Efficient Byzantine Agreement with Faulty Minority[⋆]

Zuzana Beerliová-Trubíniová, Martin Hirt, and Micha Riser

ETH Zurich, Department of Computer Science,
{bzuzana,hirt}@inf.ethz.ch, micha@povworld.org

Abstract. Byzantine Agreement (BA) among n players allows the players to agree on a value, even when up to t of the players are faulty.
In the broadcast variant of BA, one dedicated player holds a message, and all players shall learn this message. In the consensus variant of BA, every player holds (presumably the same) message, and the players shall agree on this message.
BA is the probably most important primitive in distributed protocols, hence its efficiency is of particular importance.
BA from scratch, i.e., without a trusted setup, is possible only for $t < n/3$. In this setting, the known BA protocols are highly efficient ($\mathcal{O}(n^2)$ bits of communication) and provide information-theoretic security.
When a trusted setup is available, then BA is possible for $t < n/2$ (consensus), respectively for $t < n$ (broadcast). In this setting, only computationally secure BA protocols are reasonably efficient ($\mathcal{O}(n^3\kappa)$ bits).
When information-theoretic security is required, the most efficient known BA protocols require $\mathcal{O}(n^{17}\kappa)$ bits of communication per BA, where κ denotes a security parameter. The main reason for this huge communication is that in the information-theoretic world, parts of the setup are *consumed* with every invocation to BA, and hence the setup must be refreshed. This refresh operation is highly complex and communication-intensive.
In this paper we present BA protocols (both broadcast and consensus) with information-theoretic security for $t < n/2$, communicating $\mathcal{O}(n^5\kappa)$ bits per BA.

Keywords: Byzantine agreement, broadcast, consensus, information-theoretic security, multi-party computation, efficiency.

1 Introduction

1.1 Byzantine Agreement, Consensus, and Broadcast

The problem of Byzantine agreement (BA), as originally proposed by Pease, Shostak, and Lamport [PSL80], is the following: n players $P_1, \ldots, P_n$ want to reach agreement on some message m, but up to t of them are faulty and try

[⋆] This work was partially supported by the Zurich Information Security Center. It represents the views of the authors.

to prevent the others from reaching agreement. There are two flavors of the BA problem: In the broadcast problem, a designated player (the sender) holds an input message m, and all players should learn m and agree on it. In the consensus problem, every player P_i holds (supposedly the same) message m_i, and the players want to agree on this message.

More formally, a protocol with P_S giving input m is a *broadcast protocol*, when every honest P_i outputs the same message $m'_i = m'$ for some m' (*consistency*), and when $m' = m$, given that P_S is honest (*validity*). Analogously, a protocol with every player P_i giving input m_i is a *consensus protocol*, when every honest P_i outputs $m'_i = m'$ for some m' (*consistency*), and when $m' = m$, given that every honest P_i inputs the same message $m_i = m$ for some m (*validity*).

1.2 Models and Bounds

We assume that the players are connected with a complete synchronous network of secure channels. Complete means that each pair of players shares a channel. Synchronous means that all players share a common clock and that the message delay in the network is bounded by a constant.

The feasibility of broadcast and consensus depends on whether or not a trusted setup (e.g. a PKI setup) is available. When no trusted setup is available ("from scratch"), then consensus and broadcast are achievable if and only if at most $t < n/3$ players are corrupted. When a trusted setup is available, then consensus is achievable if and only if at most $t < n/2$ players are corrupted, and broadcast is achievable if and only if at most $t < n$ players are corrupted. All bounds can be achieved with information-theoretical security, and the bounds are tight even with respect to cryptographic security. We stress in particular that no broadcast protocol (even with cryptographic intractability assumptions) can exceed the $t < n/3$ bound unless it can rely on a trusted setup [FLM86,Fit03]. The main difference between protocols with information-theoretic security and those with cryptographic security is their efficiency.

1.3 Efficiency of Byzantine Agreement

We are interested in the communication complexity of BA protocols. The *bit complexity* of a protocol is defined as the number of bits transmitted by all honest players during the whole protocol, overall.

In the model without trusted setup, Byzantine agreement among n players is achievable for $t < n/3$ communicating $\mathcal{O}(n^2)$ bits [BGP92,CW92]. In the model with a trusted setup, the communication complexity of BA heavily depends on whether information-theoretic security is required or cryptographic security is sufficient. When cryptographic security is sufficient, then $\mathcal{O}(n^3\kappa)$ bits are sufficient for reaching agreement, where κ denotes the security parameter [DS83]. When information-theoretic security is desired, then reaching agreement requires at least $\mathcal{O}(n^6\kappa)$ bits of communication [BPW91,PW96,Fit03].

However, the latter result *consumes* the setup, i.e., a given setup can be used only for one single BA operation. Of course, one can start with a m times larger setup which supports m BA operations, but the number of BA operations is a priori fixed, and the size of the setup grows linearly with the number of intended BA operations. This diametrically contrasts the cryptographic scenario, where a fixed-size setup is sufficient for polynomially many BA operations. In [PW96], a method for *refreshing* the setup is shown: They start with a compact setup, use some part of the setup to perform the effective BA operation, and the remaining setup to generate a new, full-fledged setup. With this approach, a constant-size setup is sufficient for polynomially many BA invocations. However, with every BA invocation, the setup must be refreshed, which requires a communication of $\mathcal{O}(n^{17}\kappa)$ bits [PW96,Fit03]. Hence, when the initial setup should be compact, then the costs for a BA operation of [PW96] is as high as $\mathcal{O}(n^{17}\kappa)$ bits.

1.4 Contributions

We present a protocol for information-theoretically secure Byzantine agreement (both consensus and broadcast) which communicates $\mathcal{O}(n^4\kappa)$ bits when the setup may be consumed (i.e., the number of BA operations per setup is a priori fixed). This contrasts to the communication complexity of $\mathcal{O}(n^6\kappa)$ bits of previous information-theoretically secure BA protocols [BPW91,PW96].

More importantly, we present a refresh operation for our BA protocol, communicating only $\mathcal{O}(n^5\kappa)$ bits, contrasting the complexity of $\mathcal{O}(n^{17}\kappa)$ bits of previous refresh protocols [PW96]. This new results allows for polynomially many information-theoretically secure BA operations from a fixed-size setup, where each BA operations costs $\mathcal{O}(n^5\kappa)$ bits.

