a + 2 \cdot t_s < n$. Then ACS is t_a-secure, t_a-terminating, and t_s-valid with termination, and has t_a-set quality (as defined in Definition 2.5).

Proof. Lemma 4.8 implies that ACS is t_a-live as well as t_a-terminating. If an honest party outputs a set S from ACS, then (as long as at most t_s parties are corrupted) at least one honest party must have output S from ACS*. Thus, ACS inherits t_a-set quality, t_a-consistency, and t_s-validity (without termination) from ACS* (cf. Theorem 4.1). It is straightforward to extend t_s-validity to t_s-validity with termination using an identical argument as in Lemma 4.8. □

4.1.4 Communication Complexity of ACS

Let $|v|$ be the size of each party's input. Recall that each instance of BB has communication complexity $O(n^2 |v|)$, and each instance of BA has cost $O(n^2)$. Since the inner protocol ACS* consists of n parallel instances of BB and BA, the cost of the inner protocol is $O(n^3 |v|)$. In the remaining steps, each party sends a set of size at most n plus a signature share (or signature) to everyone else, contributing an additional $O(n^2 \cdot (n |v| + \kappa))$ communication. The total communication for ACS is thus $O(n^3 |v| + n^2 \kappa)$.

4.2 A Block Agreement Subprotocol

Here, we present a *block agreement* protocol $\mathsf{BLA}_{\text{tdg}}^{t_a,t_s}$ (abbreviated BLA for the remainder of this section). Block agreement can be viewed as a network-agnostic version of validated multivalued Byzantine agreement, in which the goal is to agree on a shared output that satisfies an external validity property.

The inputs and outputs of the block agreement protocol are objects called *pre-blocks*. Within TARDIGRADE, pre-blocks act as a midway point between raw inputs and finished blocks. Formally, recall from Definition 2.6 that a pre-block is a vector of length n, where the ith entry is either $\perp$ or a message along with a valid signature by P_i on that message. The *quality* of a pre-block is defined as the number of entries that are not $\perp$; we say that a pre-block is a k*-quality pre-block* if it has quality at least k.

Later, in our atomic broadcast construction, we will use the fact that our BLA protocol terminates quickly if the network is synchronous. In the case where the network is asynchronous, we only require that any honest parties who generate output must agree.

The rest of this section is devoted to constructing a concrete protocol for block agreement, ultimately proving the following theorem:

Theorem 4.2. Fix a maximum input length $|v|$. There is a block-agreement protocol BLA (as defined in Definition 2.6) with communication complexity $O(n^3\kappa^2 + n^2\kappa|v|)$ that is t-secure for any $t < n/2$ when run in a synchronous network and terminates in time $5\kappa\Delta$.

Our block agreement protocol is modeled after the synod protocol of Abraham et al. [40]. In the rest of this section, we will work our way up to the complete block agreement protocol in several steps, starting with a simple multi-valued *value-proposal* protocol.

4.2.1 A Value-Proposal Subprotocol

Our first building block (see Figure 4.4) allows a designated party (called the proposer) to propose a pre-block. (For the purposes of this subprotocol, we can mostly ignore the semantics of pre-blocks and simply treat the inputs and outputs as abstract values.) During the proposal protocol, parties send votes to show their support for a proposed pre-block. Each vote carries a round number r, a pre-block β, and a set of signed messages C. A tuple (r, β, C) is called a round r-vote (or just an r-*vote*) for a pre-block β if it satisfies either of the following:

- $r = 0$ and $C = \emptyset$, or
- $r > 0$ and C is a set of at least $t + 1$ signed messages $\langle \mathsf{commit}, r_i, \beta \rangle_i$ from distinct parties such that $r_i \geq r$.

When the exact value of r is unimportant or clear from context, we simply refer to the tuple as a *vote*.

At the start of the propose protocol, the proposer (denoted P^*) waits to receive a set V of signed votes on valid pre-blocks such that $|V| \geq t + 1$. Then, the proposer determines a safe pre-block to propose from among these votes by finding a vote (r^*, β^*, C^*) in V such that r^* is greater than or equal to the round number of all other votes in V (breaking ties by lowest party index). The proposer then multicasts a *proposal* message $\langle \mathsf{propose}, (r^*, \beta^*, C^*), V \rangle_*$. An honest party who receives a proposal will consider it valid if all of the following hold:

- the signatures on the propose message and on each vote in V are valid,
- β^* is a valid pre-block,
- there is an r^*-vote for β^* in V,
- V contains at least $t + 1$ votes,

- r^* is greater than or equal to the round number of all votes in V.

If any of these conditions are not met, the proposal is not considered valid.

$\mathsf{Prop}^t_{\mathbf{tdg}}(r, \beta, C)$

1. (All parties) At time 0: send $\mathsf{vote}_i := \langle \mathsf{vote}, (r, \beta, C) \rangle_i$ to P^*.
2. (Only proposer) Until time Δ: Set $V = \emptyset$. Upon receiving a vote vote_j from party P_j on a valid pre-block: if this is the first such message received from P_j during this round, add vote_j to V.
3. (Only proposer) At time Δ, if $|V| \geq t$, find the vote (r^*, β^*, C^*) in V such that r^* is greater than or equal to the round number of all votes in V (breaking ties by lowest party index), and multicast $\langle \mathsf{propose}, (r^*, \beta^*, C^*), V \rangle_*$.
4. (All parties) At time 2Δ, if a valid $m = \langle \mathsf{propose}, (r^*, \beta^*, C^*), V \rangle_*$ has been received from P^*, multicast m. Otherwise, output $\perp$.
5. (All parties) At time 3Δ: let m_j denote the valid proposal forwarded by P_j (if any). If there exists m_j such that $m_j \neq m$, output $\perp$. Otherwise, output the pre-block β^* carried by the proposal m.

Figure 4.4: A protocol parameterized by threshold t and designated proposer P^*, from the perspective of party P_i.

In the rest of this section, to simplify the notation, we omit the superscript t from $\mathsf{Prop}^t_{\mathsf{tdg}}$.

We first show that any two honest parties who generate output in this protocol agree on their output.

Lemma 4.10. If honest parties P_i and P_j output $\beta_i, \beta_j \neq \perp$, respectively, in an execution of $\mathsf{Prop}_{\mathsf{tdg}}$, then $\beta_i = \beta_j$.

Proof. If P_i outputs $\beta_i \neq \perp$, then P_i must have received a valid proposal message for β_i by time 2Δ. That message is forwarded by P_i to P_j, and hence P_j either outputs $\perp$ (if the proposals do not match) or the same value β_i. $\square$

Next, we show that if each honest party P_i inputs a vote (r_i, β, C_i) on the same pre-block β, and no honest party ever receives a vote (r', β', C') such that $r' \geq \min_i\{r_i\}$ and $\beta' \neq \beta$, then any honest party who outputs a value other than $\perp$ outputs β.

Lemma 4.11. If each honest party P_i inputs an r_i-vote to $\mathsf{Prop}_{\text{tdg}}$ on the same valid pre-block β (possibly with different values r_i), and if no honest party ever receives a r'-vote on $\beta' \neq \beta$ with $r' \geq \min_i\{r_i\}$, then every honest party outputs either β or $\perp$.

Proof. Consider an honest party P who outputs $\beta \neq \perp$. That party must have received a valid proposal message from P^*, which in turn must contain a vote (r_i, β, C_i) from at least one honest party P_i. Under the assumptions of the lemma, any other vote (r', β', C') contained in the proposal message with $r' \geq r_i$ has $\beta' = \beta$. It follows that P outputs β. □

Finally, we show that when P^* is honest then all honest parties do indeed generate output.

Lemma 4.12. If each honest party P_i inputs a vote (r_i, β_i, C_i) on some valid pre-block β_i to $\mathsf{Prop}_{\text{tdg}}$, and P^* is honest, then there is some $(n-t)$-quality pre-block $\beta \neq \perp$ such that every honest party outputs β.

Proof. P^* will receive at least $t+1$ votes from honest parties, and so sends a valid proposal message on some $(n-t)$-quality pre-block β to all honest parties. (It is possible for the set V to include votes from dishonest parties, but these votes must be on $(n-t)$-quality pre-blocks.) Since P^* is honest, and the adversary cannot forge signatures on other proposals behalf of P^*, this is the only valid proposal message the honest parties will receive. Therefore, all honest parties output $\beta \neq \perp$. □

This concludes our analysis of $\mathsf{Prop}_{\text{tdg}}$.

4.2.2 A Graded Consensus Subprotocol

Next, we present a graded consensus protocol $\mathsf{GC}^t_{\text{tdg}}$ (abbrev. GC_{tdg}). This protocol builds on $\mathsf{Prop}_{\text{tdg}}$ to achieve a form of graded consensus on pre-blocks. The protocol's outputs are tuples (β, C, g), where β is a pre-block, C is a collection of signatures (defined more formally below), and g is a value in $\{0, 1, 2\}$ called the *grade*.

As in Abraham et al. [40], we assume an atomic leader-election mechanism $\mathsf{ElectLeader}$. Upon receiving input r from a majority of parties, $\mathsf{ElectLeader}$ chooses a uniform leader $P^* \in \{1, \ldots, n\}$ and sends (r, P^*) to all parties. (Note that if $t < n/2$, as is the case here, then at least one honest party must call $\mathsf{ElectLeader}$ with input r before the adversary can learn the identity of the leader.) A leader-election mechanism tolerating any $t < n/2$ faults can be realized (in the synchronous model with a PKI) based on general assumptions [47]; it can also be realized more efficiently using a threshold unique signature scheme.

Below, we refer to a message $\langle \mathsf{commit}, r, \beta \rangle_i$ as a *valid commit message* from P_i on a pre-block β if the quality of β is at least $(n-t)$ and the associated signature is valid. Commit messages are used to construct *notify messages* $(\mathsf{notify}, r, \beta, C)$. A notify message $(\mathsf{notify}, r, \beta, C)$ is *valid* if β is an $(n-t)$-quality pre-block and C is a set of valid commit messages such that (1) all commit messages carry the same pre-block β, (2) C contains messages from at least $t+1$ distinct parties, and (3) for each $c_i = \langle \mathsf{commit}, r_i, \beta \rangle_i \in C$ the round number r_i is greater than or equal to r.

Lemma 4.13. Assume that the input of each honest party P_i to GC_{tdg} is a vote on the same $(n-t)$-quality pre-block β. If no honest party ever receives an r'-vote on $\beta' \neq \beta$ such that r' is greater than or equal to the smallest round number carried by an honest parties' input in step 1 of GC_{tdg},

```
$\mathsf{GC}^t_{\mathsf{tdg}}(r, \beta, C)$

At time $0$: Set $C' = \emptyset$. Call $\mathsf{ElectLeader}(r)$ and let $(r, P^*)$ denote the output. Run $\mathsf{Prop}_{\mathsf{tdg}}$
        using input $(r, \beta, C)$.
At time $3\Delta$: Let $\beta^*$ denote the output of $\mathsf{Prop}_{\mathsf{tdg}}$. If $\beta^* \neq \perp$, multicast $\langle \mathsf{commit}, r, \beta^* \rangle_i$.
        Until time $4\Delta$, upon receiving a valid commit message $c_j = \langle \mathsf{commit}, r_j, \beta_j \rangle_j$ from
        $P_j$, if this is the first such message received from $P_j$, add $c_j$ to $C'$.
At time $4\Delta$: If there is a subset $C'' \subseteq C'$ of commit messages on the same pre-block $\beta'$ such
        that (a) $|C''| \geq t + 1$, and (b) for each $c_j = \langle \mathsf{commit}, r_j, \beta' \rangle_j \in C''$, $r_j \geq r$, then
        multicast $(\mathsf{notify}, r, \beta', C'')$, output $(\beta', C'', 2)$, and terminate.
At time $5\Delta$: If a valid notify message $(\mathsf{notify}, r, \beta^*, C^*)$ has been received, output $(\beta^*, C^*, 1)$
        and terminate (if there is more than one such message, choose arbitrarily). Otherwise,
        output $(\perp, \perp, 0)$ and terminate.
```

Figure 4.5: A graded consensus protocol parameterized by threshold t from the perspective of party P_i.

then (1) no honest party sends a commit message on $\beta' \neq \beta$ and (2) any honest party who outputs a nonzero grade outputs β.

Proof. By Lemma 4.11, every honest party outputs either β or $\perp$ in every execution of Π_{Propose} in step 1. It follows that no honest party P_i sends a commit message on $\beta' \neq \beta$, proving the first part of the lemma. Since at most t parties are corrupted, this means an honest party will receive fewer than $t + 1$ valid commit messages on anything other than β; it follows that if an honest party outputs grade $g = 2$ then that party outputs $(\beta, C, 2)$.

Arguing similarly, no honest party will receive a valid notify message on anything other than β. Hence each honest party that outputs grade 1 outputs $(\beta, C', 1)$. □

Lemma 4.14. If an honest party outputs (β, C, g) such that $g \neq 0$ in an execution of $\mathsf{GC}_{\mathsf{tdg}}$, then no honest party sends a commit message on $\beta' \neq \beta$.

Proof. Say an honest party outputs (β, C, g) where g is nonzero. That party must have received a valid notify message on β. Therefore, C must contain signatures from at least $t + 1$ distinct parties. It follows that at least one honest party P must have sent a commit message on β. This means that P must have received β as its output from $\Pi^{P^*}_{\mathsf{Propose}}$. By Lemma 4.10, this means the pre-block output by any other honest party from $\Pi^{P^*}_{\mathsf{Propose}}$ is either β or $\bot$. □

Lemma 4.15. If some honest party outputs (β, C, g) with grade $g = 2$ in an execution of GC_{tdg}, then each honest party P_i outputs (β_i, C_i, g_i) such that $\beta_i = \beta$ and $g > 0$.

