Towards Optimal Parallel Broadcast under a Dishonest Majority

Daniel Collins\textsuperscript{1⋆}, Sisi Duan\textsuperscript{2}, Julian Loss\textsuperscript{3}, Charalampos Papamanthou\textsuperscript{4}, Giorgos Tsimos\textsuperscript{5⋆⋆}, and Haochen Wang\textsuperscript{6}

\textsuperscript{1}Purdue University, colli594@purdue.edu
\textsuperscript{2}Tsinghua University, duansisi@mail.tsinghua.edu.cn
\textsuperscript{3}CISPA Helmholtz Center, lossjulian@gmail.com
\textsuperscript{4}Yale University, charalampos.papamanthou@yale.edu
\textsuperscript{5}University of Maryland, tsimos@umd.edu
\textsuperscript{6}Tsinghua University, whc20@mails.tsinghua.edu.cn

Abstract. The parallel broadcast (PBC) problem generalises the classic Byzantine broadcast problem to the setting where all \( n \) nodes broadcast a message and deliver \( O(n) \) messages. PBC arises naturally in many settings including multi-party computation. Recently, Tsimos, Loss, and Papamanthou (CRYPTO 2022) showed PBC protocols with improved communication, against an adaptive adversary who can corrupt all but a constant fraction \( \epsilon \) of nodes (i.e., \( f < (1-\epsilon)n \)). However, their study is limited to single-bit messages, and their protocols have large polynomial overhead in the security parameter \( \kappa \): their \textsc{TrustedPBC} protocol achieves \( \tilde{O}(n^2\kappa^4) \) communication and \( O(\kappa \log n) \) rounds. Since these factors of \( \kappa \) are in practice often close (or at least polynomially related) to \( n \), they add a significant overhead. In this work, we propose three parallel broadcast protocols for \( L \)-bit messages, for any size \( L \), that significantly improve the communication efficiency of the state-of-the-art.

We first propose a new extension protocol that uses a \( \kappa \)-bit PBC as a black box and achieves i) communication complexity of \( O(Ln^2 + P(\kappa)) \), where \( P(\kappa) \) is the communication complexity of the \( \kappa \)-bit PBC, and ii) round complexity same as the \( \kappa \)-bit PBC. By comparison, the state-of-the-art extension protocol for regular broadcast (Nayak et al., DISC 2020) incurs \( O(n) \) additional rounds of communication. Next, we propose a protocol that is secure against a static adversary, for \( \kappa \)-bit messages with \( \tilde{O}(n^2\kappa^2 + \kappa^3 + \kappa) \) communication and \( O(\kappa) \) round complexity, where \( K \) is an arbitrarily small constant such that \( 0 < K < 1 \). Finally, we propose an adaptively-secure protocol for \( \kappa \)-bit messages with \( \tilde{O}(n^2\kappa^2 + \kappa^3) \) communication overhead and \( O(\kappa \log n) \) round complexity by modifying and improving the next-best protocol \textsc{TrustedPBC} in several key ways. Notably, our latter two protocols are \( \tilde{O}(\kappa^2-K) \) and \( O(\kappa^2) \) times more communication-efficient, respectively, than the state-of-the-art protocols while achieving the same round complexity.

⋆ This work was completed while working at EPFL and visiting CISPA.
⋆⋆ Part of this work was completed while visiting CISPA.
Keywords: Byzantine broadcast, parallel Byzantine broadcast, improved communication complexity

1 Introduction

Byzantine broadcast (BC) is a fundamental primitive for many cryptographic protocols and distributed systems. The goal of BC is to allow a designated sender to distribute its input value such that all honest nodes output the same value, even if a fraction of Byzantine nodes (including, potentially, the sender) fail arbitrarily. In spite of a large body of work studying broadcast with a single sender, in many applications such as multi-party computation (MPC) and verifiable secret sharing (VSS) broadcast is most commonly required in parallel, i.e., with every sender broadcasting simultaneously.

Motivated by this observation, Tsimos, Loss, and Papamanthou [38] recently gave the first designated parallel broadcast (PBC) protocol, TrustedPBC. Denoting $n$ as the number of nodes and $\kappa$ as the length of a signature, TrustedPBC achieves $\widetilde{O}(n^2\kappa^4)$ communication against up to $f < (1 - \epsilon)n$ adaptive and malicious corruptions (for some $0 < \epsilon < 1$) under the assumption of a trusted PKI. Compared to naively running $n$ parallel BC instances, TrustedPBC sees substantial improvements in communication, mostly related to factors in $n$.

However, TrustedPBC cannot be considered practical: it works only for binary values and incurs an overhead of $\kappa^4$ over the optimal communication complexity of $\Omega(n^2)$ even for this limited use case. This is both a theoretical as well as a practical limitation. Indeed, it is often reasonable to think of $n$ and $\kappa$ as polynomially related. For example, if we conservatively pick $\kappa = 128$ and generously set the number of nodes to $n = 16384$, then the above comes out to $O(n^4)$ bits of communication complexity! In many practical cases, $n$ and $\kappa$ are even closer in size. Given this motivation, this work provides PBC protocols with significantly improved communication complexity over the state-of-the-art.

Contributions. In this work, we revisit the communication complexity of PBC with $L$-bit inputs in the synchronous setting assuming $f < (1 - \epsilon)n$ where $0 < \epsilon < 1$. The single-bit variants of our PBC protocols simply follow, which also enjoy improved communication. We consider both the static and weakly adaptive adversarial models (adaptive for short). As summarized in Table 1, we provide three protocols with improved communication. Our solutions do not trade factors of $\kappa$ for factors in $n$ and solely decrease factors in $\kappa$.