This substantial speed-up is primarily due to a new concept, namely that the refresh operation does not need to succeed all the time. Whenever the setup is to be refreshed, the players try to do so, but if they fail, they pick a fresh setup from an a priori prepared stock. Furthermore, using techniques from the player-elimination framework [HMP00], the number of failed refresh operations can be limited to t. Using algebraic information-theoretic pseudo-signatures [SHZI02] for appropriate parameters, the function to be computed in the refresh protocol becomes algebraic, more precisely a circuit over a finite field with multiplicative depth 1. Such a function is very well suited for efficient non-robust computation; in fact, it can be computed based on a simple one-dimensional Shamir-sharing, although $t < n/2$.[1] This allows a very simple refresh protocol with low communication overhead.

Compared to the refresh protocol of [PW96], our refresh protocol has the disadvantage that it requires $t < n/2$, whereas the protocol of [PW96] can cope with $t < n$. However, almost all applications using BA as sub-protocol (like voting, biding, multi-party computation, etc.) inherently require $t < n/2$, hence the limitation on our BA protocol is usually of theoretical relevance only.

[1] Note that general MPC protocols for this model need a three-level sharing, namely a two-dimensional Shamir sharing ameliorated with authentication tags [RB89,Bea91,BH06].

2 Preliminaries

2.1 Formal Model

We consider a set of n players $\mathcal{P} = \{P_1, \ldots, P_n\}$, communicating over pairwise secure synchronous channels. Many constructions require a finite field $\mathcal{F}$; we set this field to $\mathcal{F} = GF(2^\kappa)$ where κ is a security parameter (we allow a negligible error probability of $\mathcal{O}(2^{-\kappa})$). To every player $P_i \in \mathcal{P}$, a unique non-zero element $\alpha_i \in \mathcal{F} \setminus \{0\}$ is assigned. The faultiness of players is modeled by a central computationally unlimited adversary adaptively corrupting up to $t < n/2$ players and taking full control over them.

We assume that there is a trusted setup, i.e., before the protocol starts, a fixed probabilistic function $\mathsf{Init} : 1^\kappa \rightarrow (\mathsf{state}_1, \ldots, \mathsf{state}_n)$ is run by a trusted party, and every player $P_i \in \mathcal{P}$ secretly receives state_i as his initial state.

2.2 Information-Theoretically Secure Signatures

A classical (cryptographic) signature scheme consists of three algorithms: KeyGen, Sign, and Verify. KeyGen generates two keys, a *signing key* for the signer and a public *verification key*; Sign computes a signature for a given message and a given signing key; and Verify checks whether a signature matches a message for a given verification key. A secure signature scheme must satisfy that every signature created by Sign is accepted by Verify (with the corresponding signing/verification keys, completeness), and without the signing key it is infeasible to compute a signature which is accepted by Verify (unforgeability). Classical signature schemes provide cryptographic security only, i.e., an unbounded forger can always find an accepting signature for any given message, with exhaustive search, using Verify as test predicate.

As an information-theoretically secure signature scheme must be secure even with respect to a computationally unbounded adversary, every verifier must have a different verification key, and these verification keys must be kept private. Thus it cannot be automatically guaranteed that a signature is either valid for all verifiers or for no verifier (it might be valid for one verifier, but invalid for another one). Therefore, an additional property called *transferability* is required: It is impossible for a faulty signer to produce a signature which, with non-negligible probability, is valid for some honest verifier without being valid for some other honest verifier. We say that a signature scheme is information-theoretically secure if it is complete, unforgeable and transferable.

In [SHZI02], a so called (ψ, ψ')-secure signature scheme is presented, which allows the signer to sign a message $m \in \mathcal{F}$ such that any of the players in $\mathcal{P}$ can verify the validity of the signature. As long as the signer signs at most ψ messages and each verifier verifies at most ψ' signatures the success probability of attacks is less then $1/|\mathcal{F}| = 2^{-\kappa}$.

Here, we use a one-time signature scheme (i.e., one setup allows only for one single signature), where every verifier may verify up to $t + 2$ signatures (of the same signer). In context of [SHZI02], this means that we set $\psi = 1$ and $\psi' = t+2$.

By simplifying the notation (and by assuming that $2t+1 \leq n$), we receive the following scheme:

KeyGen: Key generation takes as input the string 1^κ, and outputs the signing key sk to the signer P_S and the n verification keys $\mathsf{vk}_1, \ldots, \mathsf{vk}_n$ to the respective verifiers $P_1, \ldots, P_n$. The signing key is a random vector $\mathsf{sk} = (p_0, \ldots, p_{n+1}, q_0, \ldots, q_{n+1}) \in \mathcal{F}^{2(n+2)}$, defining the polynomial

$$\begin{aligned} F_{\mathsf{sk}}(V_1, \ldots, V_{n+1}, M) &= \left(p_0 + \sum_{j=1}^{n+1} p_j V_j\right) + M \left(q_0 + \sum_{j=1}^{n+1} q_j V_j\right) \\ &= p_0 + M q_0 + \sum_{j=1}^{n+1} (p_j + M q_j)\, V_j . \end{aligned}$$

The verification key vk_i of each player $P_i \in \mathcal{P}$ is the vector $\mathsf{vk}_i = (v_{i,1}, \ldots, v_{i,n+1}, x_i, y_i)$, where the values $v_{i,1}, \ldots, v_{i,n+1}$ are chosen uniformly at random from $\mathcal{F}$, and the x_i- and y_i-values characterize the polynomial F_{sk}, when applied to $v_{i,1}, \ldots, v_{i,n+1}$, i.e., $x_i = p_0 + \sum_{j=1}^{n+1} p_j v_{i,j}$ and $y_i = q_0 + \sum_{j=1}^{n+1} q_j v_{i,j}$.

Sign: The signature σ of a message $m \in \mathcal{F}$ is a vector $\sigma = (\sigma_0, \ldots, \sigma_{n+1})$, characterizing the polynomial F_{sk} when applied to m, i.e., $\sigma_j = p_j + m q_j$ for $j = 0, \ldots, n+1$.

Verify: Given a message m, a signature $\sigma = (\sigma_0, \ldots, \sigma_{n+1})$, and the verification key $\mathsf{vk}_i = (v_{i,1}, \ldots, v_{i,n+1}, x_i, y_i)$ of player P_i, the verification algorithm checks whether

$$x_i + m y_i \stackrel{?}{=} \sigma_0 + \sum_{j=1}^{n+1} \sigma_j v_{i,j} \qquad \Big(= F_{\mathsf{sk}}(v_{i,1}, \ldots, v_{i,n+1}, m)\Big).$$

The protocol has the following sizes: Signing key: $(2n+4)\kappa$ bits; verification key: $(n+3)\kappa$ bits; signature: $(n+2)\kappa$ bits. The total information distributed for one signature scheme (called *sig-setup*) consists of $(n^2 + 5n + 8)\kappa$ bits.