Proof. Say an honest party P outputs (β, C, g) such that $g = 2$. By Lemma 4.14, this means no honest party sent a commit message on $\beta' \neq \beta$; it is thus impossible for any honest party to output $\beta' \neq \beta$ with a nonzero grade. Since P sends a valid notify message on β to all honest parties before terminating, every honest party will output β with a nonzero grade. □

Lemma 4.16. During an execution of GC_{tdg}, the event that every honest party outputs the same $(n-t)$-quality pre-block β with a grade of 2 occurs with probability at least $1/2$.

Proof. The leader P^* is honest with probability at least $\frac{n-t}{n} > 1/2$. If the leader is honest, agreement on an $(n-t)$-quality pre-block β follows from Lemma 4.15. Therefore, it remains to show that whenever the leader is honest, every honest party outputs grade 2.

Assume P^* is honest. Lemma 4.12 implies that every honest party receives the same pre-block $\beta \neq \bot$ as output from $\mathsf{Prop}_{\text{tdg}}$. Thus, every honest party sends a valid commit message on β by time 3Δ. Consequently, each honest party P_j receives $n-t$ commit messages on the same pre-block β before time 4Δ. This causes them to output with grade $g = 2$. □

In Figure 4.6 we describe the complete block-agreement protocol $\mathsf{BLA}^t_{\text{tdg}}$ (abbrev. $\mathsf{BLA}_{\text{tdg}}$). Note that parties do not terminate upon generating output; instead, parties terminate after participating in all κ rounds of the protocol.

Lemma 4.17. If $t < n/2$, then $\mathsf{BLA}_{\text{tdg}}$ is t-secure (as defined in Definition 2.6).

```
$\mathsf{BLA}^t_{\mathbf{tdg}}(\beta)$

Set $(\beta', C') = (\beta, \emptyset)$ and set $r = 0$.
While $r < \kappa$:
    · At time $5r \cdot \Delta$: run $\mathsf{GC}_{\mathsf{tdg}}$ using input $(r, \beta', C')$.
    · At time $5(r+1) \cdot \Delta$:
        – Let $(\beta^*, C^*, g)$ denote the output of $\mathsf{GC}_{\mathsf{tdg}}$.
        – If $g > 0$, set $(\beta', C') = (\beta^*, C^*)$.
        – If $g = 2$, output $\beta'$.
        – Set $r = r + 1$.
```

Figure 4.6: A block-agreement protocol with security parameter κ and threshold parameter t, from the perspective of party P_i.

Proof. Assume at most t parties are corrupted during an execution of BLA. Termination follows trivially from the protocol description, as parties terminate after κ fixed-length rounds.

Let r^* be the first round in which some honest party outputs a pre-block β. We first show that in every subsequent round, the following hold: (1) every honest party P_i uses as its input in step 1 an r_i-vote on β; and (2) the adversary cannot construct an r'-vote on $\beta' \neq \beta$ for any $r' \geq \min_i\{r_i\}$.

Consider some honest party P_i who outputs a pre-block β in round r^*. P_i must have received a valid notify message for β during the graded consensus subprotocol for that round. By Lemma 4.15, this means every honest party received a valid notify message for β in the same execution of $\mathsf{GC}_{\mathsf{tdg}}$, and so claim (1) holds in round $r^* + 1$. Moreover, Lemma 4.14 implies that no honest party sent a commit message on $\beta' \neq \beta$ in the execution of $\mathsf{GC}_{\mathsf{tdg}}$, and so claim (2) also holds in round $r^* + 1$. Lemma 4.13 implies, inductively, that the two claims will continue to hold in every subsequent round. Thus, any other honest party P_j who generates output in BLA also outputs β, regardless of whether it outputs in round r^* or a later round. This proves t-consistency.

Lemma 4.16 shows that in each round of BLA, with probability at least $1/2$ there is an $(n-t)$-quality pre-block β such that all honest parties output β in that round. Thus, after κ rounds all honest parties have generated $(n-t)$-quality output except with negligible probability. This proves t-validity. $\square$

4.2.3 Communication Complexity of Block Agreement

Within the propose subprotocol, parties send and receive votes. Each vote is a tuple (r, β, C) where r is an integer round number, β is a pre-block, and C is a set of $O(n)$ signatures σ_i on $(\mathsf{commit}, r_i, \beta)$. (Because r_i is not necessarily equal to r_j for all σ_i, σ_j in C, the signatures cannot be combined into a single threshold signature.) Thus, the total size of each vote is $O(n\kappa + |m|)$, where $|m|$ denotes the size of a pre-block. The most expensive step of the propose subprotocol requires all parties to send a vote to all other parties, resulting in an overall communication complexity for the propose subprotocol of $O(n^2(n\kappa + |m|)) = O(n^3\kappa + n^2|m|)$.

In the graded consensus subprotocol, the parties participate in one run of the propose subprotocol and send a constant number of all-to-all messages of size $O(n\kappa + |m|)$. Since both of these steps cost $O(n^3\kappa + n^2|m|)$, the overall communication complexity for one instance of graded consensus is the same as that of the propose protocol.

$\mathsf{BLA}_{\text{tdg}}$ runs κ rounds of the graded consensus protocol, for a total communication cost of $O(\kappa \cdot (n^3\kappa + n^2|m|)) = O(n^3\kappa^2 + n^2\kappa \cdot |m|)$.

4.3 A Network-Agnostic Atomic Broadcast Protocol

In this section, we use the subprotocols we have built so far to construct TARDIGRADE (abbrv. TDG), an atomic broadcast protocol that is t_s-secure in a synchronous network and t_a-secure in an asynchronous network for any t_a, t_s satisfying condition $\bigstar$. The complete pseudocode appears in Figure 4.7.

The protocol proceeds in a sequence of logical intervals called epochs. In each epoch, the parties agree on a single set of values called a block. We represent the sequence of blocks output by each party P_i as a write-once array Blocks_i.

During each epoch, the parties begin by running the block agreement protocol BLA. If they do not receive output from BLA within a fixed time limit, they abandon BLA and use ACS to agree on a block instead. On the other hand, if a party does receive output from BLA, it uses that output as its input to ACS.

TARDIGRADE's network-agnostic properties arise from the complementary properties of its subprotocols. If the network is synchronous, then BLA ensures that all honest parties input the same value B to the ACS protocol. This is exactly why ACS needs to have t_s-validity with termination: so that, in this case, all parties agree on the singleton set $\{B\}$ after running ACS. On the other hand, if the network is asynchronous and at most t_a parties are corrupted, it is possible that BLA will not succeed, and parties may input different values to ACS. However, in this case t_a-security of ACS ensures that the parties will agree on a set of values $B = \{B_1, B_2, \ldots\}$. Moreover, the output-quality property ensures that at least a constant fraction of the values in B were contributed by honest parties; this will be important for guaranteeing liveness of the full atomic broadcast protocol.

4.3.1 Technical Overview

Each epoch of TARDIGRADE proceeds in four basic steps. First, there is an information-gathering step in which each party sends its encrypted input to all other parties, and waits for a fixed amount of time to receive inputs from others. In the second phase, any party who received enough inputs during the first phase will use them as input to the synchronous block agreement protocol BLA. If the network is synchronous and at most t_s parties are corrupted, BLA is guaranteed to output a set of inputs that contains sufficiently many honest parties' inputs. BLA is run for a fixed amount of time, with the timeout chosen to ensure that (with high probability) it will terminate before the timeout if the network is synchronous. This brings us to the third step, in which parties run the ACS protocol. If a party received a pre-block as output from BLA before the timeout, it passes that pre-block as input to ACS; otherwise, it assembles inputs from other parties into a pre-block and inputs that instead. In the fourth and final step, the parties participate in threshold decryption to recover the plaintext transactions, and then output a block containing those transactions.

4.3.2 Technical Details

At the start of each epoch, each party P_i chooses a set V_i of L/n transactions from among the first L transactions in its local buffer, where L is a parameter corresponding to block size. (For now, we leave L as a variable; later, in the communication complexity analysis, we will discuss how L is actually set.)[4] Next, P_i encrypts V_i using a (t_s, n)-threshold encryption scheme to give

[4]We assume without loss of generality that parties always have at least L transactions in their buffer, since they can always pad their buffers with null transactions.

TARDIGRADE

For each iteration $k = 1, 2, \ldots$ do:

- At time $T_k = \lambda \cdot (k-1)$: sample $V \leftarrow \mathsf{ProposeTxs}(L/n, L)$ and encrypt V using pk to produce a ciphertext μ. Multicast $(\mathsf{input}, \langle\mu\rangle_j)$.
- Upon receiving a signed input $(\mathsf{input}, \langle\mu\rangle_i)$ from P_i (for iteration k):
  - If this is the first input received for iteration k, create a new pre-block $\beta_j^k := (\perp, \ldots, \perp)$ and set $\mathsf{ready}_k := \mathsf{false}$.
  - If $\beta_j^k[i] = \perp$: set $\beta_j^k[i] := \langle\mu\rangle_j$.
  - If β_j^k is an $(n-t_s)$-quality pre-block and $\mathsf{ready}_k = \mathsf{false}$, set $\mathsf{ready}_k := \mathsf{true}$.
- At time $T_k + \Delta$: if $\mathsf{ready}_k = \mathsf{true}$, run BLA using input β_j^k.
- At time $T_k + \Delta + 5\kappa\Delta$:
  - Terminate BLA (if it has not already terminated). If BLA had output an $(n-t_s)$-quality pre-block β^*, run ACS using input β^*. Else, wait until $\mathsf{ready}_k = \mathsf{true}$ and then run ACS using input β_j^k.
  - When ACS terminates, if the output set contains any valid pre-blocks, input the unique pre-blocks to ConstructBlock to produce a block B. Then set $\mathsf{blocks}[k] := B$ and delete from buf_j any transactions that appear in $\mathsf{blocks}[k]$.

$\mathsf{ProposeTxs}(\ell, M)$: choose a set V of ℓ values $\{\mathsf{tx}_1, \ldots, \mathsf{tx}_\ell\}$ uniformly (without replacement) from the first M values in buf_j, then output V.

$\mathsf{ConstructBlock}(B^*)$: B^* is assumed to be a set of valid pre-blocks. For each unique ciphertext in each pre-block $\beta \in B^*$, participate in threshold decryption. Wait for all decryptions to complete, then output the set B of all unique transactions in the obtained plaintexts.

Figure 4.7: Atomic broadcast protocol TARDIGRADE, from the perspective of party P_j.

a ciphertext μ_i.[5] Each party signs its ciphertext and multicasts it, then waits for a fixed period of time to receive signed ciphertexts from the other parties. Whenever a party receives a signed ciphertext during this time, it adds that signed ciphertext to its pre-block. Any party who forms an $(n-t_s)$-quality pre-block in this way within the time limit will input that pre-block to BLA. Next, the parties run BLA until it terminates or times out. If a party receives an $(n-t_s)$-quality pre-

[5]As in HoneyBadger [6], transactions are encrypted to limit the adversary's ability to selectively censor certain transactions.

block as output from BLA before the timeout, it inputs that pre-block to the ACS protocol ACS*. Otherwise, it abandons BLA and inputs the $(n - t_s)$-quality pre-block it collected earlier (waiting for additional signed ciphertexts to arrive if necessary). Eventually, ACS* outputs a set of pre-blocks, at which point a helper function ConstructBlock is used to decrypt the accompanying ciphertexts and form the final block.

Each party begins epoch k when its local clock reaches time $T_k := \lambda \cdot (k - 1)$, where $\lambda > 0$ is a *spacing parameter*. (The value of λ is irrelevant for the security proofs, but can be tuned to achieve better performance in practice; see further discussion in Section 4.3.3.) If the network is synchronous, parties' clocks are synchronized and so all parties begin each epoch at the same time. If the network is asynchronous, we do not have this guarantee. In either case, parties do not necessarily finish agreeing on block k prior to starting epoch $k' > k$, and so it is possible for parties to be participating in several epochs in parallel.

Theorem 4.3 (Completeness and consistency of TDG)**.** Fix t_a, t_s satisfying condition ★. Then TDG is t_a-complete and -consistent when run in an asynchronous network, and t_s-complete and -consistent when run in a synchronous network (as defined in Definition 2.7).

Proof. First, consider the case where at most t_s parties are corrupted and the network is synchronous. In the beginning of each epoch k, each honest party multicasts a set of transactions, and so every honest party can form an $(n - t_s)$-quality pre-block by time $T_k + \Delta$. Thus, every honest party starts running BLA at time $T_k + \Delta$ using an $(n - t_s)$-quality pre-block as input. By t_s-security of BLA in a synchronous network (note $t_s < n/2$), with overwhelming probability every honest party outputs the same $(n - t_s)$-quality pre-block β^* from BLA by time $T_k + \Delta + 5\kappa\Delta$. Therefore, each honest party inputs β^* to ACS. By t_s-validity with termination of ACS, every honest party obtains the same set of values as output from ACS. So all honest parties eventually receive $n - t_s > t_s$ valid decryption shares for each ciphertext in each valid pre-block contained in the output of ACS, and so they all output the same block.