We begin with a new extension protocol for PBC, $\text{PBC}_L^*$, that reduces the $L$-bit PBC problem to a $\kappa$-bit PBC oracle. Compared to prior extension protocols, e.g., running $n$ BC instances by Nayak, Ren, Shi, Vaidya, and Xiang (NRSVX) [34], $\text{PBC}_L^*$ achieves both improved communication and round complexity. The adversarial assumption of $\text{PBC}_L^*$ depends on the underlying $\kappa$-bit PBC oracle. If the $\kappa$-bit PBC is adaptively secure, $\text{PBC}_L^*$ is adaptively secure.

We then present $\text{PBC}_{\kappa}^{\text{static}}$, a $\kappa$-bit PBC protocol in the static adversarial setting. $\text{PBC}_{\kappa}^{\text{static}}$ can in fact be generalized to $L$-bit PBC. However, using it
as a $\kappa$-bit PBC in our extension protocol will result in a more communication-efficient PBC. Compared to the state-of-the-art protocols BulletinBC [38] and FloodBC [10], it enjoys substantially improved communication complexity and the same or better round complexity. The core idea is to reduce the problem of PBC among $n$ nodes to $L$-bit PBC among a small committee of $\kappa$ nodes. Based on the most optimal constructions known so far for $L$-bit PBC, $\text{PBC}^{\text{static}}_\kappa$ achieves $\tilde{O}(n^2\kappa^1+K + n\kappa^3 + \kappa^4)$ communication and $O(\kappa)$ rounds for $\kappa$-bit broadcast, where $K$ is an arbitrarily small constant such that $0 < K < 1$.

Finally, we present $\text{PBC}^{\text{adaptive}}_\kappa$, a $\kappa$-bit PBC protocol secure under an adaptive adversary. Our starting point for building PBC under an adaptive adversary is TrustedPBC of Tsimos et al. [38], the most efficient 1-bit PBC protocol known so far that achieves $\tilde{O}(n^2\kappa^4)$ communication. We first construct a $\kappa$-bit PBC with $O((n^2\kappa^2 + n\kappa^3) \cdot \log^2 n)$ communication, more than a $O(\kappa^3)$ improvement over the communication complexity of TrustedPBC. We can use $\text{PBC}^{\text{adaptive}}_\kappa$ as a $\kappa$-bit PBC to our extension protocol we obtain a more communication-efficient

<table>
<thead>
<tr>
<th>$m$</th>
<th>Protocol</th>
<th>Model</th>
<th>Adv.</th>
<th>$f &lt;$</th>
<th>Communication</th>
<th>Rounds</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TrustedPBC [38]</td>
<td>trusted static</td>
<td>$(1-\epsilon)n$</td>
<td>$O(n^2\kappa^4)$</td>
<td>$O(\kappa)$</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>BulletinPBC [38]</td>
<td>trusted adaptive</td>
<td>$(1-\epsilon)n$</td>
<td>$O(n^2\kappa^4)$</td>
<td>$O(\kappa)$</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>TrustedPBC [38]</td>
<td>trusted static</td>
<td>$(1-\epsilon)n$</td>
<td>$O(n^2\kappa^4)$</td>
<td>$O(\kappa)$</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>TLP [38]</td>
<td>trusted adaptive</td>
<td>$(1-\epsilon)n$</td>
<td>$O(n^2L + \mathcal{P}(\kappa))$</td>
<td>$O(\kappa)$</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>NRSVX [34]</td>
<td>trusted adaptive</td>
<td>$(1-\epsilon)n$</td>
<td>$O(n^2L + \mathcal{P}(\kappa))$</td>
<td>$O(\kappa)$</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>PBC$^\text{static}_L$ (§3)</td>
<td>CRS+$\star$</td>
<td>$(1-\epsilon)n$</td>
<td>$O(n^2L + \mathcal{P}(\kappa))$</td>
<td>$O(\kappa)$</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>PBC$^\text{adaptive}_L$ (§3 + §4)</td>
<td>trusted static</td>
<td>$(1-\epsilon)n$</td>
<td>$O(n^2L + \mathcal{P}(\kappa))$</td>
<td>$O(\kappa)$</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>PBC$^\text{adaptive}_L$ (§3 + §5)</td>
<td>trusted adaptive</td>
<td>$(1-\epsilon)n$</td>
<td>$O(n^2L + C)$, $C=\tilde{O}(n^2\kappa^2 + n\kappa^3)$</td>
<td>$O(\kappa)$</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Comparison of the PBC protocols where honest nodes broadcast messages length $\leq |m|$. $\dagger$PBC that runs $n$ parallel instances. $\star$The assumption (bulletin board PKI, trusted PKI and/or common reference string (CRS)) and the adversarial model (static or adaptive) depend on the underlying $\kappa$-bit PBC oracle. $\mathcal{P}(x)$ is the communication complexity of $x$-bit PBC, and $\mathcal{T}(\kappa)$ is the round complexity of $\kappa$-bit PBC. $K$ is an arbitrarily small constant such that $0 < K < 1$. $\tilde{O}(f(n))$ indicates that the complexity of an algorithm is $O(f(n) \cdot \text{poly}(\log n))$ for some polynomial poly.