Note that a sig-setup for the player set $\mathcal{P}$ is trivially also a valid sig-setup for every player subset $\mathcal{P}' \subseteq \mathcal{P}$. We will need this observation later.

3 Protocol Overview

Basically, the new broadcast protocol is the protocol of [DS83], ameliorated with information-theoretically secure signatures [SHZI02]. Similarly to [PW96], we start with a compact (constant-size) setup, which allows only for few broadcasts, and use some of these broadcasts for broadcasting the payload, and some of them to refresh the remaining setup, resulting in a fresh, full-fledged setup.

We borrow ideas from the player-elimination framework [HMP00] to substantially speed-up the refresh protocol: The generation of the new setup is performed non-robustly, i.e., it may fail when an adversary is present, but then the failure is detected by (at least) one honest player. At the end of the refresh protocol, the players jointly decide (using one BA-Operation) whether the refresh has succeeded or not; if yes, they are happy to have generated a new setup. If it failed, they run a fault-handling procedure, which yields a set E of two players, (at least) one of them faulty. As originally the set $\mathcal{P}$ contains an honest majority, also the set $\mathcal{P} \setminus E$ contains an honest majority. So the player set is reduced to $\mathcal{P}' \leftarrow \mathcal{P} \setminus E$ (with at most $t' \leftarrow t-1$ faulty players).

We are still missing the fresh setup; however, as with each fault-handling, one faulty player is eliminated from the actual player set, faults can occur only t times. For these t cases, we have a stock of t prepared setups, and with each fault, we take one out of this stock. This way it is ensured that at any point in the protocol, we have t' prepared setups on stock, where t' is the maximum number of faulty players in $\mathcal{P}'$. More precisely, the protocol runs as follows:

Initial Setup: The procedure Init generates $2+5t$ BA-setups[2]; one for the first BA operation, one for the first invocation of the refresh protocol, and t extra setups for the stock, each consisting of 2 BA-setups for replacing the failed refresh and 3 BA-setups for localizing the set $E \subseteq \mathcal{P}$ in the fault-handling procedure. The actual player set is set to $\mathcal{P}' = \mathcal{P}$ and the maximum number of faulty players in $\mathcal{P}'$ to $t' = t$.

Broadcast/Consensus: To perform a BA operation, the protocol Broadcast, resp. Consensus is invoked with the payload. In parallel, Refresh is invoked to refresh the reduced setup. If successful, Refresh produces two BA-setups using only one single BA operation. If Refresh fails, 5 BA-setups are taken from the stock, an elimination set $E \subseteq \mathcal{P}'$ is localized (using 3 BA's) and eliminated ($\mathcal{P}' \leftarrow \mathcal{P}' \setminus E$, $t' \leftarrow t'-1$), and the two remaining BA-setups are kept as new state – for the next Broadcast/Consensus operation.

In our presentation, we ignore the fact that faulty players can sent *no* message (or a message in a wrong format) when they are expected to send a message to an honest player. As general rule, we assume that when an honest player does not receive an expected message, he behaves as if he had received the zero-message.

4 Broadcast and Consensus

We present the protocols for the actual broadcast and consensus operation.

Note that the Refresh protocol outputs correct BA setup for $\mathcal{P}'$ only (rather than $\mathcal{P}$). However, as $\mathcal{P} \setminus \mathcal{P}'$ might contain honest players we need to achieve BA in $\mathcal{P}$. We first present the BA protocols for $\mathcal{P}'$, then show how to realize BA in $\mathcal{P}$ using these protocols.

As [SHZI02] signatures can cope only with message in the field $\mathcal{F}$, also our BA protocols are limited to messages $m \in \mathcal{F}$. An extension to longer messages is sketched in Appendix A.

[2] Init invokes KeyGen $4+10t$ times in parallel for each signer $P_S \in \mathcal{P}$. As will become clear later, $2n$ sig-setups are equivalent to one BA-setup.

We first present a broadcast protocol that allows a sender $P_S \in \mathcal{P}'$ to consistently distribute a message $m \in \mathcal{F}$ to the players in $\mathcal{P}'$.[3] The protocol is essentially the protocol of [DS83], with a simplified description of [Fit03]. In addition, the protocol is modified such that in one protocol run every player verifies at most $\psi' = t+2$ signatures of each signer (as required by our signature scheme).

Every player maintains a set $\mathcal{A}$ of accepted messages, a set $\mathcal{N}$ of newly accepted messages, and (one or several) sets Σ_m of received signatures for a message m.

Protocol Broadcast'**.**

0. Sender P_S: Send m and the corresponding signature σ_S to all $P_i \in \mathcal{P}'$.
1. $\forall P_i \in \mathcal{P}'$: If P_i received from the sender a message m together with a valid signature σ_S set $\mathcal{A} = \mathcal{N} = \{m\}$ and $\Sigma_m = \{\sigma_S\}$.

k. In each Step $k = 2, \ldots, t' + 1$, execute the following sub-steps for every player $P_i \in \mathcal{P}' \setminus \{P_S\}$:

$k.1$ For every message $m \in \mathcal{N}$, compute the signature σ_i on m, and send $(m, \Sigma_m \cup \{\sigma_i\})$ to all players in $\mathcal{P}'$. Set $\mathcal{N} = \{\}$.

$k.2$ In turn, for every message (m, Σ_m) received in Sub-step $k.1$ do:

- If $m \in \mathcal{A}$, or if $|\mathcal{A}| \geq 2$, ignore the message,
- else if Σ_m contains valid signatures from at least k different players in $\mathcal{P}'$, including P_S, include m in $\mathcal{A}$ and in $\mathcal{N}$,
- else ignore the received message (m, Σ_m) and all further messages from the player who has sent it.

$t'+2$. $\forall P_i$: if $|\mathcal{A}| = 1$, then accept $m \in \mathcal{A}$ as the broadcasted value. Otherwise, the sender is faulty, and accept $m =\perp$ (or any fixed pre-agreed value from $\mathcal{F}$) as the broadcasted value.

One can easily verify that the protocol Broadcast' is as secure as the used signature scheme [DS83,Fit03] and that every player verifies at most $t+2$ signatures from the same signer. Furthermore, every signer P_i issues up to two signatures; however, the second one is for the sole goal of proving to other players that the sender P_S is faulty, and the secrecy of P_i's signing key is not required anymore. Hence, it is sufficient to use a one-time signature scheme, whose unforgeability property is broken once the signer issues two signatures.