The case where the network is asynchronous and at most t_a parties are corrupted is similar. In each epoch, each honest party multicasts a set of transactions and so every honest party eventually receives input from at least $n - t_a \geq n - t_s$ distinct parties and can form an $(n - t_s)$-quality pre-block. This means that every honest party eventually runs ACS using an $(n - t_s)$-quality pre-block as input. By t_a-security of ACS, all honest parties eventually receive the same output B^* from ACS. So all honest parties will eventually receive $n - t_a > t_s$ valid decryption shares for each ciphertext in each pre-block of B^*, and they all output the same block. □

We now turn our attention to liveness. We begin by proving a bound on the number of honest parties who contribute transactions to a block. Formally, we say that an honest party P_i *contributes transactions to a block* in some execution if an honest party calls $\mathsf{ConstructBlock}$ on a set of pre-blocks B^* that contains a pre-block β such that $\beta[i] \neq \bot$. Using this, we show that any transaction that is at the front of most honest parties' buffers will eventually be output with overwhelming probability. Liveness follows by arguing that any transaction that is in the buffer of all honest parties will eventually move to the front of most honest parties' buffers.

Lemma 4.18. Fix t_a, t_s satisfying condition $\bigstar$, and assume at most t_a parties are corrupted and the network is asynchronous, or at most t_s parties are corrupted and the network is synchronous. Then in an execution of TDG, for any block B output by an honest party, at least $n - (t_s + t_a)$ honest parties contributed transactions to B.

Proof. First, consider the case where at most t_a parties are corrupted and the network is asynchronous. As in the proof of Theorem 4.3, every honest party executes ACS using an $(n - t_s)$-quality pre-block as input. Thus, the input of every honest party to ACS contains at least $n - (t_s + t_a)$ ciphertexts created by honest parties. By t_a-set quality of ACS, the output of ACS contains some honest party's input and the lemma follows.

Next, consider the case where at most t_s parties are corrupted and the network is synchronous. As shown in the proof of Theorem 4.3, every honest party executes ACS using the same $(n - t_s)$-quality pre-block β as input. By t_s-validity with termination of ACS, all honest parties output $B^* = \{\beta\}$ from ACS. Because β is $(n - t_s)$-quality, it contains at least $(n - 2t_s)$ honest parties' ciphertexts; the lemma follows. □

Lemma 4.19. Assume the conditions of Lemma 4.18. Consider an epoch k and a transaction tx such that, at the beginning of epoch k, all but at most t_s honest parties have tx among the first L transactions in their buffers. Then for any $r > 0$, tx is in $\mathsf{blocks}[k : k + r]$ except with probability at most $(1 - 1/n)^{r+1}$.

Proof. By Lemma 4.18, at least $n - (t_s + t_a)$ honest parties contribute transactions to $\mathsf{blocks}[k]$. So even if t_s parties are corrupted, at least one of the $n - 2t_s$ honest parties who have tx among the first L transactions in their buffers contributes transactions to $\mathsf{blocks}[k]$. That party fails to include tx in the set V of transactions it chooses with probability $\binom{L-1}{L/n}/\binom{L}{L/n} = 1 - \frac{1}{n}$, and so tx is in $\mathsf{blocks}[k]$ except with probability at most $1 - \frac{1}{n}$. (Note that this does not take into account the fact that the adversary may be able to choose which honest parties contribute transactions to B. However, because the parties encrypt their transactions, the adversary's choice has no effect on the calculation.) If tx does not appear in $\mathsf{blocks}[k]$, then we can repeat the argument in all successive epochs $k + 1, \ldots, k + r$ until it does. □

Theorem 4.4 (Liveness)**.** Fix $t_a \leq t_s$ with $t_a + 2 \cdot t_s < n$. Then TDG is t_a-live in an asynchronous network, and t_s-live in a synchronous network (as defined in Definition 2.7).

Proof. Suppose all honest parties have received a transaction tx. If, at any point afterward, tx is not in some honest party's buffer then tx must have already been included in a block output by that party (and that block will eventually be output by all honest parties). If all honest parties have tx in their buffers, then they each have a finite number of transactions ahead of tx. By completeness, all honest parties eventually output a block in each epoch. Additionally, by Lemma 4.18, at least $n - (t_s + t_a)$ honest parties' inputs are incorporated into each block, and so in each epoch all but at most t_s honest parties each remove at least L/n transactions from their buffers. It follows that eventually all but at most t_s honest parties will have tx among the first L transactions in their buffers. Once that occurs, Lemma 4.19 implies that tx is included in the next κ blocks except with probability negligible in κ. □

4.3.3 Efficiency and Choice of Parameters

The communication cost per epoch is dominated by the cost of the ACS and BLA subprotocols. Both ACS and BLA are run on pre-blocks, which have size $L \cdot |\mathsf{tx}| + O(n \cdot \kappa)$. Thus, each execution of BLA incurs cost $O(n^3\kappa^2 + n^2L|\mathsf{tx}|\kappa)$, and an execution of ACS incurs cost $O(n^4\kappa + n^3L|\mathsf{tx}|)$. The overall communication per block is therefore $O(n^4\kappa + n^3\kappa^2 + n^3L|\mathsf{tx}| + n^2L|\mathsf{tx}|\kappa)$.

At the beginning of every epoch, each honest party uniformly selects L/n transactions from among the first L transactions in its buffer. The following lemma shows that the expected number of *distinct* transactions they collectively choose is $O(L)$.

Lemma 4.20. Assume the conditions of Lemma 4.18. In any epoch of TDG, the expected number of distinct transactions contributed by honest parties to the block B output by the honest parties in that epoch is at least $L/4$.

Proof. The expectation is minimized when all honest parties have the same L transactions as the first L transactions in their buffers, so we assume this to be the case. As in Lemma 4.19, for some particular such transaction tx, the probability that some particular honest party fails to include tx in the set V of transactions it chooses is $1-\frac{1}{n}$. Since, by Lemma 4.18, at least $n-(t_s+t_a) > n/3$ honest parties contribute transactions to B, the probability that none of those parties choose tx is at most $\left(1-\frac{1}{n}\right)^{n/3} \leq e^{-1/3} < 3/4$, and so tx is chosen by at least one of those parties with probability at least $1/4$. (Once again, we do not take into account the fact that the adversary may be able to choose which honest parties contribute transactions, because all transactions remain encrypted until after the adversary makes their decision.) The lemma follows by linearity of expectation. $\square$

Because each block contains $O(L)$ transactions, the communication cost per transaction is $O((n^4\kappa + n^3\kappa^2)/L + n^3|\mathsf{tx}| + n^2|\mathsf{tx}|\kappa)$. So for $L = \Theta(n\kappa)$, the amortized communication cost per transaction is $O(n^3|\mathsf{tx}| + n^2|\mathsf{tx}|\kappa)$.

We remark that although each block contains at least $L/4$ distinct transactions in expectation, it is possible that some of those transactions are not new, i.e., they may have already been included in a previous block. This is possible because an honest party may sample its input in some epoch before it has finished outputting blocks in all previous epochs. Thus, the communication cost per transaction computed above may be an underestimate.

While repeated transactions have no effect on consistency,[6] they do negatively affect live-

[6]Our definition of atomic broadcast is only concerned with agreeing on an order over abstract values; the question of how to handle repeated transactions is left to the application level.

ness. In particular, the number of redundant transactions is related to the spacing parameter λ as well as the actual network conditions and the parties' local clocks. For example, if λ is too small, it is possible for honest parties to input similar sets of transactions to many consecutive blocks, which results in wasted block space; however, setting λ to be too large could introduce unnecessary delays in the synchronous case. In the rest of this section, we derive a bound on the rate at which parties clear transactions from their buffers in terms of λ, actual message delays, and the speed of parties' local clocks.

In what follows, it is useful to measure time by a "global clock," even though honest parties' clocks are not guaranteed to be synchronized with each other or with this global clock. More formally, imagine an external observer with a clock running at some fixed rate ρ. (The observer's clock is not visible to the honest parties and is not assumed to be synchronized with parties' local clocks.) Let ρ_i denote the (possibly variable) rate at which P_i's local clock runs relative to the observer's clock.

Fix some finite interval $I = [\mathsf{Start}, \mathsf{End}]$ during an execution of the protocol. From the perspective of the observer, it is possible to identify bounds $\rho_{min}(I)$, $\rho_{max}(I)$ on the skew of honest parties clocks during interval I, so that for all honest P_i, $\rho_{min} \leq \rho_i \leq \rho_{max}$. The observer can also determine an upper bound $\delta(I)$ such that any message sent by time $T \in [\mathsf{Start}, \mathsf{End} - \delta]$ is delivered by time $T + \delta$. (Note that in an asynchronous network, $\delta(I)$ may be significantly greater than Δ.) Lastly, we let β_{max} denote the maximum number of transactions in any honest party's buffer during the given interval. We emphasize that ρ_{min}, ρ_{max}, δ, and β_{max} do not need to be known by the honest parties, and are used only for the analysis.

For each i and each k, let $\mathsf{Start}_{i,k}$ and $\mathsf{End}_{i,k}$ be the time according to the observer's clock when P_i begins epoch k and when P_i outputs block k, respectively, and let $I_{i,k}$ denote the interval

$[\mathsf{Start}_{i,k}, \mathsf{End}_{i,k}]$. (By completeness of the protocol, $\mathsf{End}_{i,k}$ is well-defined for all i and k.)

The claims below apply in either setting (t_a corruptions in an asynchronous network, or t_s corruptions in a synchronous network); however, in a synchronous network the bounds are naturally simpler because we have $\rho_{min} = \rho_{max} = \rho$ and $\delta = \Delta$.

Lemma 4.21. For any epoch k and party P_i, the number of concurrent epochs that P_i begins running during the interval $I_{i,k} := [\mathsf{Start}_{i,k}, \mathsf{End}_{i,k}]$ is at most $\tau := \frac{\rho_{max}}{\lambda} \cdot \left(\frac{5\kappa\Delta+\Delta+A(\delta,\kappa)}{\rho_{min}}\right)$ (with overwhelming probability), where ρ_{min}, ρ_{max}, δ, β_{max}, and τ are measured by an external observer over the interval $I_{i,k}$, and $A(\delta, \kappa)$ is an upper bound such that the local running time of ACS for P_i during the interval $I_{i,k}$ is at most $A(\delta, \kappa)$, with overwhelming probability in κ.

Proof. Let ρ_i be the rate of P_i's local clock (or an upper bound on the rate, if it is variable). Each honest party P_i begins a new block every λ clock ticks, as measured by their local clock. Thus, the number of new blocks started by an honest party P_i during the interval $I_{i,k}$ is the length of $I_{i,k}$ (in global time) divided by λ/ρ_i.

We would like to find an upper bound on the length of $I_{i,k}$ for all honest P_i. The most significant contributors to the length of $I_{i,k}$ are the running time of BLA and ACS. The local running time of BLA is at most $5\kappa\Delta + \Delta$ for any honest party, because P_i will timeout at this time if BLA has not yet output. Thus, the running time of BLA for P_i according to the observer's clock is at most $\frac{5\kappa\Delta+\Delta}{\rho_i}$. By assumption, $\rho_{min} \leq \rho_i$ for all P_i, and so the global running time of BLA for any honest party is at most $\frac{5\kappa\Delta+\Delta}{\rho_{min}}$. Similarly, the running time of ACS from the observer's perspective is bounded above by $\frac{A(\delta,\kappa)}{\rho_{min}}$ (with overwhelming probability in κ).

We can simply add the bounds for BLA and ACS together to get an upper bound on the entire length of the interval $I_{i,k}$. Plugging this bound into the expression we had originally, we have the following bound on the number of new blocks started by any P_i during the interval $I_{i,k}$, which holds with overwhelming probability in κ:

$$\frac{|I_{i,k}|}{\lambda/\rho_i} \leq \frac{\frac{5\kappa\Delta+\Delta}{\rho_{min}} + A(\delta,\kappa)}{\frac{\lambda}{\rho_{max}}} = \frac{\rho_{max}}{\lambda} \cdot \left(\frac{5\kappa\Delta + \Delta + A(\delta,\kappa)}{\rho_{min}}\right) \tag{4.2}$$

This completes the proof. □

The following lemma concerns the overall progress of the honest parties.

Lemma 4.22. Let t denote the number of dishonest parties during an execution of TDG, and let tx be a transaction that has been received by each honest P_i by time $\mathsf{Start}_{i,k}$. Moreover, let ρ_{min}, ρ_{max}, δ, β_{max}, and τ be bounds as described above over the interval $I^{H}_{k,k+c_x\cdot\tau} := [\min_{P_i\in H}(\mathsf{Start}_{i,k}), \max_{P_j\in H}(\mathsf{End}_{j,k+c})]$. Then with overwhelming probability in the security parameter κ, there are at least $n-t$ honest parties P_i such that P_i removes at least β_{max} transactions from their buffer during the interval $[\mathsf{Start}_{i,k}, \mathsf{Start}_{i,k+c_x\cdot\tau}]$, where $c_x := \frac{\beta_{max}}{L/n} \cdot \frac{n-t}{n-(t_s+t_a)}$.

Proof. By Lemma 4.21, we know that every honest party P_i has output block k by time $\mathsf{Start}_{i,k+\tau}$. Therefore, by time $\mathsf{Start}_{i,k+\tau}$, P_i has removed from their buffer any transactions that are included in $\mathsf{blocks}[k]$. In particular, if P_i's input was included in block k, then P_i must have removed at least L/n transactions from the front of their buffer between time $\mathsf{Start}_{i,k}$ and $\mathsf{Start}_{i,k+\tau}$.