L-bit PBC. This is more than a $O(\kappa^3)$ improvement over the communication complexity of TrustedPBC (when used to broadcast $\kappa$ bit messages bit by bit). Similarly to $\text{PBC}^\text{static}_\kappa$, we can use $\text{PBC}^\text{adaptive}_\kappa$ as a $\kappa$-bit PBC to our extension protocol we obtain a more communication-efficient $L$-bit PBC.

1.1 Technical Overview

A new extension protocol for $L$-bit PBC, $\text{PBC}^*_L$ reduces $L$-bit PBC to a $\kappa$-bit PBC and uses an erasure coding proof (ECP) system, a notion due to Alhaddad, Duan, Varia, and Zhang [5]. ECP allows the encoder to prove succinctly and non-interactively that an erasure-coded fragment is consistent with a constant-sized commitment to the original data block. ECP works like an accumulator scheme for erasure coding but it has a feature that cannot be directly obtained from accumulators without an interactive protocol: ECP can be used to determine if a given fragment corresponds to the original data block. Leveraging this feature, our extension protocol achieves improved communication compared to prior extension protocols. Additionally, the round complexity remains essentially the same as the $\kappa$-bit PBC, incurring three extra rounds of communication. In contrast, the most communication-efficient extension protocol known so far (for BC) [34] incurs $O(n)$ rounds on top of the underlying $\kappa$-bit BC oracle.

$\kappa$-bit static PBC with $O(n^2\kappa^{1+K} + nk^3 + \kappa^4)$ communication. The state-of-the art PBC protocols rely on a small committee of $O(\kappa)$ randomly selected nodes to help decide whether some value should be output. A tempting solution is for the committee members can reach an agreement on some value (e.g., a bit in BC and $n$ bits in PBC) and then convey the results to all nodes. While this is feasible for a system in the honest majority setting [2, 28–30], there is no straightforward way to convey the results to non-committee nodes under a corrupt majority, an observation also been pointed out by prior works [16, 41]. In this work, we make the tempting solution work under a static adversary.

Our $\text{PBC}^\text{static}_\kappa$ protocol uses a reduction from PBC to a $\mathcal{C}()$ protocol among $\lambda = O(\kappa)$ committee members. $\mathcal{C}()$ has a useful property: after running the $\mathcal{C}()$ protocol, if an honest committee member sees some value for the first time, any other honest committee member also sees the value for the first time, i.e., honest committee members reach a certain level of agreement on their received values. An interesting finding is that we can use an $L$-bit PBC among $\lambda$ committee members to build the $\mathcal{C}()$ protocol. As the protocol is executed among the committee members, the $\mathcal{C}()$ protocol does not become the communication bottleneck.

Our protocol only incurs $O(\kappa^K)$ rounds treating the $\mathcal{C}()$ protocol as an oracle. Our insight is that prior works allow the signing committee to create their signatures in different rounds of the protocol. In contrast, as honest committee members in our protocol have access to $\mathcal{C}()$, either all honest committee members create signatures for some input bit or none of them creates a signature.

Via an application of the Chernoff bound, we show that a committee size of $\frac{\mu}{\mu^2 + 1} \log \frac{1}{\delta}$ allows the fraction of honest nodes in the committee to remain almost the same as the entire system with $1 - \text{negl}(\lambda)$ probability, where $\mu$ is
a small constant such that $0 < \mu < \epsilon$. As every node can always expect to receive matching signatures for honest committee members, it is not difficult to argue that the maximum number of rounds during which one can expect to collect signatures from all committee members is bounded by a constant. The round complexity of $PBC^\text{static}_\kappa$ is thus the same as the $C()$ protocol, i.e., $O(\kappa)$ for our construction. Additionally, $PBC^\text{static}_\kappa$ also enjoys improved communication compared to prior works, mostly because nodes only interact with the committee members. Using $PBC^\text{static}_\kappa$ in the framework of our extension protocol $PBC^*_L$, we obtain an $L$-bit PBC with $O(n^2 L + n^2 \kappa^{1+K} + n\kappa^3 + \kappa^4)$ communication.

**Communication-efficient $\kappa$-bit PBC under an adaptive adversary.** Borrowing from the ideas of TrustedPBC [38], we aim to construct a PBC protocol secure against adaptive adversaries, that allows for efficiently broadcasting messages from message spaces of any size. To recap, TrustedPBC is a committee-based protocol for PBC against an adaptive adversary. Each node acts as a sender and initially sends its message, signed, to all nodes. Then, through multiple rounds, nodes attempt to accept valid messages. At a high level, a message in round $r$ is valid if it is accompanied by $r$ many signatures from the committee defined for that message. During each round, nodes first forward valid messages by a call to a gossiping primitive called DISTINCTCONVERGE that allows for more communication-efficient message dissemination. In the second step of every round, nodes then check if they are in the committee for the respective message and if so, they add their own signature, (making the message valid for the next round,) and send the updated message and list of signatures to all. The total number of rounds for this protocol is related to the maximum committee size (and is $O(\kappa)$), and each round requires $O(\log n)$ many steps. The total communication of the protocol is $\tilde{O}(n^2 \cdot \kappa^4)$.

In order to construct a PBC protocol for messages of multiple bits (initially for $O(\kappa)$ and via the extension, for any size $L$) we first have to extend the committee election of TrustedPBC to account for multiple potential messages (we require one committee per message). We follow an idea similar as in Bacho et al. [6], where each message defines a separate committee among nodes, in the same way as in the binary case of [16]. This means however, that there can be as many potential committees as the size of the message space. Still, the key idea is that each node will only verify and forward up to two messages per sender; any sender who signs two or more signatures can only be corrupted, and the two messages alongside the signatures provide proof to every honest node. So, no dishonest sender can force the communication to increase by injecting a large number of valid messages in the protocol.