To construct a consensus protocol in $\mathcal{P}'$, we use a trick of [Fit04]: Every player needs two sig-setups, a primary scheme for the same purpose as in the above protocol, and an alternative scheme for identifying the message (if there is any) originally held by the majority of the players. During the protocol execution, every player P_i additionally maintains (one or several) sets Σ'_m, containing alternative signatures σ'_j (issued by P_j) for m, where Σ'_m with $|\Sigma'_m| \geq n' - t'$ now "replaces" the sender's signature in the above broadcast protocol. Now we present the consensus protocol for $\mathcal{P}'$, each P_i holding a message $m_i \in \mathcal{F}$:

[3] Note that Broadcast' will not be used in the paper, it is presented only for the sake of clarity of the protocol Consensus'.

Protocol Consensus'**.**

0. $\forall P_i \in \mathcal{P}'$: Send m_i and the corresponding (alternative) signature σ'_i to all players in $\mathcal{P}'$.
1. $\forall P_i \in \mathcal{P}'$: If there exists a message m received (together with a valid signature) from at least $n' - t'$ different players, let Σ'_m denote the set of all these signatures, and set $\mathcal{A} = \mathcal{N} = \{m\}$ and $\Sigma_m = \{\}$. If no such message exists, set $\mathcal{A} = \mathcal{N} = \{\}$.

k. In each Step $k = 2, \ldots, t' + 2$, execute the following sub-steps for every player $P_i \in \mathcal{P}'$:

k.1 For every message $m \in \mathcal{N}$, compute the signature σ_i on m, and send $(m, \Sigma'_m, \Sigma_m \cup \{\sigma_i\})$ to all players in $\mathcal{P}'$. Set $\mathcal{N} = \{\}$.

k.2 In turn, for every message (m, Σ'_m, Σ_m) received in Sub-step k.1 do:

- If $m \in \mathcal{A}$, or if $|\mathcal{A}| \geq 2$, ignore the message,
- else if Σ_m contains valid signatures in the primary scheme from at least $k - 1$ different players in $\mathcal{P}'$, and Σ'_m contains valid signatures in the alternative scheme from at least $n' - t'$ different players in $\mathcal{P}'$, then include m in $\mathcal{A}$ and in $\mathcal{N}$,
- else ignore the received message (m, Σ'_m, Σ_m) and all further messages from the player who has sent it.

$t'+3$. $\forall P_i$: if $|\mathcal{A}| = 1$, accept $m \in \mathcal{A}$ as the agreed value, otherwise (there was no pre-agreement) accept $m =\perp$.

The security of the protocol Consensus' follows immediately from the security of the protocol Broadcast', and the fact that every player issues at most one signature in the alternative scheme, and each such signature is verified at most $t + 1$ times. The communication complexity of BA in $\mathcal{P}'$ is at most $4n^3|\sigma| + 3n^2\kappa + n^2|\sigma| = (8n^4 + 26n^3 + 9n^2)\kappa$.

Broadcast and consensus in $\mathcal{P}$ can be constructed from consensus in $\mathcal{P}'$:

Protocol Broadcast**.**

1. The sender $P_S \in \mathcal{P}$ sends the message m to every player $P_j \in \mathcal{P}'$.
2. Invoke Consensus' to reach agreement on m among $\mathcal{P}'$.
3. Every player $P_i \in \mathcal{P}'$ sends the agreed message m to every player $P_j \in \mathcal{P}$.
4. Every player $P_j \in \mathcal{P}$ accepts the message m which was received most often.

Protocol Consensus**.**

1. Invoke Consensus' to reach agreement on m among $\mathcal{P}'$.
2. Every player $P_i \in \mathcal{P}'$ sends the agreed message m to every player $P_j \in \mathcal{P}$.
3. Every player $P_j \in \mathcal{P}$ accepts the message m which was received most often.

The security of these protocols follows from the security of Consensus' and from $t' < n'/2$ and $t < n/2$. The communication complexity of BA in $\mathcal{P}$ is at most $(8n^4 + 26n^3 + 11n^2)\kappa$.

5 Refreshing the Setup

5.1 Overview

To "refresh" the setup means to compute a new setup which allows for two BA operations, while this computation consumes only one BA-setup. The protocol Refresh generates the new setup with a special-purpose MPC among the players in $\mathcal{P}'$. This computation is performed *non-robustly*: Every sub-protocol either achieves its intended goal, or it fails. When it fails, then at least one honest player detects the failure. We do not require agreement on the fact whether or not a sub-protocol has failed. Only at the very end of Refresh, the players agree on whether or not a player has detected a failure during the computation (using consensus, thereby consuming one BA-setup). The computation takes only random values as input, so in case of failure, privacy is of no interest.

The computation of the verification keys will not only be non-robust, but even non-detectable, i.e., it might output wrong values without any (honest) player detecting the failure. However, once the verification keys are generated, their correctness is verified, and honest players can detect whether or not there was a failure.

We provide a fault-handling sub-protocol, to be invoked when Refresh fails, which localizes a set $E \subseteq \mathcal{P}'$ of two players, where (at least) one of them is faulty. This allows to reduce the actual player set, thereby reducing the maximum number of faulty players, thereby limiting the number of times Refresh can fail. In this fault-handling sub-protocol, every players sends to some designated player all messages he has received during the course of the protocol, as well as all random elements he sampled (which define the sent messages). Given this information, the designated player can help to compute the set E to eliminate.

In the sequel, we present the used sub-protocols (all of them non-robust), and finally the protocols Refresh and FaultHandling. The protocol Refresh invokes once the protocol Consensus', hence it consumes one valid BA-setup. The protocol FaultHandling invokes 3 times the protocol Broadcast; it requires enough BA-setups for that. However, the protocol FaultHandling is invoked only t times in total, so the required BA-setups can be prepared at beforehand.

For the sake of a simpler presentation, we give to every player P_i a flag fail_i, which is initialized to false, and is set to true once P_i has detected a failure. We say that a protocol *succeeds* when no player has detected a failure; otherwise, the protocol *fails*.

5.2 Secret Sharing

We use standard Shamir sharing [Sha79]. We say that a value a is t'-shared among the players $\mathcal{P}'$ if there exists a degree-t' polynomial $f(\cdot)$ with $f(0) = a$, and every (honest) player $P_i \in \mathcal{P}'$ holds a share $\langle a \rangle_i = f(\alpha_i)$, where α_i is the unique evaluation point assigned to P_i. We denote the collection of shares as $\langle a \rangle$. Observe that we can easily add up shared values, namely $\langle a + b \rangle = (\langle a \rangle_1 + \langle b \rangle_1, \ldots, \langle a \rangle_{n'} + \langle b \rangle_{n'})$. We write $\langle a \rangle + \langle b \rangle$ as a short hand.