Next, we can extend this argument to apply to sets of honest parties. Recall from Lemma 4.18 that at least $n-(t_s+t_a)$ honest parties' inputs are included in each block. Let $S^*_{k+c\cdot\tau}$ denote the set of honest parties whose inputs are included in block $k+c\cdot\tau$ ($c=0,1,2,\ldots$). For each $P_i \in S^*_{k+c\cdot\tau}$, notice that P_i must have selected L/n transactions from their buffer as input at time $\mathsf{Start}_{i,k+c\cdot\tau}$, and those transactions were included in block $k+c\cdot\tau$. Therefore, P_i

must have removed at least L/n transactions from their buffer at some point during the interval $[\mathsf{Start}_{i,k+c\cdot\tau}, \mathsf{Start}_{i,k+(c+1)\tau}]$.

Consider a sequence of sets S^*_k, $S^*_{k+\tau}$, $S^*_{k+2\tau}, \dots$, defined as above. Suppose the adversary is able to choose S^* in each epoch, subject to the constraint that each S^* must contain at least $n-(t_s+t_a)$ honest parties. We would like to find an upper bound on number of epochs needed to ensure that all but t_s of the honest parties have tx among the first L transactions in their buffer. For convenience, assume that each honest party initially has exactly β_{max} transactions in its buffer ahead of tx, and assume without loss of generality that parties $P_1, \dots, P_{n-t}$ are honest. In the worst case, the adversary chooses the honest parties for each set in the sequence in a round robin fashion, i.e.:

$$S^*_{k+c\cdot\tau} = \{P_i \mid 1 + c\cdot(n-(t_s+t_a)) \leq i \leq c + c\cdot(n-(t_s+t_a))\mathsf{mod}(n-t)\}. \tag{4.3}$$

Let $c_x := \frac{\beta_{max}}{L/n}\cdot\frac{n-t}{n-(t_s+t_a)}$, and consider the sequence of sets $S^*_k, \dots, S^*_{k+c_x\cdot\tau}$ determined according to the round robin strategy. All together, each honest party is in at least $\lfloor\frac{(n-(t_s+t_a))\cdot c_x}{n-t}\rfloor = \lfloor\frac{\beta_{max}}{L/n}\rfloor$ distinct sets in the sequence. Therefore, each honest party P_i has removed at least $\lfloor\frac{\beta_{max}}{L/n}\rfloor\cdot L/n = \beta_{max}$ transactions from their buffer during the interval $[\mathsf{Start}_{i,k}, \mathsf{Start}_{i,k+c_x\cdot\tau}]$. $\square$

The analysis above represents a rough worst-case analysis of liveness, in terms of clock (de-)synchronization and protocol parameters. In a deployment scenario, clock synchronization may be out of the developer's control; however, these results suggest that poor choices of protocol parameters can add unnecessary load to the network without benefitting liveness. Methods for experimentally determining good choices of parameters for real network conditions are an interesting direction for future work.

4.4 From Atomic Broadcast to State Machine Replication

To transform TARDIGRADE into a full state machine replication protocol, it suffices to modify it so that parties output a proof along with each block. We will use the folklore generic transformation from ABC to SMR using digital signatures: at the end of each epoch, parties simply use their threshold signing key to sign the block and epoch number. Once a party collects $t_s + 1$ partial signatures on (B, e) at the end of epoch e, it combines them into a single signature σ and writes (B, σ) to $\mathsf{blocks}_i[e]$. (Note that this change does not increase the overall communication complexity: the cost of sending a block and partial signature all-to-all is asymptotically lower than the cost of the rest of the protocol.)

Lemma 4.23. If TARDIGRADE is modified to output blocks of form (B, σ) as described above, then the resulting protocol is an SMR protocol that is t_a-secure when run in an asynchronous network, and t_s-secure when run in a synchronous network (as defined in Definition 2.8).

The consistency, liveness, and completeness properties follow straightforwardly from the security of the atomic broadcast protocol, so it only remains to sketch external validity. Fortunately, this is also straightforward: in each epoch e there can exist at most one block B for which there is a valid signature, because the original atomic broadcast protocol is consistent (so all honest parties sign the same block) and there are at most t_s corrupted parties (so the corrupted parties cannot create a proof on their own).

For the sake of simplicity, from this point on, we use TARDIGRADE to refer to the SMR protocol rather than the ABC protocol.

4.5 An Impossibility Result for Network-Agnostic SMR

In this section, we show that our protocol achieves the optimal tradeoff between the security thresholds t_a, t_s. We will prove the result directly, without invoking the corresponding result for Byzantine agreement (although the proof techniques are ultimately similar). An alternative would be to show a reduction from Byzantine agreement to state machine replication in the network-agnostic setting: given such a reduction, our earlier impossibility result for network-agnostic BA would immediately imply a matching impossibility result for network-agnostic SMR. This approach initially appears promising, because there is a folklore result constructing BA from SMR in a *synchronous* network; however, the usual reduction assumes that the SMR protocol has a specific liveness property that is not achieved by network-agnostic (or even asynchronous) protocols.

To better understand the issue, recall the folklore reduction for synchronous BA and SMR. Suppose we have a synchronous SMR protocol with the following property: any transaction received by all honest parties by time T_0 must be output by round $T_0 + T_{\text{conf}}$. (T_{conf} is sometimes called the *confirmation time*; see e.g. [48], Section 6.2.) This property is common among synchronous SMR protocols, and enables us to achieve BA as follows. At time 0, each party sends a vote consisting of its input and a signature on its input to all other parties. Then, at time Δ, parties input the votes they received to their buffers and run the SMR protocol. Finally, at time $\Delta + T_{\text{conf}}$, each party gathers all of the votes that were output by the SMR protocol and outputs the bit that received the majority of votes. (If there are any parties who signed votes on more than one bit, count only the first vote that was output by the SMR protocol.)

We can see that the reduction hinges on an upper bound on confirmation time and an upper bound on the time when all inputs have been received, and so it is not obvious how to adapt it to general network-agnostic protocols without introducing additional properties or assumptions. Fortunately, we can still prove the result directly. In fact, we will prove a stronger result: there is no SMR protocol that achieves both t_s-liveness in a synchronous network and t_a-consistency in an asynchronous network for any t_a, t_s, n with $t_a + 2t_s \geq n$.

Theorem 4.5. Fix t_a, t_s, n with $t_a + 2t_s \geq n$. If an n-party SMR protocol is t_s-live in a synchronous network, then it cannot also be t_a-consistent in an asynchronous network (as defined in Definition 2.8).

Proof. Assume $t_a + 2t_s = n$ and fix an SMR protocol Π. Partition the n parties into sets S_0, S_1, S_a where $|S_0| = |S_1| = t_s$ and $|S_a| = t_a$. Consider the following experiment:

- Choose uniform $m_0, m_1 \leftarrow \{0,1\}^\kappa$. At global time 0, parties in S_0 begin running Π holding only m_0 in their buffer, and parties in S_1 begin running Π holding only m_1 in their buffer.

- All communication between parties in S_0 and parties in S_1 is blocked. All other messages are delivered within time Δ.

- Create virtual copies of each party in S_a, call them S_a^0 and S_a^1. Parties in S_a^b begin running Π (at global time 0) with their buffers containing only m_b, and communicate only with each other and parties in S_b.

Compare this experiment to a hypothetical execution E_{sync} of Π in a synchronous network, in which parties in S_1 are corrupted and simply abort, and the remaining parties are honest and initially hold only (uniformly chosen) m_0 in their buffer. The views of parties $S_0 \cup S_a^0$ in the experiment are distributed identically to the views of the honest parties in E_{sync}. Thus, t_s-liveness of Π implies that in the experiment, all parties in S_0 include m_0 in some block. Moreover, since parties in S_0 never receive information about m_1, they include m_1 in any block with negligible probability. By a symmetric argument, in the experiment, all parties in S_1 include m_1 in some block, and include m_0 in any block with negligible probability.

Now, consider a hypothetical execution E_{async} of Π, this time in an asynchronous network. In this execution, parties in S_0 and S_1 are honest while parties in S_a are corrupted. The parties in S_0 and S_1 initially hold $m_0, m_1 \leftarrow \{0,1\}^\kappa$, respectively. The corrupted parties interact with parties in S_0 as if they are honest and have m_0 in their buffer, and interact with parties in S_1 as if they are honest and have m_1 in their buffer. Meanwhile, all communication between parties in S_0 and S_1 is delayed indefinitely. The views of the honest parties in this execution are distributed identically to the views of $S_0 \cup S_1$ in the above experiment, yet the conclusion of the preceding paragraph shows that t_a-consistency is violated with overwhelming probability. $\square$

Chapter 5: Improving the Security and Efficiency of Network-Agnostic SMR

In the previous chapter, we introduced TARDIGRADE, a proof-of-concept network-agnostic SMR protocol. In this chapter, we will build on that foundation to present two "next generation" network-agnostic SMR protocols, UPDATE and UPSTATE.

For the most part, UPDATE and UPSTATE follow the basic template used in TARDIGRADE. In particular, the major building blocks of BLA and ACS will remain the same; the primary differences lie in the design of those building blocks. Additionally, we will skip the intermediate step of constructing explicit ABC protocols, and instead construct the final SMR protocols directly.

5.1 UPDATE: Network-Agnostic SMR with Optimal Thresholds and $O(n^3)$ Communication Complexity

This section presents UPDATE, an adaptively secure network-agnostic SMR protocol. Compared to its predecessor, TARDIGRADE, UPDATE achieves the same security guarantees (network-agnostic security against an adaptive adversary, for optimal thresholds t_s, t_a) while achieving better communication complexity — UPDATE has a total cost of $O(\kappa n^3)$ communication per block and an amortized cost of $O(\kappa n^2)$ communication per transaction. This improvement is largely due to an improved ACS construction based on error-correcting codes. Our earlier construction for adaptively secure BLA can be used as-is; however, for consistency across protocols, $\mathsf{BLA}_{\text{upd}}$

is sometimes used as an alias.

In the rest of this section, we introduce the new ACS construction in Section 5.1.1 and then present the full protocol in Section 5.1.2.

5.1.1 ACS Using Error-Correcting Codes

The protocol proceeds as outlined in Figure 5.3. The high-level design is similar to ACS protocols discussed previously; the main difference is the more sophisticated mechanism for input encoding ($\mathsf{INDI}_{\text{upd}}$, Figure 5.1), which uses Reed-Solomon codes. There is also a reconstruction subprotocol (referred to as $\mathsf{RECON}_{\text{upd}}$, Figure 5.1) in which parties multicast signed vote messages for a particular party upon successfully reconstructing that party's input. Upon receiving t_s+1 votes for the same party P_j, parties multicast a commit message carrying this certificate and the combined signature.[1] At the end of the protocol, protocol $\mathsf{Term}_{\text{upd}}$ (Figure 5.2) assembles an output certificate that allows parties to output and terminate (OC 0), ensuring no honest parties are "left behind."

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

[1] We note that recently, Das et al. [49] proposed an asynchronous reliable broadcast protocol using error correcting codes (but without digital signatures) that is related to this step.

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.

$\mathsf{INDI}_{\mathbf{upd}}(x)$

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 $\varphi_{i,j} := \mathsf{TS.Sign}(\mathsf{PK}, \mathsf{sk}_i, (s_{i,j}, h_i))$ and send $v_{i,j} := (s_{i,j}, h_i, \varphi_{i,j})$ 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.

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

1. Parse $v_{j,k}$ as $(s_{j,k}, h_j, \varphi_{j,k})$, discarding any messages 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 $\perp$.

Figure 5.1: Input distribution and reconstruction from the perspective of party $P_{i \in \{1,\ldots,n\}}$.

Across the protocols, we use PK as the public keys output by $\mathsf{TS.KeyGen}$ and sk_i the secret key associated to P_i. For simplicity, in $\mathsf{ACS}_{\mathsf{upd}}$ 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. Throughout this section, we assume a binary BA protocol $\mathsf{BA}_{\mathsf{upd}}$ (or just BA, for short) 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 $O(n^2)$.

$\mathsf{Term}_{\mathbf{upd}}(x, h)$

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, combine 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.

Figure 5.2: Termination helper protocol from the perspective of party $P_{i\in\{1,\dots,n\}}$.

Lemma 5.1. 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 $\mathsf{ACS}_{\mathsf{upd}}$.

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.

$\mathsf{ACS}_{\textbf{upd}}^{t_a,t_s}(x)$

1. Run $\mathsf{INDI}_{\text{upd}}(x)$ 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 $\mathsf{RECON}_{\text{upd}}$. If $\mathsf{RECON}_{\text{upd}}$ outputs x_j, multicast $\mathsf{vote}_i := \langle \mathsf{vote}, \langle h_j\rangle_i, \varphi_{j,i}\rangle_i$.
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 S^* to be the set of indices of the BA instances that output 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 $\mathsf{Term}_{\text{upd}}$.

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

OC 3. Else if P_i has (i) $|S^*| \geq n - t_a$, (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 $j \in S^*$, then output $S = \bigcup_{j\in S^*} x_j$ and terminate.

Figure 5.3: An ACS protocol from the perspective of party $P_{i\in\{1,\dots,n\}}$.

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 $\mathsf{RECON}_{\text{upd}}$ 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 5.2. 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 instance. However, if the network is asynchronous equivocation is not necessarily detected. Nevertheless, as we see below, validity will still hold.

Lemma 5.3. $\mathsf{ACS}_{\mathsf{upd}}$ satisfies t_s-validity with termination (as defined in Definition 2.5).