Second, we aim to reduce the communication of TrustedPBC by as many factors of $\kappa$ as possible. This proves far from trivial. We combine several techniques to achieve communication of $O((n^2 \cdot \kappa^2 + n \cdot \kappa^3) \cdot \log^2 n)$ for $\kappa$-bit PBC. This is more than a $O(\kappa^2)$ improvement over the communication of TrustedPBC. We now discuss the techniques employed for this communication improvement.

From $\tilde{O}(n^2 \kappa^4)$ to $\tilde{O}(n^2 \kappa^3)$. Via the constraint sets (as used in BULLETINPBC [38]) each node propagates a specific message—defined with respect to a message and
a sender– at most twice during the entirety of the protocol. Once a node receives a new valid message for the first time, it propagates it in the very next subround of Converge and adds it into the constraint set. In the next round, the node initiates a Converge with such newly received messages into its input set. After that, the node never propagates that specific message again.

However, this improvement by itself is not enough to remove a factor of κ from the communication. There is another step that blows up the communication, that being the construction of lists that are being forwarded at each step of DistinctConverge. We combat this by slightly modifying the way each node constructs the lists. Previously, each node constructed each list (i.e. set of messages to be forwarded to a specific node) by adding each message (from the set of messages to be forwarded) with some probability. The lists where then padded to a maximum, predetermined size that depended on the size of the set of messages to be forwarded. This step is needed to ensure that the same amount of (encrypted) bits are sent to each other node in a step of a protocol. In this manner, messages act as cover traffic for each other and thwart any adaptive advantage of an attacker. Instead, we propose a slightly more efficient sampling technique. Each node constructs the lists by similarly sampling the messages. If any list exceeds a predetermined size (which is approximately \( \kappa \times \text{size of set of messages to be propagated} / n \)), then the node resamples the list until it does not exceed that size. This can be done within \( O(n) \) many resamplings and guarantees that the lists are an even allocation of the bulk of messages to be propagated. Afterwards, the node pads all the lists similarly to that maximum size. The combination of these two improvements shaves off a factor of \( \kappa \) from the overall communication of the protocol.

From \( \tilde{O}(n^2\kappa^3) \) to \( \tilde{O}(n^2\kappa^2) \). The next improvement we propose is a more universal. We notice that the signatures’ batch sizes grow up to \( O(\kappa) \), where each signature is of size \( O(\kappa) \) as well. This leads to an unavoidable \( O(n^2 \cdot \kappa^3) \) communication over the PBC’s execution since at worst case, for at least \( f = O(n) \) senders, the entire \( O(\kappa) \)-sized committee will send to all \( n \) parties a batch of \( O(\kappa) \) many, \( O(\kappa) \)-sized signatures. We make use of aggregate signatures to lower the communication, making each batch a signature of size \( O(\kappa) \). Nodes can verify the aggregate signature with respect to the signers’ public keys. We also leverage recursive non-interactive zero-knowledge proofs (NIZKs) in order for our use of \( \mathcal{F}_{\text{mine}} \), the functionality we use to elect committees [2], to not bottleneck the communication complexity of the protocol.

1.2 Related Work

Round complexity of Byzantine broadcast. The celebrated work by Dolev and Strong [18] showed that in a synchronous system with \( n \) nodes, there exists an \( (f + 1) \)-round deterministic BC that tolerates up to \( f \) Byzantine nodes for \( f < n \). Additionally, \( f + 1 \) rounds (i.e., \( O(n) \) rounds) is optimal for deterministic BC protocols. Many follow-up works focus on lowering the round complexity of BC. Randomized protocols [7, 37] are found to be effective in overcoming the
lower bound. Feldman and Micali [21] showed a randomized BC protocol for $f < n/3$ achieving $O(1)$ round, assuming private channels only. In the authenticated setting, subsequent works [22,28] showed that $O(1)$ round can be achieved for $f < n/2$. Garay, Katz, Koo, and Ostrovsky [25] showed that for the corrupt majority setting, a randomized BC protocol achieves $\Theta(n/(n - f))$ round complexity. Fitzi and Nielsen [23] further improved the concrete number of rounds in the same setting. Wan, Xiao, Shi, and Devadas (WXSD) [41] presented a BC protocol that achieves $O((n/(n - f))^2)$ round complexity under the trusted setup assumption and weakly adaptive adversary. In another work, Wan, Xiao, Devadas, and Shi [40] presented a BC protocol that handles strongly adaptive adversary in $O(\kappa)$ rounds, where a strongly adaptive adversary can perform after-the-fact-removal.

**Byzantine agreement vs. Byzantine broadcast.** Byzantine agreement (BA) typically has two forms: Byzantine broadcast (BC) and Byzantine agreement (also called Byzantine consensus). In BC, a designated broadcaster sends an input value to the nodes and honest nodes output the same value. In Byzantine agreement, every node holds an input and honest nodes output the same value. BA with single-bit inputs is also called binary Byzantine agreement. In the synchronous setting, BC can be solved for $f < n$ and BA can be solved for $f < n/2$. Similar to that for BC, deterministic BA requires $O(n)$ rounds in the worst case and several works meet the bound assuming $f < n/3$ [8,20,26]. Momose and Ren [32] recently showed that $O(\kappa n^2)$ communication complexity and $f < n/2$ are possible in authenticated setting. In addition, randomized BA protocols can achieve sublinear rounds or even constant rounds [3,21,28].