In order to let a dealer $P_D \in \mathcal{P}'$ verifiably share a value a according to Shamir sharing, we employ the following (non-robust) protocol (based on the VSS protocol of [BGW88])

Protocol Share.

1. *(Distribution.)* P_D selects the coefficients $c_{0,1}, c_{1,0}, \ldots, c_{t',t'}$ at random, and sets $f(x,y) = a + c_{1,0}x + c_{0,1}y + c_{1,1}xy + \ldots + c_{t',t'}x^{t'}y^{t'}$. Then, to every $P_i \in \mathcal{P}'$, P_D computes and sends the polynomials $f_{i,\star}(y) = f(\alpha_i, y)$ and $f_{\star,i}(x) = f(x, \alpha_i)$.
2. *(Checking.)* For every pair $P_i, P_j \in \mathcal{P}'$, P_i sends $f_{i,\star}(\alpha_j)$ to P_j, who compares the received value with $f_{\star,j}(\alpha_i)$. P_j sets $\mathsf{fail}_j = \mathsf{true}$ if some difference is non-zero.
3. *(Output)* Every P_j outputs $\langle a \rangle_i = f_{i,\star}(0)$.

Lemma 1. *The protocol* Share *has the following properties: (Completeness) If all players in $\mathcal{P}'$ correctly follow the protocol, then the protocol succeeds. (Correctness) If the protocol succeeds, then the outputs $(\langle a \rangle_1, \ldots, \langle a \rangle_{n'})$ define a degree-t' polynomial $f(\cdot)$. (Validity & Privacy) If the protocol succeeds and the dealer is honest with input a, then $f(0) = a$ and no subset of t' players obtains any information on a. (Complexity) The protocol communicates at most $(2n^2 - 2n)\kappa$ bits and requires at most $(^1\!/\!_4 n^2 + {}^1\!/\!_2 n - {}^3\!/\!_4)\kappa$ random bits.*

The following protocol lets the players in $\mathcal{P}'$ reconstruct a correctly Shamir shared value a towards a designated player $P_R \in \mathcal{P}'$:

Protocol Recons.

1. Every player $P_i \in \mathcal{P}'$ sends his share $\langle a \rangle_i$ to the recipient P_R.
2. P_R verifies whether $\langle a \rangle_1, \ldots, \langle a \rangle_{n'}$ lie on a degree-t' polynomial $f(\cdot)$ and outputs $a = f(0)$ if yes. Otherwise, P_R sets $\mathsf{fail}_R = \mathsf{true}$ and outputs $a = 0$.

Lemma 2. *The protocol* Recons *has the following properties: (Completeness) If all players in $\mathcal{P}'$ correctly follow the protocol, then the protocol succeeds. (Correctness) If the protocol succeeds, then P_R outputs the correct secret a. (Complexity) The protocol communicates at most $(n-1)\kappa$ bits and requires no randomness.*

5.3 Generating Random Values

We present a (trivial) protocol that allows the players to generate a random value $c \in_R \mathcal{F}$, known to all players in $\mathcal{P}'$.

Protocol GenerateRandom.

1. $\forall P_i \in \{P_1, \ldots, P_{t'+1}\}$: select a random value $c_i \in_R \mathcal{F}$ and invoke Share to share c_i among $\mathcal{P}'$.
2. The players compute $\langle c \rangle = \sum_{i=1}^{t'+1} \langle c_i \rangle$.

3. $\forall P_k \in \mathcal{P}'$: invoke Recons to reconstruct $\langle c \rangle$ towards player P_k.

Lemma 3. *The protocol* GenerateRandom *has the following properties: (Completeness) If all players in $\mathcal{P}'$ correctly follow the protocol, then the protocol succeeds. (Correctness) If the protocol succeeds, then it generates a uniformly random value $c \in_R \mathcal{F}$, known to all players $P_j \in \mathcal{P}'$. (Complexity) The protocol communicates at most $(n^3 + n^2 - 2n)\kappa$ bits and requires at most $({}^1\!/\!{}_8 n^3 + {}^3\!/\!{}_8 n^2 + {}^3\!/\!{}_8 n - {}^5\!/\!{}_8)\kappa$ random bits.*

Proof (sketch). Completeness and complexity follow from inspecting the protocol. We now focus on the case when the protocol succeeds. There is at least one honest player P_h in $\{P_1, \ldots, P_{t'+1}\}$, who chooses his value c_h uniformly at random. As in Step 1, the adversary does not obtain any information about c_h (privacy of Share), and as the values c_i of *every* player $P_i \in \mathcal{P}'$ are fixed after Step 1 (Correctness of Share), c_h is statistically independent of all other values c_j $(j \neq i)$. Hence, the sum $c_1 + \ldots + c_{t'+1}$ is uniformly distributed. □

5.4 Generating one Sig-setup

Recall that a sig-setup for a designated signer P_S consists of the signing key $(p_0, \ldots, p_{n'+1}, q_0, \ldots, q_{n'+1})$, which should be random and known only to the signer P_S, and one verification key $(v_{i,1}, \ldots, v_{i,n'+1}, x_i, y_i)$ for each player $P_i \in \mathcal{P}'$, where the values $v_{i,1}, \ldots, v_{i,n'+1}$ should be random and known only to P_i,[4] and the values x_i and y_i are computed as $x_i = p_0 + \sum_{j=1}^{n'+1} p_j v_{i,j}$ and $y_i = q_0 + \sum_{j=1}^{n'+1} q_j v_{i,j}$, respectively. Table 1 summarizes the steps needed to compute these values.

Player	Inputs (rand.)			Intermediate (shared)			Outputs
P_S	p_0	$\cdots$	$p_{n'+1}$				
	q_0	$\cdots$	$q_{n'+1}$				
P_1	$v_{1,1}$	$\cdots$	$v_{1,n'+1}$	$p_1 v_{1,1}$	$\cdots\cdots$	$p_{n'+1} v_{1,n'+1}$	$x_1 = p_0 + \sum_k p_k v_{1,k}$
				$q_1 v_{1,1}$	$\cdots\cdots$	$q_{n'+1} v_{1,n'+1}$	$y_1 = q_0 + \sum_k q_k v_{1,k}$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots\cdots$
$P_{n'}$	$v_{n',1}$	$\cdots$	$v_{n',n'+1}$	$p_1 v_{n',1}$	$\cdots\cdots$	$p_{n'+1} v_{n',n'+1}$	$x_{n'} = p_0 + \sum_k p_k v_{n',k}$
				$q_1 v_{n',1}$	$\cdots\cdots$	$q_{n'+1} v_{n',n'+1}$	$y_{n'} = q_0 + \sum_k q_k v_{n',k}$

Table 1. Preparing one sig-setup

[4] The randomness of $v_{i,1}, \ldots, v_{i,n'+1}$ is needed for the sole reason of protecting the verifier P_i, hence it must be guaranteed for honest verifiers only.