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 $\mathsf{Term}_{\mathsf{upd}}$ (in OC 1). If at least $t_s + 1$ parties call $\mathsf{Term}_{\mathsf{upd}}$ 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 $\mathsf{Term}_{\mathsf{upd}}$ 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 $\mathsf{Term}_{\mathsf{upd}}$, 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 S^* corresponding to the same value. By assumption, the honest parties have the same input x, so this implies a strict majority of values S^* correspond to corrupted parties. However, this contradicts the assumption that only t_s parties are corrupted, because $\lfloor \frac{|S^*|}{2} \rfloor \geq t_s$. Therefore, no honest party outputs via OC 3 when all honest parties have the same input. □

Lemma 5.4. $\mathsf{ACS}_{\mathsf{upd}}$ satisfies t_a-set quality (as defined in Definition 2.5).

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 $\mathsf{RECON}_{\mathsf{upd}}$ 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 5.5. $\mathsf{ACS}_{\mathsf{upd}}$ is t_a-terminating (as defined in Definition 2.5).

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 5.2, 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 S^* of BA instances that output 1. Finally, by Lemma 5.1, all honest parties reconstruct the same inputs associated to 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 $S^* = \{i|\mathsf{BA}_i \text{ output } 1\}$ for some $|S^*| \geq n - t_a$. 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 $\{x_j\}_{j \in S^*}$ and receives the certificates needed to terminate, since P_i must have sent these certificates to all other parties during $\mathsf{ACS}_{\text{upd}}$.

□

Lemma 5.6. $\mathsf{ACS}_{\text{upd}}$ satisfies t_a-consistency (as defined in Definition 2.5).

Proof. Assume an honest party P_i has output S. By Lemma 5.5, 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 $\left\lfloor \frac{n-t_a}{2} \right\rfloor + 1$ BA instances must intersect at an honest party, and so $x = x'$. Finally, if both P and P' contributed shares via OC 2, then they agree on S^*, and once again $x = x'$.

Second, suppose towards a contradiction that P_j output $S = \cup_{j \in S^*} x_j$ 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 S^* that output 1. By Lemma 5.1, we have $S' = S$. □

5.1.2 UPDATE: Full Protocol

We are now ready to present the complete protocol, UPDATE. As in TARDIGRADE, there is a spacing parameter (denoted here by μ) 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.

Theorem 5.1. Under condition ★, UPDATE is (1) t_s-consistent and t_s-complete if the network

```
UPDATE

Step 1. Proposal selection.
1.1 At time $T_e = \mu(e-1)$: Set $B_i^e := (\perp, \dots, \perp)$ 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 $\mathsf{BLA}^e_{\text{upd}}$. If $\mathsf{BLA}^e_{\text{upd}}$ terminates,
    let $B^*$ be the output.
2.2 At time $T_e + (5R+1)\Delta$: Terminate $\mathsf{BLA}^e_{\text{upd}}$ if not already terminated.
2.3 Pass $B^*$ or wait until $\mathsf{ready}_e = \mathsf{true}$ and pass $B_i^e$ as input to $\mathsf{ACS}^e_{\text{upd}}$.
2.4 Receive $S = \{B_j^*\}_{j \in S^*}$, where $S^* \subset \{1, \dots, n\}$ from $\mathsf{ACS}^e_{\text{upd}}$.

Step 3. Output and public verification.
3.1 On input $S = \{B_j^*\}_{j \in S^*}$, for each $j \in S^*$, do:
    - Jointly decrypt the values in $S = \{x_j\}_{j \in S^*}$.
    - Create a block by sorting $\bigcup_{j \in S^*} x_j$ 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 $\pi$ as $\langle h \rangle$ and proof of correct decryption of $S$.
    - Remove the transactions in block from the buffer and output $(\mathsf{block}, \pi)$.
3.3 Update $e \leftarrow e + 1$.
```

Figure 5.4: An SMR protocol with adaptive security, parameterized by thresholds t_a, t_s.

is synchronous and (2) t_a-consistent and t_a-complete if the network is asynchronous (defining t-security as in Definition 2.8).

Proof. We start with (1). Say a honest party P output a valid block in epoch e. Then P must have generated output in $\mathsf{ACS}_{\text{upd}}$ in epoch e, call it B, and at least $t_s + 1$ decryption shares on

B were gathered. By t_s-validity with termination of $\mathsf{ACS}_{\text{upd}}$, all honest parties will output B if they started $\mathsf{ACS}_{\text{upd}}$ with a valid pre-block B, so to prove t_s-consistency of UPDATE, it remains to show that all honest parties input the same B to $\mathsf{ACS}_{\text{upd}}$. Since the network is synchronous, by time $T_e + \Delta$, all honest parties have managed to assemble a $(n - t_s)$-quality pre-block B_i^e and input it to $\mathsf{BLA}_{\text{upd}}$, which is t_s-terminating, t_s-valid and t_s-consistent, so it terminates by time $T_e + (5R + 1)\Delta$ with the same valid output B. Finally, t_s-completeness follows from t_s-validity with termination of $\mathsf{ACS}_{\text{upd}}$.

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

Theorem 5.2. Under condition ★, UPDATE is (1) t_s-externally valid if the network is synchronous and (2) t_a-externally valid if the network is asynchronous (defining t-external validity as in Definition 2.8).

Proof. By Theorem 5.1, all honest parties will output the same valid block, obtained by decrypting the output of $\mathsf{ACS}_{\text{upd}}$, which means that they have valid certificates of correct decryption. External validity follows from the fact that the adversary cannot generate invalid certificates because it controls fewer than $t_s + 1$ parties. $\square$

In preparation for the proof of liveness, we prove a general result about the number of honest parties whose inputs are included in each pre-block.

Lemma 5.7. Assume there are at most t_y corrupted parties. If there are $n - t_s - t_x$ honest entries in a pre-block output in any given epoch, then there exist at least

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

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

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

honest parties who output more than once over r epochs.

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

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

$$\begin{aligned} r(n - t_s - t_x) &\leq (n - t_y - M_\alpha + 1)\left(\alpha r\frac{n - t_s - t_x}{n - t_y} - 1\right) + r(M_\alpha - 1) \\ M_\alpha &\geq 1 + (n - t_s - t_x)\frac{1 - \alpha + \frac{n-t_y}{e(n-t_x-t_s)}}{1 - \alpha\frac{n-t_s-t_x}{n-t_y} + \frac{1}{r}}, \end{aligned} \tag{5.1}$$

which contradicts the assumption at the beginning of the proof.

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

Lemma 5.7 implies that at least $n - t_x - t_s$ honest parties output at least once over r epochs. Moreover, we always have at least one honest party consistently outputting over r epochs, since

$$\alpha \leq \min\left\{1, \frac{1 + (n - t_y - 1)/r(n - t_s - t_x)}{1 - 1/(n - t_y)}\right\} = 1, \tag{5.2}$$

as long as $n > \max\{t_s + t_x, t_y\}$, which always happens in our settings. (A similar argument follows for the committee-based protocols with $\hat{t}_s = (1 - \epsilon)t_s$ and $\hat{t}_a = (1 - \epsilon)t_a$ corruptions.) we are ready to prove the liveness property of UPSTATE.

Theorem 5.3. Under condition ★, UPDATE is (1) t_s-live if the network is synchronous and (2) t_a-live if the network is asynchronous (defining t-liveness as in Definition 2.8).

Proof. Assume all honest parties (at least $n - t_s$ in (1) and at least $n - t_a$ in (2)) have received a transaction tx. If by some epoch e, tx is not in an honest party's buffer anymore it means it was output in blocks$[e']$ for $e' < e$. Then, by consistency of UPDATE proven in Theorem 5.1, tx will not be in any honest party's buffer after epoch e'. Otherwise, suppose tx is still in an honest party P's buffer at epoch e. By completeness of UPDATE proven in Theorem 5.1, each party outputs a block in every epoch. This block is obtained by decrypting a pre-block of $(n - t_s)$-set quality to which at least $n - t_s - t_a$ honest parties contributed L/n transactions, by Lemma 5.8 proved below. Thus, each honest party that contributes removes in expectation at least L/n transactions from their buffer in each epoch. Assuming parties cannot receive an infinite amount of transactions in a finite number of epochs, there will be a finite number of transactions in P's buffer alongside tx. By the lower bound in Lemma 5.7, honest parties continue to clear transactions from their buffers so that eventually tx appears among the first L transactions of their buffers. Once this has occurred, the probability that tx fails to appear in the output block at the e'th epoch if at least one of the honest parties that contributes its input to the block has tx among the first L transactions of its buffer is at most $1-1/n$. Thus, a transaction tx is included in $\mathsf{blocks}[e : e+r]$ with probability at least $1-(1-1/n)^{r+1}$, which approaches 1 as r goes to infinity. □

Lemma 5.8. Under condition ★, at least $n-t_s-t_a$ honest parties have contributed transactions in any block output by a honest party in UPDATE.

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

In case the network is asynchronous and there are at most t_a corrupted parties, $n-t_s-t_a$ entries in the pre-block originate from honest parties. By t_a-set quality of $\mathsf{ACS}_{\mathsf{upd}}$ (Lemma 5.4), the output of $\mathsf{ACS}_{\mathsf{upd}}$ contains at least a pre-block of $(n-t_s)$-quality, therefore with $(n-t_s-t_a)$ honest entries.

In case the network is synchronous, each honest party has received all messages from all other honest parties upon reaching Step 2 of UPDATE, so the number of honest entries in their pre-blocks is at least $(n-t_s)$. Moreover, all honest parties complete $\mathsf{BLA}_{\mathsf{upd}}$ with the same output pre-block B containing $(n - t_s)$ honest entries. By the t_s-validity with termination of $\mathsf{ACS}_{\text{upd}}$ (Lemma 5.3), the output pre-block of $\mathsf{ACS}_{\text{upd}}$ is also B. $\square$

5.1.3 Communication Complexity of UPDATE

First, 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 BLA. If BLA outputs, it outputs a pre-block of size $O(L|\mathsf{tx}|)$. The input to 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 incurs $O(nL|\mathsf{tx}| + n^2\kappa)$ total communication. In Step 2, BLA incurs $O(\kappa n^3 + \kappa n^2 L|\mathsf{tx}|)$ total communication and ACS incurs $O(\kappa n^3 + n^2 L|\mathsf{tx}|)$ total communication. Finally, in step 3, the parties assemble an output block and then multicast the signatures of the hash of the block to construct a proof, incurring $O(\kappa n^2)$ communication.

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

5.2 UPSTATE: Network-Agnostic SMR with Almost-Optimal Thresholds and $O(n^2)$ Communication Complexity

This section presents UPSTATE. UPSTATE achieves better communication complexity than UPDATE, but there is a tradeoff — UPSTATE is only secure against a static adversary, and supports almost-optimal corruption thresholds $\hat{t}_s, \hat{t}_a$ instead of the optimal t_s, t_a.

UPSTATE and its building blocks take full advantage of the static setting by dividing work among small (constant-size) committees. In the rest of this section, we present the modified ACS and BLA subprotocols in Sections 5.2.1 and 5.2.2, and then present the full protocol and proofs in Section 5.2.3.

5.2.1 A Committee-Based ACS Protocol for UPSTATE

Pseudocode for the ACS protocol appears in Figure 5.6. At the beginning of the protocol, inputs of size ℓ are passed to the input selection procedure $\mathsf{INSE}_{\text{ups}}$ (Figure 5.5), which determines the *primary committee* $\mathcal{C}$. Next, each party multicasts the codewords they received from the members of the primary committee.

To further 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 according to the second mechanism described in Section 2.2.5.

Like in UPDATE, error correcting codes are used to split inputs into $b = \hat{t}_s$ blocks. As before, we require the error correcting code scheme to tolerate either $\hat{t}_s$ erasures or $\hat{t}_a$ errors and

$\hat{t}_s - \hat{t}_a$ erasures.

For simplicity, in $\mathsf{ACS}_{\text{ups}}$, 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.

$\mathsf{INSE}_{\mathbf{ups}}(x_i)$

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.

Figure 5.5: An input selection subprotocol (handling input encoding and primary committee election) for UPSTATE, from the perspective of party $P_{i\in\{1,\ldots,n\}}$.

Lemma 5.9. $\mathsf{ACS}_{\text{ups}}$ 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 (defining each property as in Definition 2.5).

Sketch of proof. We discuss the changes arising from the use of committees in the proofs for the properties of the $\mathsf{ACS}_{\text{ups}}$ protocol. The proof will then follow from the proofs of Lemmas 5.1– 5.6.

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

$\mathsf{ACS}^{t_a,t_s}_{\textbf{ups}}(x_i)$

1. Run $\mathsf{INSE}_{\text{ups}}(x_i)$ 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 $\mathsf{RECON}_{\text{upd}}$. If $\mathsf{RECON}_{\text{upd}}$ 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 S^* to be the set of indices of the BA instances that delivered 1.
7. Output according to step 7 in $\mathsf{ACS}_{\text{upd}}$, where the set in OC 1 has size $(1 - t_s/n)\kappa$ and the sets in OC 2 and OC 3 have size $(1 - t_a/n)\kappa$.

Figure 5.6: An ACS protocol for UPSTATE, parameterized by thresholds t_a and t_s, from the perspective of party $P_{i\in\{1,\ldots,n\}}$.