**Scalable BA and BC.** Besides BC protocols we reviewed in the introduction, a line of work studies BA assuming a large $n$ in both synchronous setting [2,13,29] and asynchronous setting [11]. For instance, King and Saia studied BA in the synchronous setting and presented a BA protocol with $O(n^{1.5})$ communication complexity [29]. Abraham et al. [2] proposed recently binary BA with subquadratic communication complexity. In the asynchronous setting, Blum, Katz, Liu-Zhang and Loss [11] present a BA protocol achieving subquadratic communication complexity under an adaptive adversary assuming $f < (1 - \epsilon)n/3$.

**L-bit BA and BC.** BA with long input messages is also called multivalued Byzantine agreement (MBA). MBA can be reduced to binary agreement, both in the synchronous model and asynchronous model [17,33,39]. PBC with long input messages in the synchronous model is also known as interactive consistency [35]. Additionally, a line of research studies extension protocols for BC and BA to support L-bit inputs [9,24,31]. Most of these works focus on reducing the communication complexity compared to running L parallel BC or BA instances. A typical approach is to use erasure codes [27,36]. In this work, we use the BC protocol by Nayak, Ren, Shi, Vaidya, and Xiang (NRSVX) [34] and also propose a new extension protocol for PBC.
2 Preliminaries

2.1 Model

We consider a system with \( n \) nodes \( \{P_1, \cdots, P_n\} \), running over authenticated channels. Among the \( n \) nodes, \( f \) of them may become Byzantine and fail arbitrarily. We assume \( f < (1 - \epsilon)n \), where \( \epsilon \) is a constant and \( 0 < \epsilon < 1 \). Nodes that are not Byzantine are called honest. We consider a synchronous network, where there exists an upper bound on the network and message processing delay.

We consider both the static adversary model and the adaptive adversary model. In the static adversary model, the adversary corrupts nodes prior to the start of the protocol. In the adaptive adversary model, where the adversary can choose the set of corrupted replicas at any moment during the execution of the protocol based on the state it accumulated. In this work, we focus on the weakly adaptive adversary model, where the adversary cannot perform ”after-the-fact-removal” and retroactively erase the messages the replica sent before they become corrupted. Additionally, we assume atomic sends \[11\] where an honest node \( P_i \) can send to multiple nodes simultaneously, without the adversary being able to corrupt \( P_i \) in between sending to two nodes.

We assume a trusted setup unless otherwise specified, where a trusted party generates and distributes keys to the nodes prior to the protocol execution.

Let \( \kappa \) denote the cryptographic security parameter and \( \lambda \) the statistical parameter. In practice, \( \lambda = \mathcal{O}(\kappa) \) (typically \( \lambda < \kappa \)). So far, we used the same parameter \( \kappa \) to denote the maximum of these two values. When we discuss the concrete complexities in the main body of the paper, we differentiate \( \lambda \) and \( \kappa \).

We next recall the Chernoff bounds that we use in this work.

**Fact 1 (Chernoff Upper Tail Bound).** Suppose \( \{X_n\} \) is the independent \( \{0,1\}\)-random variables, and \( X = \sum_i X_i \). Then for any \( \tau > 0 \):

\[
\Pr \left( X \geq (1 + \tau)E(X) \right) \leq \exp \left( -\frac{\tau \cdot \min\{\tau, 1\} \cdot E(X)}{3} \right)
\]

2.2 Definitions

**Parallel broadcast (PBC).** In a system with \( n \) nodes \( \{P_1, \cdots, P_n\} \), PBC executes \( n \) parallel BC, where each node \( P_i \) provides an input \( v_i \) and outputs an \( n \)-value vector \( v_i \). Each slot \( s \) in \( v_i \) is dedicated for the value broadcast by \( P_s \), the output of which is denoted as \( v_i[s] \). In this work, we study PBC with both 1-bit inputs and \( L \)-bit inputs where \( L > 1 \).

**Definition 1 (\( f \)-Secure Parallel Broadcast).** Let \( \Pi \) be a protocol executed by nodes \( \{P_1, \cdots, P_n\} \), where each node \( P_i \) holds an input \( v_i \) and each node outputs a \( n \)-size vector \( v_i \). \( \Pi \) should achieve the following properties with probability \( 1 - \text{negl} (\kappa) \) whenever at most \( f \) nodes are corrupted.

-- **\( f \)-Validity:** If \( P_s \) is honest, the output \( v_i \) at any honest node \( P_i \) satisfies \( v_i[s] = v_s \).
− **Consistency**: All honest nodes output the same vector \( v' \).

We will need an *external validity* property for some of our constructions defined as follows. It is worth mentioning that the predicate \( Q \) is not necessarily a “function”. Instead, it may depend on the local state of the nodes [1, 19]. In this case, we may call the predicate a *locally validated predicate.*

− **External validity**: Given a predicate \( Q \), any honest node \( P_i \) that terminates outputs a value \( v_i \) such that for each \( v_i[s] \neq \perp, Q(v_i[s]) \) holds by at least one honest node.

**Protocol naming convention** \( \text{PBC}_y^x \). To differentiate the protocols we study in this paper, we use the \( \text{PBC}_y^x \) to denote a PBC protocol where each node provides an \( x \)-size input that is secure under \( y \) model. For example, \( \text{PBC}_{1}^{\text{static}} \) denotes a 1-bit PBC assuming a static adversary.