In our protocol, first every player P_i chooses and secret-shares his verification key $(v_{i,1}, \ldots, v_{i,n'+1})$. Then, the players jointly generate three random vectors $(p_0, \ldots, p_{n'+1})$, $(q_0, \ldots, q_{n'+1})$, and $(r_0, \ldots, r_{n'+1})$. The first two of these vectors will serve as signing key, and the third will serve as blinding in the verification of the computation. Then, for each of these three vectors, the values $x_1, \ldots, x_{n'}$, respectively $y_1, \ldots, y_{n'}$ or $z_1, \ldots, z_{n'}$, are computed. This computation is not detectable: It might be that one of the x_i, y_i or z_i values is wrong, and still no honest player has detected a failure (however, when all players correctly follow the protocol, then all values will be correct). The correctness of these values is verified in an additional verification step: Two random challenges ρ and φ are generated, and the linearly combined (and blinded) signing vector $(\rho p_0 + \varphi q_0 + r_0, \ldots, \rho p_{n'+1} + \varphi q_{n'+1} + r_{n'+1})$ is computed, and (distributively) compared with the linearly combined verification keys. If all checks are successful, then (with overwhelming probability) all keys are correctly computed.

Protocol GenerateSignatureSetup**.**

1. *(Generate $v_{i,k}$-values.)* Every $P_i \in \mathcal{P}'$ selects random $v_{i,1}, \ldots, v_{i,n'+1}$ and invokes Share to share them.
2. *(Generate p_k-values.)* Invoke GenerateRandom $n' + 1$ times to obtain shared $p_0, \ldots, p_{n'+1}$.
3. *(Compute x_i-values.)* For every x_i, execute the following steps:
   3.1 Every $P_j \in \mathcal{P}'$ (locally) computes $c_{i,j} = \sum_{k=1}^{n'+1} \langle p_k \rangle_j \langle v_{i,k} \rangle_j$ and invokes Share to share it.
   3.2 The players compute $\langle x_i \rangle = \langle p_0 \rangle + \sum_{j=1}^{n'} \lambda_j \langle c_{i,j} \rangle$, where λ_j denotes the j-th Lagrange coefficient[5].
4. *(Generate q_k/y_i-values.)* Generate $(q_0, \ldots, q_{n'+1})$ and $(y_1, \ldots, y_{n'})$ along the lines of Steps 2–3.
5. *(Generate r_k/z_i-values.)* Generate $(r_0, \ldots, r_{n'+1})$ and $(z_1, \ldots, z_{n'})$ along the lines of Steps 2–3.
6. *(Check correctness of the computed x_i/y_i-values)*
   6.1 Invoke GenerateRandom twice to generate random challenges ρ and φ.
   6.2 For $k = 1, \ldots, n' + 1$, compute and reconstruct towards every player $\langle s_k \rangle = \rho \langle p_k \rangle + \varphi \langle q_k \rangle + \langle r_k \rangle$.
   6.3 For $i = 1, \ldots, n'$, compute $\langle w_i \rangle = s_0 + \sum_{k=1}^{n'+1} s_k \langle v_{i,k} \rangle$.
   6.4 For $i = 1, \ldots, n'$, compute $\langle \widetilde{w}_i \rangle = \rho \langle x_i \rangle + \varphi \langle y_i \rangle + \langle z_i \rangle$.
   6.5 For $i = 1, \ldots, n'$, reconstruct to every player $\langle d_i \rangle = \langle w_i \rangle - \langle \widetilde{w}_i \rangle$.
   6.6 Every P_j checks whether $d_i \stackrel{?}{=} 0$ for $i = 1, \ldots, n'$, and sets $\mathsf{fail}_j = \mathsf{true}$ in case of any non-zero value.
7. *(Announce x_i/y_i-values.)* For every $P_i \in \mathcal{P}'$, invoke Recons to reconstruct $\langle x_i \rangle$ and $\langle y_i \rangle$ towards P_i.

[5] The j-th Lagrange coefficient can be computed as $\lambda_j = \prod_{i=1, i \neq j}^{n'} \frac{-\alpha_i}{\alpha_j - \alpha_i}$.

Lemma 4. *The protocol* GenerateSignatureSetup *has the following properties: (Completeness) If all players in $\mathcal{P}'$ correctly follow the protocol, then the protocol succeeds. (Correctness) If the protocol succeeds, then (with overwhelming probability) it generates a correct signature setup. (Privacy) If the protocol succeeds, then no subset of t' players obtains any information they are not allowed to obtain. (Complexity) The protocol communicates at most* $(11n^4 + 4n^3 - 3n^2 - 13n)\kappa$ *bits and requires at most* $(2n^4 + 4n^3 + 2n^2 + 3)\kappa$ *random bits.*

Proof (sketch). (Completeness) We consider the case that all players follow the protocol, hence no sub-protocol fails. Observe that for every $i = 1, \ldots, n'$, the points $(\alpha_1, c_{i,1}), \ldots, (\alpha_{n'}, c_{i,n'})$ lie on a degree-$2t'$ polynomial $f_i(\cdot)$ with $f_i(0) = \sum_{k=1}^{n'+1} p_k v_{i,k}$. This polynomial is well defined because $n' > 2t'$, hence we can interpolate $f_i(0)$ with Lagrange's formula.[6] This interpolation is done distributively, i.e., every player P_j shares his $c_{i,j}$, then these sharings are combined using Lagrange's formula, and p_0 is distributively added, resulting in a sharing of $x_i = p_0 + \sum_{k=1}^{n'+1} p_k v_{i,k}$. Similarly, $y_i = q_0 + \sum_{k=1}^{n'+1} q_k v_{i,k}$ and $z_i = r_0 + \sum_{k=1}^{n'+1} r_k v_{i,k}$. Clearly, for any ρ and φ, $(\rho p_0 + \varphi q_0 + r_0) + \sum_{k=1}^{n'+1} (\rho p_k + \varphi q_k + r_k) v_{i,k} = \rho x_i + \varphi y_i + z_i$, hence $d_i = 0$, and no player detects a failure in Step 6.6.