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

5.2.2 A Committee-Based BLA Protocol for UPSTATE

5.2.2.1 Overview

Next, we will transform our original BLA protocol into a *committee-based* BLA protocol $\mathsf{BLA}_{\text{ups}}^{t_a,t_s}$ (Figure 5.9). This variant of BLA achieves $\hat{t}_s$-termination, $\hat{t}_s$-validity and $\hat{t}_s$-consistency (except with negligible probability in the security parameter) in a network that is synchronous with up to $\hat{t}_s = (1-\epsilon)t_s$ static corruptions. (Throughout this section, unless otherwise noted, all claims and proofs assume a synchronous network with $\hat{t}_s$ static corruptions.)

The new version of the protocol is still based on subprotocols for value-proposal (Figure 5.7) and graded consensus (Figure 5.8), but each building block must be carefully modified to achieve the desired result. Because the high level analysis follows similarly to the analysis of the original BLA protocol, we sketch the main ideas of the proof rather than reproving security in detail, with a focus on the parts of the protocol that have changed.

As a caveat, in this section only, we use a slightly different notion of validity (Definition 5.1, below). This is to account for the fact that only committee members contribute to the pre-block.

Definition 5.1. A BLA protocol with committee parameter κ is *t-valid* if the following holds: if every honest party inputs an $(1 - t/n)\kappa$-quality pre-block, then every honest party outputs an $(1 - t/n)\kappa$-quality pre-block.

The key insight driving this variant of BLA is to elect a leader who proposes an input among the ones sent by the parties, such that honest parties will commit on the same value. Importantly, the proposal of inputs is performed before the leader election.[2] Due to the forward

[2]To the best of our knowledge, the technique of electing leaders after proposals as a defense against adaptive adversaries was popularized by Abraham et al. [50].

secure signatures, the adversary cannot later corrupt the leader and cause them to equivocate.

The value proposal protocol itself remains largely the same; one difference to note is the slightly more sophisticated encoding. Within each execution of $\mathsf{Prop}_{\text{ups}}$, parties encode their inputs and send the codewords and hashes to the other parties, such that honest parties are able to reconstruct the proposed pre-block of the leader. Since parties might output a different pre-block than the one they started with in the current round, they must send the hash and the codewords of their new input during the next round.

$\mathsf{Prop}_{\mathbf{ups}}^{t_a,t_s}(r, B, C)$

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

Figure 5.7: A value-proposal protocol for UPSTATE from the perspective of party $P_{i\in\{1,\ldots,n\}}$ in round r.

The modifications to the graded consensus protocol are similarly straightforward; the main difference is that only the committee members vote.

For now, we assume that honest parties input $(1-t_s/n)\kappa$-quality pre-blocks of total length κ

$\mathsf{GC}_{\mathbf{ups}}^{t_a,t_s}(r, B, C)$

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

Figure 5.8: A graded consensus protocol for UPSTATE from the perspective of party $P_{i\in\{1,\dots,n\}}$ in round r.

to the block agreement protocol; later, this will be enforced by the SMR protocol. 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.

We now state the central lemma and sketch the main ideas of the security proof below. The formal proof can be obtained by expanding the proofs of the basic $\mathsf{BLA}_{\mathsf{tdg}}$ proofs in Section 4.2.

Lemma 5.10. $\mathsf{BLA}_{\mathsf{ups}}$ achieves $\hat{t}_s$-termination and $\hat{t}_s$-consistency (as defined in Definition 2.6) and $\hat{t}_s$-validity (as defined in Definition 5.1) except with negligible probability in the security parameter.