2.3 Building Blocks

**The \( F_{\text{mine}} \) oracle.** We follow prior work [2, 16, 38] and define the \( F_{\text{mine}} \) ideal functionality that we use for random committee selection. \( F_{\text{mine}} \) is parameterised by the total number of nodes and a *mining* probability \( p_{\text{mine}} \). The functionality provides two interfaces: \( F_{\text{mine}} \) and \( F_{\text{mine}.\text{verify}}() \). In particular, a node \( P_i \) can query \( F_{\text{mine}} \) to check whether it is an eligible member of the committee. The query of the \( F_{\text{mine}} \) function is also called a *mining* attempt. Upon receiving a mining attempt for the first time, \( F_{\text{mine}} \) flips a random coin and returns a binary result. It returns 1 with mining probability \( p_{\text{mine}} \). If 1 is returned, \( P_i \) is part of the committee. After \( P_i \) has successfully made a mining attempt, \( F_{\text{mine}} \) returns the same answer for all future identical queries.

In our work, we use three different types of committee: a committee in the static adversary model, a *signing* committee, and a *forwarding* committee. To differentiate the mining attempt and verification of them, we define the input to \( F_{\text{mine}} \) and \( F_{\text{mine}.\text{verify}}() \) in the form of \((\text{type}, \text{val}, i)\) where \( \text{val} \) might consist of multiple values and \( i \) is the identity of the node that queries the function. We present in Figure 1 the functionality of \( F_{\text{mine}} \). \( F_{\text{mine}} \) can be implemented with (concretely efficient) non-interactive zero-knowledge proofs of size \( O(\kappa) \) as shown in [2]. The use of \( F_{\text{mine}} \) will not incur any additional asymptotic communication overhead for our protocols.

**Aggregate signatures.** An aggregate signature scheme (generalising a *multi-signature* scheme) can aggregate \( S \) signatures into one signature, therefore reducing the size of signatures. Given \( S \) signatures \( \sigma_i = \text{Sign}(sk_i, m) \) on the same message \( m \) with corresponding public keys \( pk_i \) for \( 1 \leq i \leq S \), a multi-signature scheme can combine the \( S \) signatures above into one signature \( \Sigma \) where \( |\Sigma| = |\sigma_i| \). The combined signature can be verified by anyone using a verification function \( \text{Ver}(PK, \Sigma, m, L) \), where \( L \) is the list of signers and \( PK \) is the union of \( S \) public keys \( pk_i \). Moreover, signatures that are themselves combined signatures can be aggregated recursively/iteratively in the same fashion, which we assume is possible even when the intersection of the set of signers is non-empty.
Initialization:
- Mining probability $p_{\text{mine}}$.
- Let $\text{call}_i \leftarrow \bot$ for any $i \in [n]$.

On input $F_{\text{mine}}(\text{type}, \text{val}, i)$ from node $P_i$:
- If $\text{call}_i = \bot$, output $b = 1$ with probability $p_{\text{mine}}$, or $b = 0$ with probability $1 - p_{\text{mine}}$ and set $\text{call}_i = b$.
- Else output $\text{call}_i$.

On input $F_{\text{mine}}.\text{verify}(\text{type}, \text{val}, j)$ from node $P_i$:
- If $\text{call}_j = 1$, output 1, otherwise output 0.

Fig. 1: Functionality $F_{\text{mine}}$. $\text{val}$ can be $\bot$ or consists of multiple values.

By leveraging the PKI and associating public keys with their indices, alongside the arity of the signature, a signature signed by $S$ nodes can be represented in either $O(\kappa + S \log n)$ (i.e., using $\log n$ bits per node) or $O(\kappa + n)$ bits (using a bitmask). We assume they are unforgeable in an ideal sense in this work but in practice an appropriate unforgeability notion suffices.

**Erasure codes.** An $(m, n)$ erasure coding scheme over a data block $M$ is specified by two algorithms ($\text{encode}, \text{decode}$). The $\text{encode}$ algorithm takes as input $m$ data fragments of $M$, and outputs $n > m$ coded fragments. The $\text{decode}$ algorithm takes as input any $m$-size subset coded fragments and outputs the original data block containing $m$ data fragments. Namely, if $d \leftarrow \text{encode}(M)$ and $d = [d_1, \ldots, d_n]$, then $\text{decode}(d_{i_1}, \ldots, d_{i_m}) = M$ for any distinct $i_1, \ldots, i_m \in [1..n]$.

**Erasure coding proof (ECP) system.** Erasure coding proof (ECP) system is a notion introduced by Alhaddad, Duan, Varia, and Zhang [5]. The idea is to allow the encoder to prove succinctly and noninteractively that an erasure-coded fragment is consistent with a constant-sized commitment to the original data block. Consider an $(m, n)$ erasure code that encodes a message $M$ into a set of $n$ fragments $d_1, d_2, \cdots, d_n$. ECP system is designed to allow for efficient dispersal of these fragments. A proof contains two parts: a constant-sized commitment $\phi$ plus a per-node witness $\pi_1$ that is about as long as $d_i$ (namely, about $|M|/m$). Together, $\phi$ and $\pi_i$ convince node $P_i$ that $d_i$ is the correct data fragment for the message committed to by $\phi$. That is, reconstruction from any subset of $m$ valid fragments corresponding to the same commitment $\phi$ would lead to the same original message $M$. An ECP system consists of three algorithms.