(Correctness) We have to show that when the protocol succeeds, then for $i = 1, \ldots, n'$ holds $x_i = p_0 + \sum_{k=1}^{n'+1} p_k v_{i,k}$ and $y_i = p_0 + \sum_{k=1}^{n'+1} q_k v_{i,k}$. Observe that after Step 5, the values $v_{i,k}, p_k, q_k, r_k, x_i, y_i, z_i$ are fixed (they all are t'-shared). When x_i and y_i do no satisfy the required equation above, then only with negligible probability, for random ρ and φ they satisfy the equation $(\rho p_0 + \varphi q_0 + r_0) + \sum_{k=1}^{n'+1} (\rho p_k + \varphi q_k + r_k) v_{i,k} = \rho x_i + \varphi y_i + z_i$.

(Privacy) We have to show that when the protocol succeeds, every player learns only his respective key (plus some random data he could have generated himself with the same probability). First observe that in Steps 1–5, the only communication which takes place is by invocation of Share, which leaks no information to the adversary. In Step 6, the values $s_1, \ldots, s_{n'+1}$ and $d_1, \ldots, d_{n'}$ are reconstructed. Every value s_k is blinded with a random r_k (unknown to the adversary), so is uniformly random from the viewpoint of the adversary. The values d_i are either 0 (and hence the adversary can easily simulate them), or the protocol fails (and all computed values are discarded).

(Complexity) The complexity can be verified by inspecting the protocol. □

5.5 The Refresh-Protocol

In order to refresh a BA-setup, we need to generate two BA-setups, consuming only one BA-setup. Remember that one BA-setup consists of $2n'$ sig-setups (2 for every potential signer); hence, Refresh needs to generate $4n'$ sig-setups.

[6] Note that $f_i(0)$ is arbitrary when a single player is incorrect — something we do not care for when arguing about completeness.

Protocol Refresh.

0. $\forall P_i \in \mathcal{P}'$: set $\mathsf{fail}_i = \mathsf{false}$.
1. Invoke GenerateSignatureSetup $4n'$ times in parallel to generate 4 sig-setups for each signer $P_S \in \mathcal{P}'$.
2. $\forall P_i \in \mathcal{P}'$: Send fail_i to every $P_j \in \mathcal{P}'$.
3. $\forall P_j \in \mathcal{P}'$: Set $\mathsf{fail}_j = \mathsf{true}$ if any received bits $\mathsf{fail}_i = \mathsf{true}$.
4. Invoke Consensus' with P_j's input being fail_j. Denote the output as fail.
5. $\forall P_i \in \mathcal{P}'$: send fail to every $P_j \in (\mathcal{P} \setminus \mathcal{P}')$.
6. $\forall P_j \in (\mathcal{P} \setminus \mathcal{P}')$: Set fail as the majority of the received bits.

It is easy to see that Refresh fails when any GenerateSignatureSetup failed for an honest player. On the other hand, when all players follow the protocol, then Refresh succeeds. Refresh communicates $\mathcal{O}(n^5)\kappa$ bits.

5.6 Fault Handling

The following fault-handling procedure is invoked only when Refresh has failed (i.e., the players agree on $\mathsf{fail} = \mathsf{true}$). The goal of FaultHandling is to localize a set $E \in \mathcal{P}'$ of two players, such that (at least) one of them is faulty.

FaultHandling exploits the fact that there is no need to maintain the secrecy of the failed Refresh protocol. Basically, in FaultHandling the whole transcript of Refresh is revealed and there will be a message from some player P_i to some player P_j, where P_i claims to have sent some other message than P_j claims to have received — hence either P_i or P_j is lying, and we can set $E = \{P_i, P_j\}$. Unfortunately, it would be too expensive to publicly reveal the whole transcript; instead, the transcript is revealed towards a selected player (e.g. $P_k \in \mathcal{P}'$ with the smallest index k), who searches for the fault and announces it.

We stress that the considered transcript not only contains the messages of all invocations of the protocol GenerateSignatureSetup, but also the messages of the protocol Refresh. This is important because it might be that no fault occurred in GenerateSignatureSetup, but still some (corrupted) player P_i claimed to have $\mathsf{fail}_i = true$.

Protocol FaultHandling.

1. Every $P_i \in \mathcal{P}'$ sends to P_k all random values chosen during the course of the protocol Refresh (including all sub-protocols), as well as all values received during the course of Refresh.
2. P_k computes for every P_i the messages P_i should have sent (when being correct) during the course of Refresh; this can be done based on the random values and the received messages of P_i.
3. P_k searches for a message from some player $P_i \in \mathcal{P}'$ to some other player $P_j \in \mathcal{P}'$, where P_i should have sent a message x_i (according to his claimed randomness), but P_j claims to have received x_j, where $x_i \neq x_j$. Denote the index of this message by ℓ.
4. P_k invokes Broadcast to announce (i, j, ℓ, x_i, x_j).

5. P_i invokes Broadcast to announce whether he indeed sent x_i in the ℓ-th message.
6. P_j invokes Broadcast to announce whether he indeed received x_j in the ℓ-th message.
7. If Both P_i and P_j confirm to have sent x_i, respectively to have received x_j, then $E = \{P_i, P_j\}$. If P_i does not confirm to have sent x_i, then $E = \{P_k, P_i\}$. If P_j does not confirm to have received x_j, then $E = \{P_k, P_j\}$.

FaultHandling requires 3 BA invocations and communicates $O(n^5\kappa)$ bits.

6 Conclusions

We have presented a BA protocol for n players that achieves information-theoretic security against $t < n/2$ faulty players, communicating $\mathcal{O}(n^5\kappa)$ bits (for some security parameter κ). The protocol requires a compact constant-size setup, as all BA protocols that tolerate $t \geq n/3$ do (also those with cryptographic security only), and allows for polynomially many BA operations.

This result improves on the existential result of [PW96], which communicates $\mathcal{O}(n^{17}\kappa)$ bits per BA.

References

[Bea91] D. Beaver. Secure multiparty protocols and zero-knowledge proof systems tolerating a faulty minority. *Journal of Cryptology*, pp. 75–122, 1991.

[BGP92] P. Berman, J. A. Garay, and K. J. Perry. Bit optimal distributed consensus. *Computer Science Research*, pp. 313–322, 1992.

[BGW88] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proc. 20th STOC*, pp. 1–10, 1988.

[BH06] Z. Beerliova-Trubiniova and M. Hirt. Efficient multi-party computation with dispute control. In *TCC 2006*, LNCS 3876 , pp. 305–328, 2006.