Sketch of proof. Parties terminate after participating in all R rounds of the protocol (even if they

```
$\mathsf{BLA}^{t_a,t_s}_{\textbf{ups}}(B_i)$

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

Figure 5.9: A BLA protocol for UPSTATE from the perspective of party $P_{i\in\{1,\ldots,n\}}$.

generated output earlier), so $\hat{t}_s$-termination is ensured by design.

We now argue $\hat{t}_s$-validity. Suppose all honest parties start $\mathsf{BLA}_{\text{ups}}$ with input B of quality $t_s\kappa/n$. If the leader is honest, all honest parties terminate $\mathsf{Prop}_{\text{ups}}$ with a pre-block B^* of quality $t_s\kappa/n$. This is because honest parties agree on the committee and can reconstruct their pre-blocks after $\mathsf{INSE}^{\kappa}_{\text{ups}}$ since the code tolerates t_s erasures (honest parties distinguish invalid signatures and ignore codewords with invalid signatures). Except with negligible probability, there will be at most $t_s\kappa/n$ corrupted parties in the primary committee, so $V_{\mathcal{C}}$ cannot contain $t_s\kappa/n + 1$ votes of corrupted parties. Thus, the proposal will be valid, ensuring all honest parties receive only valid proposals by time 5Δ and will output B^*. On the other hand, if the leader is dishonest, it can refuse to send a valid proposal by the required time. However, it cannot force honest parties to accept an invalid proposal since the adversary cannot tamper with the election of the primary committee and the corrupted leader cannot forge the signatures of the honest parties that are in the primary committee. In this case, the honest parties may terminate $\mathsf{Prop}_{\text{ups}}$ with output $\perp$.

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

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

5.2.3 UPSTATE: Full Protocol

We now present the full UPSTATE SMR protocol. UPSTATE achieves network-agnostic security for almost-optimal corruption thresholds against a static adversary. More formally:

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

Theorem 5.4. Under condition ♦ except with negligible probability, UPSTATE is (1) $\hat{t}_s$-consistent, UPSTATE

Step 1. Proposal selection.

1.1 At time $T_e = \mu(e-1)$: Set $B_i^e := (\perp, \dots, \perp)$ 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 $\mathsf{INSE}_{\text{ups}}(e, x_i)$ 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 $\mathsf{BLA}_{\text{ups}}$. If $\mathsf{BLA}_{\text{ups}}$ terminates, let B^* be the output.

2.2 At time $T_e + (7R+2)\Delta$: Terminate $\mathsf{BLA}_{\text{ups}}$ if not already terminated.

2.3 Pass B^* or wait until $\mathsf{ready}_e = \mathsf{true}$ and pass B_i^e as input to $\mathsf{ACS}_{\text{ups}}$.

2.4 Receive $S = \{B_j^*\}_{j \in S^*}$, where $S^* \subset \{1, \dots, n\}$ from $\mathsf{ACS}_{\text{ups}}$.

Step 3. Output and public verification.

3.1 Run Step 3 from UPDATE.

Figure 5.10: An SMR protocol with static security for party $P_{i \in \{1,\dots,n\}}$.

$\hat{t}_s$-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 (with all properties defined as in Definition 2.8).

The proof follows along the same lines as the proofs of Theorems 5.1–5.3, using the properties of subprotocols $\mathsf{ACS}_{\text{ups}}$ and $\mathsf{BLA}_{\text{ups}}$.

5.2.4 Communication Complexity of UPSTATE

$\mathsf{ACS}_{\text{ups}}$ has communication complexity $O(\kappa n\ell + \kappa^2 n^2)$ communication and $\mathsf{BLA}_{\text{ups}}$ has communication complexity $O(R\kappa^2 n^2 + \kappa n\ell)$, per input of size ℓ. In UPSTATE, $\mathsf{BLA}_{\text{ups}}$ and $\mathsf{ACS}_{\text{ups}}$ are run on pre-blocks of size $O(L|\mathsf{tx}|)$. If the network is synchronous, the output of $\mathsf{ACS}_{\text{ups}}$ 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.

UPSTATE incurs $O(n^2 L|\mathsf{tx}|/(\kappa b)+n^2\kappa)$ total communication for step 1.3 and $O(n^2\kappa L|\mathsf{tx}|/b+n^2\kappa^2)$ total communication in step 1.4. In step 2, $\mathsf{BLA}_{\text{ups}}$ incurs $O(R\kappa^2 n^2 + \kappa n L|\mathsf{tx}|)$ total communication and $\mathsf{ACS}_{\text{ups}}$ incurs $O(\kappa n L|\mathsf{tx}| + \kappa^2 n^2)$ total communication.

Summing over all steps and using the fact that $b = \hat{t}_s = O(n)$, we obtain a total communication of $O(R\kappa^2 n^2 + \kappa n L|\mathsf{tx}|)$. Setting the proposal sample size to $L = O(R\kappa n)$ yields total communication of $O(R\kappa^2 n^2)$ per block and amortized communication complexity of $O(\kappa n)$ per transaction.

Chapter 6: Achieving Security in a Variable Network-Agnostic Model

Up to this point, we have considered a network-agnostic model that is *static*, in the sense that the underlying network is assumed to be either synchronous or asynchronous for the entire lifetime of the protocol. In this chapter, we introduce a *variable* network-agnostic model, where the network can arbitrarily transition between synchronous and asynchronous. The adversary considered in this section is a *constrained epoch-mobile adaptive adversary* who can corrupt at most t_s unique parties over the duration of the protocol, and can move between those t_s parties from epoch to epoch, with certain restrictions. Namely, the adversary cannot not exceed the corruption limit (t_s corruptions while the network is synchronous and t_a while it is asynchronous) at any moment in time; nor can it exceed that limit in any single epoch. (Although this restriction might seem arbitrary, it was chosen to circumvent a particular impossibility result of [28]; see Section 6.4 for further discussion.)

The network-agnostic protocols discussed so far, UPDATE and UPSTATE, as well as TARDIGRADE, can be made secure in this new setting under the assumption that devices have a tamper-proof *reboot* mechanism. (E.g., each device might have a small region of untamperable memory with code that reboots the device into a clean state.) Given this mechanism, only two changes are needed:

1. Parties reboot at the start of each epoch, flushing out the adversary if the party was corrupted.
2. Any time UPDATE, UPSTATE, TARDIGRADE, or their subprotocols would use a plain synchronous or asynchronous BA protocol, we instead use the network-agnostic BA protocol presented earlier.

We omit pseudocode for the modified protocols, since it would be almost identical to the originals.

The analysis in the variable network-agnostic model follows in a relatively straightforward way from the original network-agnostic analysis. In the rest of this section, we give an overview of the most closely related prior and concurrent work, followed by a discussion of the most relevant technical details, and finally the formal proofs.

6.1 Related Work

In the *proactive model* [51], the adversary can be mobile across the corrupted parties over time. *Proactive secret sharing* (PSS) was introduced by Herzberg et al. [52]. Canetti et al. [53] and Frankel et al. [54] gave solutions for synchronous DKG against adaptive proactive adversaries using verifiable secret sharing schemes. Benhamouda et al. [55] introduced a secret-sharing protocol for passing secrets from one anonymous committee to another, while Groth [56] 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. [57] presented a proactive refresh protocol assuming clock ticks that define epochs, based on [58] which recovers state in an SMR protocol. Schulze et al. [59] proposed a mobile PSS protocol in a partially synchronous network. Recently, several works [60, 61, 62] have proposed more efficient dynamic/mobile PSS protocols assuming eventual synchrony, short periods of synchrony at the end of an epoch, or synchronized epochs. As somewhat of a special case (but relevant to our work), work by Gordon et al. considered synchronous authenticated broadcast with both corrupted parties and parties who are honest but whose keys have been exposed [63].

Following the publication of our initial results, a few works have considered secret sharing and distributed key generation in a network-agnostic model. One concurrent work by Appan et al. [64] proposed a protocol for network-agnostic perfectly secure multi-party computation; their protocol uses a novel network-agnostic perfectly secure verifiable secret sharing protocol (but not proactive). Another concurrent work by Bacho et al. [65] proposed a DKG protocol with optimal resilience in a network-agnostic setting; however, they consider only a static adversary. Concurrently to our work, Yurek et al. [66] constructed asynchronous dynamic PSS protocols using different target definitions than ours (thus bypassing our impossibility result).

6.2 Technical Details

As mentioned above, we assume that each device has a tamperproof reboot mechanism that forcibly evicts the adversary. (By tamperproof, we mean that the adversary cannot prevent the reboot if it is triggered.) Since a corrupted party may not "know" they have been corrupted, the reboot mechanism is run at predetermined intervals specified by the protocol. In our protocols, parties reboot at the start of each epoch.

For simplicity, we assume the reboot is instantaneous; otherwise, the timings of the protocol steps can be adjusted to compensate. The adversary is allowed to immediately recorrupt a party after a reboot, as long as corrupting the party does not exceed the allowed threshold determined by the network state.

We emphasize that 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. We refer to a party as *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.

Throughout, we also assume authenticated channels that are separate from the public key infrastructure; this assumption is standard in other proactive protocols (e.g., [57, 59]). As the adversary corrupts the parties and learns their secret keys, it can continue to use those keys (e.g., to decrypt communication, forge signatures, and create messages that appear to have been created by the corresponding party) even after it is flushed out by the reboot, violating agreement and confidentiality of the PKI.

Critically, even though the adversary has access to up to t_s keys and 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 have locally verified, such as a hash value whose corresponding preimage was correctly reconstructed.

6.3 Security Analysis

We now turn our attention to the security analysis. Our goal is to prove the following:

Theorem 6.1. Protocols TARDIGRADE, UPDATE, and UPSTATE with reboots are secure SMR protocols (as defined in Definition 2.8) under arbitrary network changes against a constrained epoch-mobile adaptive adversary, for any t_a, t_s satisfying the appropriate condition.

Because the arguments for each of the three protocols are similar, we will work through the proof for UPDATE in detail and then sketch the main ideas of the proofs for UPSTATE and TARDIGRADE.

Lemma 6.1. In an execution of $\mathsf{ACS}_{\text{upd}}$, 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.

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

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

Lemma 6.2. Suppose there are at most t_a corruptions and $t_s - t_a$ exposed parties during an execution of $\mathsf{ACS}_{\text{upd}}$. Given a certificate $(\mathsf{commit}, \langle h \rangle)$ for a party P, all honest parties eventually reconstruct the same output.

Proof. (Lemma 6.2) Since the adversary cannot act on behalf of the exposed parties directly, the arguments for when P is honest and unexposed, and for when P is dishonest are the same as the arguments in the proof of Lemma 5.1.

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

Lemma 6.3. In an execution of $\mathsf{ACS}_{\text{upd}}$, 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. Since the adversary cannot act directly on behalf of the exposed parties, the arguments for when P is honest and unexposed and when P is dishonest can be taken directly from the proof of Lemma 5.2.

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

We now have all of the supporting results needed to prove the part of Theorem 6.1 that pertains to UPSTATE.

Proof. (Theorem 6.1, UPSTATE) 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 5.1.2.

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 each building block, and then the security of the overall protocol.

First, note that the network-agnostic BA protocol (Section 3.3) 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.

Next, consider $\mathsf{ACS}_{\mathsf{upd}}$. 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 $\mathsf{INDI}_{\mathsf{upd}}$ as if they originated from the exposed parties. By Lemma 6.1, at least $n - t_a$ BA instances will still terminate, despite exposures. Coupled with Lemmas 6.2 and 6.3, which show there cannot be conflicting certificates and all honest parties are able to eventually correctly reconstruct the same input, it follows that $\mathsf{ACS}_{\mathsf{upd}}$ 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 5.3.

$\mathsf{BLA}_{\mathsf{upd}}$ uses a leader mechanism that outputs after it has received input from a strict majority of parties, hence it is still unpredictable in the presence of t_s exposed parties. The property required of $\mathsf{BLA}_{\mathsf{upd}}$ in the asynchronous case is the following: if an honest party does output in $\mathsf{BLA}_{\mathsf{upd}}$, 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.

We now consider the full SMR protocol. A corrupted party can include signatures from an exposed party in its own $(n - t_s)$-quality pre-block. Therefore, there may be up to t_a pre-blocks input to BLA that have only $n - 2t_s$ entries originating from honest parties. If such a block is output by $\mathsf{BLA}_{\mathsf{upd}}$, then the same holds for the the output of $\mathsf{ACS}_{\mathsf{upd}}$.

By t_a-consistency and t_a-validity with termination of $\mathsf{ACS}_{\mathsf{upd}}$, 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 is t_a-consistent and t_a-complete.

Next, we argue that t_a-liveness holds. If an adversarial pre-block is output by $\mathsf{ACS}_{\mathsf{upd}}$, 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. Nevertheless, an argument similar to the analysis in Section 4.3.3 shows that the probability that tx has been output after r epochs goes to 1 as r goes to infinity.

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

Finally, we observe that 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 $2t_s + t_a < n$. □

The parts of Theorem 6.1 concerning UPSTATE and TARDIGRADE follow similarly; brief proof sketches are given below.

Sketch of proof. (Theorem 6.1, UPSTATE) Despite knowing $\hat{t}_s$ keys, a static adversary cannot actively corrupt more than $t_a\kappa/n$ parties in any of the committees with high probability (see Appendix A), because it selects the corrupted parties before the epoch starts and committee membership is unpredictable.

The argument for t_a-security of UPSTATE under arbitrary network changes follows by a similar argument to the one used in the proof for UPDATE. □

As an aside, if an adaptive adversary knows $\hat{t}_s$ keys, it can corrupt up to $t_s\kappa/n$ parties in the secondary committees. Nevertheless, since the secondary committees are only used to construct certificates of $\hat{t}_s + 1$ keys, this does not present an issue.

Sketch of proof. (Theorem 6.1, TARDIGRADE) The t_s-valid and t_a-consistent reliable broadcast protocol used in the non-terminating protocol $\mathsf{ACS}^*_{\text{tdg}}$ is signature-free, and furthermore, $\mathsf{ACS}^*_{\text{tdg}}$ is signature-free (apart from the BA components). Thus, the proofs for t_a-consistency, t_s-validity, t_a-liveness, and t_a-set quality of $\mathsf{ACS}^*_{\text{tdg}}$ from 4.1.2 hold.

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

The arguments for $\mathsf{BLA}_{\text{tdg}}$ and the full protocol follow similarly to the proof for UPDATE.

□

6.4 Discussion

Above, we considered an adversary that is constrained in two notable ways: it can only corrupt up to t_s unique parties over the lifetime of the protocol, and it cannot corrupt more than the appropriate t_s, t_a threshold in any epoch *or* at any point in time. Some form of constraint is necessary, due to an impossibility result for asynchronous proactive secret sharing ([28], Theorem 6), which shows that it is not possible for an APSS scheme to have both liveness and privacy without either restricting the adversary's mobility or introducing some form of synchrony assumption, such as a time signal from an external clock. One could imagine other ways of circumventing this barrier, including solutions not based on APSS, or solutions that introduce alternative constraints; we believe this is an interesting direction for future work.

Appendix A: Deferred Probability Bounds

We now present deferred results on the composition of committees. The necessary results follow as a straightforward application of standard bounds (restated below as Lemma A.1 and A.2 for the reader's convenience).

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

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

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

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

$$\mathbb{P}[X \geq s] \leq e^{-\frac{(s - t\kappa/n)^2}{s+2-t\kappa/n}}, \text{if } s > t\kappa/n, \text{ by (A.1)}. \tag{A.3}$$

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

Lemma A.2 (Hoeffding's inequality)**.** Let X be a variable sampled from a binomial distribution with n independent trials and probability of success p. Then:

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

With those generic bounds in hand, we can prove several useful results regarding the composition of committees sampled by our two mechanisms, starting with the second (VRF-based) mechanism. For now we only consider the usual static and adaptive adversaries; results related to the mobile adversary follow later.

Lemma A.3. Fix $s \leq n$ and $0 < \epsilon < 1/3$. If $\mathcal{C} \leftarrow \chi_{s,n}$, then:

1. $\mathcal{C}$ contains fewer than $(1+\epsilon) \cdot s$ parties except with probability $e^{-\frac{\epsilon^2 s}{2+\epsilon}}$.
2. $\mathcal{C}$ contains more than $(1-\epsilon) \cdot s$ parties except with probability $e^{-\frac{\epsilon^2 s}{2}}$.
3. If there are at most $\hat{t}_s \leq (1-2\epsilon) \cdot t_s$ corrupted parties, then $\mathcal{C}$ contains fewer than $(1-\epsilon) \cdot s \cdot \frac{t_s}{n}$ corrupted parties except with probability at most $e^{-\epsilon^2 s/(4-6\epsilon)}$.
4. If there are at most t_a corrupted parties, then $\mathcal{C}$ contains more than $(1-\epsilon) \cdot s \cdot t_s/n$ honest parties except with probability at most $e^{-\frac{\epsilon^2 s}{3}}$.

Proof. Let $H \subseteq [n]$ be the indices of the honest parties. Let X_j be the Bernoulli random variable indicating if $P_j \in \mathcal{C}$, so $\Pr[X_j = 1] = s/n$. Define $Z_1 = \sum_j X_j$, $Z_2 := \sum_{j \notin H} X_j$, and $Z_3 := \sum_{j \in H} X_j$. Then:

1. Since $E[Z_1] = s$, setting $\delta = \epsilon$ in Lemma A.1 yields