- $\text{ecsetup}$. The $\text{ecsetup}$ algorithm receives a security parameter $\kappa$ and sets up the system parameters $pp$.
- $\text{ecprove}_{pp}$. The $\text{ecprove}_{pp}$ algorithm takes as input a block of data $M$ and outputs $(\phi, d, \pi)$ where $|d| = |\pi| = n$. Here, $\phi$ is a (computationally) binding commitment to all erasure-coded fragments $d \leftarrow \text{encode}(M)$, and each $\pi_i$ is intended to serve as a proof that the corresponding $d_i$ is the $i$-th data fragment with respect to the commitment $\phi$. 

---

10
The \texttt{ecverify} algorithm takes as input \((\phi, d_i, \pi_i)\) and outputs a bit. If \(\text{ecverify}_{pp}(\phi, d_i, \pi_i) = 1\), then we say \(d_i\) is a valid fragment with respect to \(\phi\).

**Definition 2 (Secure ECP System).** An ECP system as specified above is secure if it satisfies the following properties with probability \(1 - \negl(\kappa)\).

- **EC-Correctness:** If an honest encoder runs \(\text{ecprove}_{pp}(M, pp)\) and obtains \((\phi, d, \pi)\) and an honest decoder reconstructs \(M'\) from a set of \(m\) valid fragments \((d_i, \pi_i)\) with respect to \(\phi\), then \(M = M'\).
- **EC-Consistency:** If an honest decoder reconstructs \(M_1\) from a set of \(m\) valid fragments with respect to \(\phi\) and another honest decoder reconstructs \(M_2\) from a set of \(m\) valid fragments with respect to \(\phi\), then \(M_1 = M_2\).

We use ECP-1 in this work. Under the trusted setup assumption, the size of the witness has the same length as each data fragment.

**Forward-secure public-key encryption (FS-PKE).** A forward-secure public-key encryption scheme [15], or FS-PKE, is a probabilistic public-key cryptosystem that additionally allows the secret key to be updated such that previous keys and encrypted plaintexts cannot be derived from an updated key. For simplicity, we consider unbounded FS-PKE, where an arbitrary number of secret key update operations, each forming a different epoch, are supported (we nonetheless only require bounded FS-PKE where the number of updates is a priori bounded). Note that the public key is fixed at key generation time. A FS-PKE consists of four algorithms.

- \(\text{fspkegen}\). The \(\text{fspkegen}\) key generation algorithm takes as input a security parameter \(\kappa\) and outputs a public/secret key pair \((pk, sk_0)\), where \(sk_0\) is associated with epoch 0.
- \(\text{fspkeenc}\). The \(\text{fspkeenc}\) encryption algorithm takes as input \((pk, i, m)\), a public key, epoch \(i \geq 0\) (associated with secret key \(sk_i\)) and message \(m\), and outputs a ciphertext \(ct\).
- \(\text{fspkedec}\). The \(\text{fspkedec}\) decryption algorithm takes as input \((pk, i, sk_i, ct)\), a public key \(pk\), epoch \(i\) associated with secret key \(sk_i\) and a ciphertext \(ct\), and outputs a message \(m\) (or \(m = \bot\) if decryption fails).
- \(\text{fspkeupd}\). The \(\text{fspkeupd}\) secret key update algorithm takes as input \((pk, i, sk_j)\), a public key \(pk\), an epoch \(i\) and a secret key \(sk_j\) associated with epoch \(j < i\), and outputs an epoch \(i\) secret key \(sk_i\).

**Definition 3 (Secure FS-PKE).** A FS-PKE as specified above is secure if it satisfies the following properties with probability \(1 - \negl(\kappa)\).

- **FS-PKE-Correctness:** For any \((pk, sk_0) \leftarrow \text{fspkegen}\), any well-formed \(sk_i\) (i.e., output by iterative calls to \(\text{fspkeupd}\) starting with \(sk_0\)) for epoch \(i\) and any message \(m\), we have \(m = \text{fspkedec}(pk, i, sk_i, \text{fspkeenc}(pk, i, m))\).
- **FS-PKE-IND-CCA:** Given a challenge oracle \(\text{chal}(j, m_0, m_1)\) that encrypts \(m_b\) for bit \(b\) under epoch \(j\) given \(|m_0| = |m_1|\), the ability to expose secret keys for epoch \(i > j\) and a decryption oracle for all ciphertexts but the challenge, an adversary cannot distinguish between the case of \(b = 0\) and \(b = 1\).
An appropriate formally-specified IND-CCA security notion can be found in [15].

We assume FS-PKE has constant-sized ciphertexts (in the security parameter), which can be achieved by using the hierarchical identity-based encryption scheme of [12] as a binary-tree encryption scheme in the construction of [15].

3  PBC$^*_L$: An Extension Protocol for L-bit PBC

In this section, we present a new extension protocol for L-bit PBC, utilizing a $\kappa$-bit PBC as an oracle. We use PBC$^*_\kappa$ to denote the oracle. Our extension protocol is secure under an adaptive adversary, as long as PBC$^*_\kappa$ is adaptively secure. Additionally, the round complexity of our protocol is the same as PBC$^*_\kappa$.

The only thing we require is transforming a $\kappa$-bit PBC protocol into a validated PBC by adding a locally validated predicate to PBC$^*_\kappa$. The same paradigm can be extended to obtain a communication-efficient L-bit extension protocol for BC as well.