[BPW91] B. Baum-Waidner, B. Pfitzmann, and M. Waidner. Unconditional Byzantine agreement with good majority. In *STACS '91*, LNCS 480, pp. 285–295, 1991.

[CDD$^+$99] R. Cramer, I. Damgård, S. Dziembowski, M. Hirt, and T. Rabin. Efficient multiparty computations secure against an adaptive adversary. In *EUROCRYPT '99*, LNCS 1592, pp. 311–326, 1999.

[CW79] L. Carter and M. N. Wegman. Universal classes of hash functions. *JCSS*, 18(4):143–154, 1979. Preliminary version in Proc. 9st STOC, 1977.

[CW92] B. A. Coan and J. L. Welch. Modular construction of a Byzantine agreement protocol with optimal message bit complexity. *Information and Computation*, 97(1):61–85, 1992.

[DS83] D. Dolev and H. R. Strong. Authenticated algorithms for Byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, Nov. 1983. Preliminary version in Proc. 14th STOC, 1982.

[Fit03] M. Fitzi. *Generalized Communication and Security Models in Byzantine Agreement.* PhD thesis, ETH Zurich, 2003.

[Fit04] M. Fitzi. Personal communication, 2004.

[FLM86] M. J. Fischer, N. A. Lynch, and M. Merritt. Easy impossibility proofs for distributed consensus problems. *Distributed Computing*, 1:26–39, 1986.

[HMP00] M. Hirt, U. Maurer, and B. Przydatek. Efficient secure multi-party computation. In *ASIACRYPT 2000*, LNCS 1976, pp. 143–161, 2000.

[PSL80] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, Apr. 1980.

[PW96] B. Pfitzmann and M. Waidner. Information-theoretic pseudosignatures and Byzantine agreement for $t >= n/3$. Technical report, IBM Research, 1996.

[RB89] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proc. 21st STOC*, pp. 73–85, 1989.

[Sha79] A. Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, 1979.

[SHZI02] J. Shikata, G. Hanaoka, Y. Zheng, and H. Imai. Security notions for unconditionally secure signature schemes. In *EUROCRYPT 2002*, LNCS 2332, pp. 434–449, 2002.

A Long Messages

The proposed BA protocols only capture messages $m \in \mathcal{F}$, i.e., κ-bit messages. In order to reach BA on longer messages, one could invoke the according BA protocol several times (once for every κ bit block). However, this would blow up the communication complexity unnecessarily high: BA of a $\ell\kappa$ bit message would require a communication complexity of $\mathcal{O}(\ell n^5 \kappa)$ bits (as opposed to $\mathcal{O}(\ell\kappa n^2 + n^{17}\kappa)$ in [PW96]). In this section, we sketch a construction that allows BA of a $\ell\kappa$ bit message at costs of $\mathcal{O}(\ell\kappa n^2 + n^5\kappa)$ bits.

In order to achieve the stated complexity, we need to replace the protocol Consensus' by $\mathsf{Consensus}_{\mathrm{long}}$'. The basic idea of $\mathsf{Consensus}_{\mathrm{long}}$' is straight forward: Every player $P_i \in \mathcal{P}'$ sends his message m_i to every other player. Then, the players use Consensus' to reach agreement on a universal hash value. If agreement is achieved, all players output the message with the agreed hash value, otherwise they output $\perp$. The key for the universal hash function is assumed to be pre-shared among the players as part of the BA-setup, and only reconstructed when needed. We also explain how this sharing is prepared in the Refresh protocol.

A.1 Protocol $\mathsf{Consensus}_{\mathrm{long}}$'

In the following, we present the protocol $\mathsf{Consensus}_{\mathrm{long}}$ among the players in $\mathcal{P}'$, reaching agreement on a $\ell\kappa$ bit message m. The protocol makes use of universal hashing [CW79]. As universal hash with key $k \in \mathcal{F}$, we use the function $U_k : \mathcal{F}^\ell \to \mathcal{F}, (m^{(1)}, \ldots, m^{(\ell)}) \mapsto m^{(1)} + m^{(2)}k + \ldots + m^{(\ell)}k^{\ell-1}$. The probability that two different messages map to the same hash value for a uniformly chosen key is at most $\ell/|\mathcal{F}|$, which is negligible in our setting with $\mathcal{F} = GF(2^\kappa)$.

Protocol $\mathsf{Consensus}_{\mathrm{long}}$'.

1. Every $P_i \in \mathcal{P}'$ sends his message m_i to every player $P_j \in \mathcal{P}'$.
2. The players reconstruct the random hash key $k \in \mathcal{F}$, which is part of the BA setup.

3. Every $P_i \in \mathcal{P}'$ computes (for his original message m_i) the universal hash $U_k(m_i)$.[7]
4. The players in $\mathcal{P}'$ invoke Consensus' to reach agreement on the hash value h.
5. If the above consensus fails (i.e., $h = \perp$), then every $P_i \in \mathcal{P}'$ outputs $\perp$. If it succeeds, then every $P_i \in \mathcal{P}'$ outputs that m_j received in Step 1 with $U_k(m_j) = h$.

One can easily see that the above protocol reaches consensus on m, and that it communicates $\mathcal{O}(\ell \kappa n^2)$ plus one invocations of Consensus', i.e., communicates $\mathcal{O}(\ell \kappa n^2 + n^4 \kappa)$ overall.

A.2 Generating the Hash Key

The protocol $\mathsf{Consensus}_{\mathrm{long}}$' needs a random hash key to be known to all players in $\mathcal{P}'$. We cannot afford to generate this hash key on-line (this would require several invocations of broadcast). Instead, we assume a robust sharing of a random field element to be part of every BA-setup. This sharing is then reconstructed when needed.

As robust sharing, we use the scheme of [CDD+99]. Essentially, this is a two-dimensional Shamir sharing, ameliorated with so called authentication tags. The sharing is constructed non-robustly; in the Share protocol, the players pairwisely check the consistency of the received shares, and fail in presence of faults. The sharing of the hash key is generated as sum of a sharing of each player in $\mathcal{P}'$. Such a sharing can be computed with communicating $\mathcal{O}(n^4\kappa)$ bits (and without involving broadcast). When the hash key is needed, then the sharing of the actual BA setup is reconstructed towards every player in $\mathcal{P}'$. This is achieved by having every player sending his shares (including the authentication tags) to every other player; this involves a communication of $\mathcal{O}(n^3\kappa)$ bits.

[7] In order to do so, the message m_i is split into blocks $m_i^{(1)}, \ldots, m_i^{(\ell)}$.