$$\Pr\left[Z_1 \geq (1+\epsilon) \cdot s\right] \leq e^{-\epsilon^2 s/(2+\epsilon)}. \tag{A.6}$$

2. Using the other half of Lemma A.1, setting $\delta = \epsilon$ yields

$$\Pr\left[Z_1 \leq (1-\epsilon) \cdot s\right] \leq e^{-\epsilon^2 s/2}. \tag{A.7}$$

3. Since $E[Z_2] \leq \hat{t}_s \cdot s/n \leq (1-2\epsilon) \cdot t_s \cdot s/n$ and $t_s/n < 1/2$, setting $\delta = \frac{\epsilon}{1-2\epsilon}$ in Lemma A.1 yields

$$\Pr\left[Z_2 \geq \frac{(1-\epsilon) \cdot t_s \cdot s}{n}\right] \leq -e^{\epsilon^2 s/(4-6\epsilon)}. \tag{A.8}$$

4. Assuming that there are at most t_a corrupted parties, then $E[Z_3] \geq (n - t_a) \cdot s/n$. Thus, plugging in $\delta = \epsilon$, we have

$$\Pr\left[Z_3 \leq (1-\epsilon)(n-t_a) \cdot s/n\right] \leq e^{-\epsilon^2 \frac{(n-t_a)s}{2n}}. \tag{A.9}$$

Next, using the fact that $t_s/n < n/2$ and $(n-t_a)/n > 2n/3$, we see that

$$(1-\epsilon)s \cdot t_s/n < (1-\epsilon)s/2 < (1-\epsilon)s \cdot 2/3 < (1-\epsilon)s \cdot (n-t_a)/n. \tag{A.10}$$

Thus, $\Pr\left[Z_3 \leq (1-\epsilon)s \cdot t_s/n\right] \leq \Pr\left[Z_3 \leq (1-\epsilon)(n-t_a) \cdot s/n\right]$. Putting these two pieces together, we have $\Pr[Z_3 \leq (1-\epsilon)s \cdot t_s/n] \leq e^{-\frac{\epsilon^2(n-t_a)s}{2n}} \leq e^{-\frac{\epsilon^2 s}{3}}$ (note the last step is only used to simplify the bound).

□

Next, we present results regarding the first (hash-based) election mechanism in the presence of the constrained mobile adversary.

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

$$\Pr[Y_s \leq \kappa t_s/n] \leq e^{-\frac{\kappa(1-(2-\epsilon)t_s/n)^2}{1-(1-\epsilon)t_s/n}}. \tag{A.11}$$

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

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

$$\Pr[Y_s \leq (1-t_s/n)\kappa] \leq e^{-\frac{\kappa(\epsilon t_s/n)^2}{1-(1-\epsilon)t_s/n}}. \tag{A.12}$$

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

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

$$\Pr[X_a \geq \kappa t_a/n] \leq e^{-\frac{(\epsilon\kappa t_a/n)^2}{2-\epsilon\kappa t_a/n}}. \tag{A.13}$$

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

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

$$\Pr[X_s \geq \kappa t_s/n] \leq e^{-\frac{(\epsilon\kappa t_s/n)^2}{2+\epsilon\kappa t_s/n}}. \tag{A.14}$$

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

The following results concern the second (self-election) mechanism in the presence of the constrained mobile adversary.

Let Z denote the number of parties selected in the secondary committee $\bar{\mathcal{C}}$. The expected value of Z is $\mathbb{E}[Z] = \kappa$. By (A.2), the probability that committee $\bar{\mathcal{C}}$ has strictly fewer than $(1-\epsilon)\kappa + 1$ members is:

$$\Pr[Z \leq (1-\epsilon)\kappa] \leq e^{-\epsilon^2\kappa/2}. \tag{A.15}$$

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

$$\Pr[Z \geq (1+\epsilon)\kappa] \leq e^{-\epsilon^2\kappa/(2+\epsilon)}. \tag{A.16}$$

Let Y_s denote the number of honest parties among the randomly elected secondary committee members. In expectation, the number of honest parties selected will be greater than the initial fraction of honest parties times the committee size: $\mathbb{E}[Y_s] \geq (1-(1-\epsilon)t_s/n)\kappa$, so equation (A.11) holds for the secondary committee as well. Analogously, equation (A.14) holds for X_a, the number of corrupted parties among the randomly elected secondary committee members when the initial corruption threshold is t_a.

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

$$\Pr[W \geq t_a \kappa / n] \leq e^{-2\kappa(t_a/n - p)^2}. \tag{A.17}$$

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

Bibliography

[1] M. Pease, Robert E. Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, 1980.

[2] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The Byzantine generals problem. *ACM Trans. Programming Language Systems*, 4(3):382–401, 1982.

[3] Fred B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Comput. Surv.*, 22(4):299–319, dec 1990.

[4] Klaus Kursawe and Victor Shoup. Optimistic asynchronous atomic broadcast. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *ICALP 2005: 32nd International Colloquium on Automata, Languages and Programming*, volume 3580 of *Lecture Notes in Computer Science*, pages 204–215, Lisbon, Portugal, July 11–15, 2005. Springer, Heidelberg, Germany.

[5] Christian Cachin and Jonathan A Poritz. Secure intrusion-tolerant replication on the internet. In *Intl. Conf. on Dependable Systems and Networks*, pages 167–176. IEEE, 2002.

[6] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of BFT protocols. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, pages 31–42, Vienna, Austria, October 24–28, 2016. ACM Press.

[7] Sisi Duan, Michael K. Reiter, and Haibin Zhang. BEAT: Asynchronous BFT made practical. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018: 25th Conference on Computer and Communications Security*, pages 2028–2041, Toronto, ON, Canada, October 15–19, 2018. ACM Press.

[8] Bingyong Guo, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. Dumbo: Faster asynchronous BFT protocols. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020: 27th Conference on Computer and Communications Security*, pages 803–818, Virtual Event, USA, November 9–13, 2020. ACM Press.

[9] Ittai Abraham, Srinivas Devadas, Danny Dolev, Kartik Nayak, and Ling Ren. Synchronous Byzantine agreement with expected $O(1)$ rounds, expected $O(n^2)$ communication, and optimal resilience. In *Financial Cryptography and Data Security: 23rd International Conference, FC 2019*, Berlin, Heidelberg, 2019. Springer-Verlag.

[10] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Maofan Yin. Sync HotStuff: Simple and practical synchronous state machine replication. In *2020 IEEE Symposium on Security and Privacy*, pages 106–118, San Francisco, CA, USA, May 18–21, 2020. IEEE Computer Society Press.

[11] Chao Liu, Sisi Duan, and Haibin Zhang. EPIC: Efficient Asynchronous BFT with Adaptive Security. In *Intl. Conf. on Dependable Systems and Networks (DSN)*, pages 437–451. IEEE, 2020.

[12] Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. All you need is DAG. In *Symposium on Principles of Distributed Computing (PODC)*, pages 165–175. ACM, 2021.

[13] Yuan Lu, Zhenliang Lu, Qiang Tang, and Guiling Wang. Dumbo-mvba: Optimal multi-valued validated asynchronous Byzantine agreement, revisited. In *Proceedings of the 39th Symposium on Principles of Distributed Computing*, PODC '20, New York, NY, USA, 2020. Association for Computing Machinery.

[14] Miguel Castro and Barbara Liskov. Practical Byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, OSDI '99, pages 173–186. USENIX Association, 1999.

[15] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 281–310, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany.

[16] Rafael Pass, Lior Seeman, and abhi shelat. Analysis of the blockchain protocol in asynchronous networks. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part II*, volume 10211 of *Lecture Notes in Computer Science*, pages 643–673, Paris, France, April 30 – May 4, 2017. Springer, Heidelberg, Germany.

[17] Shengyun Liu, Paolo Viotti, Christian Cachin, Vivien Quema, and Marko Vukolic. XFT: Practical fault tolerance beyond crashes. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 485–500, Savannah, GA, November 2016. USENIX Association.

[18] Dahlia Malkhi, Kartik Nayak, and Ling Ren. Flexible Byzantine fault tolerance. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 1041–1053, New York, NY, USA, 2019. Association for Computing Machinery.

[19] Atsuki Momose and Ling Ren. Multi-threshold byzantine fault tolerance. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021: 28th Conference on Computer and Communications Security*, pages 1686–1699, Virtual Event, Republic of Korea, November 15–19, 2021. ACM Press.

[20] Yue Guo, Rafael Pass, and Elaine Shi. Synchronous, with a chance of partition tolerance. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part I*, volume 11692 of *Lecture Notes in Computer Science*, pages 499–529, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Heidelberg, Germany.

[21] Rafael Pass and Elaine Shi. Hybrid consensus: Efficient consensus in the permissionless model. In *31st International Symposium on Distributed Computing (DISC 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[22] Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 3–33, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany.

[23] Julian Loss and Tal Moran. Combining asynchronous and synchronous Byzantine agreement: The best of both worlds, 2018. Available at `http://eprint.iacr.org/2018/235`.

[24] Chen-Da Liu-Zhang, Julian Loss, Ueli Maurer, Tal Moran, and Daniel Tschudi. MPC with synchronous security and asynchronous responsiveness. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020, Part III*, volume 12493 of *Lecture Notes in Computer Science*, pages 92–119, Daejeon, South Korea, December 7–11, 2020. Springer, Heidelberg, Germany.

[25] Klaus Kursawe. Optimistic Byzantine agreement. In *Proceedings of the 21st IEEE Symposium on Reliable Distributed Systems*, SRDS '02, page 262. IEEE Computer Society, 2002.

[26] Erica Blum, Jonathan Katz, and Julian Loss. Synchronous consensus with optimal asynchronous fallback guarantees. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019: 17th Theory of Cryptography Conference, Part I*, volume 11891 of *Lecture Notes in Computer Science*, pages 131–150, Nuremberg, Germany, December 1–5, 2019. Springer, Heidelberg, Germany.

[27] Erica Blum, Jonathan Katz, and Julian Loss. Tardigrade: An atomic broadcast protocol for arbitrary network conditions. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2021, Part II*, volume 13091 of *Lecture Notes in Computer Science*, pages 547–572, Singapore, December 6–10, 2021. Springer, Heidelberg, Germany.

[28] Andreea B. Alexandru, Erica Blum, Jonathan Katz, and Julian Loss. State machine replication under changing network conditions. In Shweta Agrawal and Dongdai Lin, editors, *Advances in Cryptology - ASIACRYPT 2022*, 2022. `https://eprint.iacr.org/2022/698`.

[29] Kartik Nayak, Ling Ren, Elaine Shi, Nitin H Vaidya, and Zhuolun Xiang. Improved extension protocols for Byzantine broadcast and agreement. In *Intl. Symposium on Distributed Computing (DISC)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[30] Ran Canetti and Tal Rabin. Fast asynchronous Byzantine agreement with optimal resilience. *Proceedings of the twenty-fifth annual ACM symposium on Theory of Computing*, 1993.

[31] Ittai Abraham, Danny Dolev, and Joseph Y. Halpern. An almost-surely terminating polynomial protocol for asynchronous Byzantine agreement with optimal resilience. In *Proceedings of the Twenty-Seventh ACM Symposium on Principles of Distributed Computing*, PODC '08, page 405–414, New York, NY, USA, 2008. Association for Computing Machinery.

[32] Arpita Patra, Ashish Choudhary, and Chandrasekharan Pandu Rangan. Simple and efficient asynchronous Byzantine agreement with optimal resilience. In *Proceedings of the 28th ACM Symposium on Principles of Distributed Computing*, PODC '09, page 92–101, New York, NY, USA, 2009. Association for Computing Machinery.

[33] Achour Mostefaoui, Hamouma Moumen, and Michel Raynal. Signature-free asynchronous Byzantine consensus with $t < n/3$ and $o(n^2)$ messages. In *Proceedings of the 2014 ACM Symposium on Principles of Distributed Computing*, PODC '14, page 2–9, New York, NY, USA, 2014. Association for Computing Machinery.

[34] Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in constantipole: Practical asynchronous Byzantine agreement using cryptography (extended abstract). In *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '00, page 123–132, New York, NY, USA, 2000. Association for Computing Machinery.

[35] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Symposium on Operating Systems Principles (OSDI)*, pages 51–68. ACM, 2017.

[36] Timo Hanke, Mahnush Movahedi, and Dominic Williams. Dfinity technology overview series consensus system, rev. 1, 2018. Available at `https://dfinity.org/faq`.

[37] Ittai Abraham, T. H. Hubert Chan, Danny Dolev, Kartik Nayak, Rafael Pass, Ling Ren, and Elaine Shi. Communication complexity of Byzantine agreement, revisited. *Distributed Computing*, 36(1):3–28, 2023.

[38] Pesech Feldman and Silvio Micali. An optimal probabilistic protocol for synchronous Byzantine agreement. *SIAM J. Comput.*, 26:873–933, 1997.

[39] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 524–541, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Heidelberg, Germany.

[40] Ittai Abraham, Srinivas Devadas, Danny Dolev, Kartik Nayak, and Ling Ren. Efficient synchronous Byzantine consensus, 2017. Available at `https://eprint.iacr.org/2017/307`.

[41] Danny Dolev and H. Raymond Strong. Authenticated algorithms for Byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983.

[42] Sam Toueg. Randomized Byzantine agreements. In *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing*, PODC '84, page 163–178, New York, NY, USA, 1984. Association for Computing Machinery.

[43] Miguel Correia, Nuno Neves, and Paulo Veríssimo. From consensus to atomic broadcast: Time-free Byzantine-resistant protocols without signatures. *The Computer Journal*, 49(1):82–96, 2006.

[44] Erica Blum, Chen-Da Liu Zhang, and Julian Loss. Always have a backup plan: Fully secure synchronous MPC with asynchronous fallback. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part II*, volume 12171 of *Lecture Notes in Computer Science*, pages 707–731, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Heidelberg, Germany.

[45] Michael Ben-Or, Boaz Kelmer, and Tal Rabin. Asynchronous secure computations with optimal resilience (extended abstract). In Jim Anderson and Sam Toueg, editors, *13th ACM Symposium Annual on Principles of Distributed Computing*, pages 183–192, Los Angeles, CA, USA, August 14–17, 1994. Association for Computing Machinery.

[46] Gabriel Bracha. An asynchronous $[(n-1)/3]$-resilient consensus protocol. New York, NY, USA, 1984. Association for Computing Machinery.

[47] Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for Byzantine agreement. *JCSS*, 75(2):91–112, 2009.

[48] Elaine Shi. Foundations of distributed consensus and blockchains. Book manuscript, 2020. Available at `https://www.distributedconsensus.net.`

[49] Sourav Das, Zhuolun Xiang, and Ling Ren. Balanced quadratic reliable broadcast and improved asynchronous verifiable information dispersal. Cryptology ePrint Archive, Report 2022/052, 2022. `https://eprint.iacr.org/2022/052`.

[50] Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. Asymptotically optimal validated asynchronous Byzantine agreement. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, PODC '19, page 337–346, New York, NY, USA, 2019. Association for Computing Machinery.

[51] Rafail Ostrovsky and Moti Yung. How to withstand mobile virus attacks (extended abstract). In Luigi Logrippo, editor, *10th ACM Symposium Annual on Principles of Distributed Computing*, pages 51–59, Montreal, QC, Canada, August 19–21, 1991. Association for Computing Machinery.

[52] Amir Herzberg, Stanislaw Jarecki, Hugo Krawczyk, and Moti Yung. Proactive secret sharing or: How to cope with perpetual leakage. In Don Coppersmith, editor, *Advances in Cryptology – CRYPTO'95*, volume 963 of *Lecture Notes in Computer Science*, pages 339–352, Santa Barbara, CA, USA, August 27–31, 1995. Springer, Heidelberg, Germany.

[53] Ran Canetti, Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Adaptive security for threshold cryptosystems. In Michael J. Wiener, editor, *Advances in Cryptology – CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 98–115, Santa Barbara, CA, USA, August 15–19, 1999. Springer, Heidelberg, Germany.

[54] Yair Frankel, Philip D. MacKenzie, and Moti Yung. Adaptively-secure optimal-resilience proactive RSA. In Kwok-Yan Lam, Eiji Okamoto, and Chaoping Xing, editors, *Advances in Cryptology – ASIACRYPT'99*, volume 1716 of *Lecture Notes in Computer Science*, pages 180–194, Singapore, November 14–18, 1999. Springer, Heidelberg, Germany.

[55] Fabrice Benhamouda, Craig Gentry, Sergey Gorbunov, Shai Halevi, Hugo Krawczyk, Chengyu Lin, Tal Rabin, and Leonid Reyzin. Can a public blockchain keep a secret? In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020: 18th Theory of Cryptography Conference, Part I*, volume 12550 of *Lecture Notes in Computer Science*, pages 260–290, Durham, NC, USA, November 16–19, 2020. Springer, Heidelberg, Germany.

[56] Jens Groth. Non-interactive distributed key generation and key resharing. Cryptology ePrint Archive, Report 2021/339, 2021. `https://eprint.iacr.org/2021/339`.

[57] Christian Cachin, Klaus Kursawe, Anna Lysyanskaya, and Reto Strobl. Asynchronous verifiable secret sharing and proactive cryptosystems. In Vijayalakshmi Atluri, editor, *ACM CCS 2002: 9th Conference on Computer and Communications Security*, pages 88–97, Washington, DC, USA, November 18–22, 2002. ACM Press.

[58] Miguel Castro and Barbara Liskov. Proactive recovery in a Byzantine-Fault-Tolerant system. In *Symposium on Operating Systems Design and Implementation (OSDI)*, 2000. USENIX Association, 2000.

[59] David A. Schultz, Barbara Liskov, and Moses Liskov. Mobile proactive secret sharing. In Rida A. Bazzi and Boaz Patt-Shamir, editors, *27th ACM Symposium Annual on Principles of Distributed Computing*, page 458, Toronto, Ontario, Canada, August 18–21, 2008. Association for Computing Machinery.

[60] Sai Krishna Deepak Maram, Fan Zhang, Lun Wang, Andrew Low, Yupeng Zhang, Ari Juels, and Dawn Song. CHURP: Dynamic-committee proactive secret sharing. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019: 26th Conference on Computer and Communications Security*, pages 2369–2386, London, UK, November 11–15, 2019. ACM Press.

[61] Robin Vassantlal, Eduardo Alchieri, Bernardo Ferreira, and Alysson Bessani. Cobra: Dynamic proactive secret sharing for confidential BFT services. In *IEEE Symposium on Security and Privacy (SP)*, pages 1528–1528. IEEE Computer Society, 2022.

[62] Matthieu Rambaud and Antoine 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`.

[63] S Dov Gordon, Jonathan Katz, Ranjit Kumaresan, and Arkady Yerukhimovich. Authenticated broadcast with a partially compromised public-key infrastructure. *Information and Computation*, 234:17–25, 2014. Elsevier, 2014.

[64] Ananya Appan, Anirudh Chandramouli, and Ashish Choudhury. Perfectly-secure synchronous MPC with asynchronous fallback guarantees. Cryptology ePrint Archive, Report 2022/109, 2022. `https://eprint.iacr.org/2022/109`.

[65] Renas Bacho, Daniel Collins, Chen-Da Liu-Zhang, and Julian Loss. Network-agnostic security comes for free in DKG and MPC. Cryptology ePrint Archive, Report 2022/1369, 2022. `https://eprint.iacr.org/2022/1369`.

[66] Thomas Yurek, Zhuolun Xiang, Yu Xia, and Andrew 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`.