3.1 The Extension Protocol

The pseudocode of our extension protocol is shown in Figure 2. There are three phases: dissemination, agreement, and reconstruction. In the dissemination phase, each node $P_i$ first sends its input $M_i$ to all nodes. To do so, it queries ecprove$_{pp}(M_i)$ and then uses the commitment $\phi_i$ as the input to PBC$^*_\kappa$. $P_i$ then enters the agreement phase. In the agreement phase, we query the PBC$^*_\kappa$ protocol to agree on the commitment. Here, we transform PBC$^*_\kappa$ into a validated PBC and additionally require every node to check one locally validated predicate $Q$ for each input $\phi_i$: $Q(\phi_i)$ holds if a node has received $M_i$ from $P_i$. In this way, we ensure that if PBC$^*_\kappa$ completes, at least one honest node holds $L_i$.

After the PBC$^*_\kappa$ outputs some value, the reconstruction phase involves two communication rounds. Here we consider the output value $\phi_j$ for slot $j$ and the process for other slots is the same. In the first round, if $P_i$ has previously received the correct value $M_j$ from $P_j$, it then queries ECP and obtains $n$ fragments $d$ and witness $\pi$. Then for each $P_j$, $P_i$ sends a (SEND, $d_i$, $\pi_i$) message to it. In the second round, every node $P_i$ for a valid fragment $d_i$. If $P_i$ receives the fragment, it fixes its $d^*_i$ as $d_i$ and then forwards to all nodes. Finally, after receiving $n - f$ valid fragments, $P_i$ decodes the fragments into $M_j$ and adds $M_j$ to its output.

3.2 Proof of Correctness and Complexity

Lemma 1. The PBC$^*_\kappa$ protocol with predicate $Q$ satisfies $f$-external validity.

Proof. As we use PBC$^*_\kappa$ as a black box, we prove the lemma without looking into the concrete construction. Namely, towards a contradiction, assume that an honest node outputs $v_i$ such that $Q(v_i[s])$ does not hold for any honest node for some slot $s$. This only holds if none of honest replicas have accepted $v_i[s]$. 

Global Parameters:
- \( M_i \) is the input of \( P_i \).
- \( \text{ExtractedSet}_i \leftarrow [\emptyset] \).

Phase 1:
- Every node \( P_i \) performs the following:
  - \( \text{ExtractedSet}_i \leftarrow M_i \).
  - Send \((\text{Disseminate}, M_i)\) to all nodes.
  - \( (\phi_i, d_i, \pi_i) \leftarrow \text{ecprove}_{pp}(L_i) \).
  - Query \( \text{PBC}_L^* \) with predicate \( Q \) and use \( \phi_i \) as input.

Phase 2:
- Upon output \( \phi_j \) for slot \( j \) in \( \text{PBC}_L^* \):
  - If \( P_i \) has previously received \((\text{Disseminate}, M_j)\) from \( P_j \)
    - \( (\phi', d, \pi) \leftarrow \text{ecprove}_{pp}(M_j) \).
    - If \( \phi' = \phi_j \), for \( t = 1, 2, \ldots, n \)
      - Send \((\text{Send}, d_t, \pi_t)\) to \( P_t \).
  - Upon receiving \((\text{Send}, d_i, \pi_i)\) from \( P_i \),
    - If \( \text{ecverify}_{pp}(\phi_j, d_i, \pi_i) = 1 \),
      - Fix \( d^*_i \) as \( d_i \) and send \((\text{Echo}, d^*_i, \pi_i)\) to all nodes.
  - Upon receiving \((\text{Echo}, d^*_i, \pi_i)\) from \( P_i \),
    - If \( \text{ecverify}_{pp}(\phi_j, d^*_i, \pi_i) = 1 \), \( B_j \).add(\( d^*_i \)).
    - Upon \(|B_j| \geq n - f\):
      - \( M_j \leftarrow \text{decode}(B_j) \)
      - \( \text{ExtractedSet}_j \leftarrow M_j \)
    - Output \( \text{ExtractedSet}_j \).

Fig. 2: The PBC\(_L^*\) protocol. \( Q \) is the locally validated predicate defined as follows:
\( Q(\phi_i) \) is valid for a node \( P_j \) if \( P_j \) has previously received \( M_i \) from \( P_i \) such that for \( (\phi, d, \pi) \leftarrow \text{ecprove}(M_i) \) it holds that \( \phi_i = \phi \).

i.e., the values from corrupt replicas are sufficient to build a secure PBC. This violates the validity property of PBC for the case where \( P_s \) is honest.

**Theorem 1.** Assuming CRS, the PBC\(_L^*\) protocol presented in Figure 2 satisfies \( f \)-validity and \( f \)-consistency with probability \( 1 - \text{negl}(\lambda) \).

**Proof.** \( f \)-Validity. Since \( P_i \) is honest, it invokes \( \text{ecprove}_{pp}(M_i) \) and outputs \((\phi_i, d, \pi)\). Then every honest node \( P_j \) will receive \( M_i \) and the locally validated predicate \( Q \) for PBC\(_L^*\) will be satisfied by every honest node. Additionally, it is not difficult to see that every honest node outputs \( \phi_i \) for the \( i \)-th slot for PBC\(_L^*\) as otherwise the EC-correctness property of ECP is violated. According to the protocol, every honest node sends a fragment to all nodes in the “send” round and eventually receives \( n - f \) fragments. Then every honest node is able to reconstruct \( M_i \), as otherwise the EC-consistency property is violated.

\( f \)-Consistency: We assume that for a slot \( s \in [n] \), an honest node \( P_i \) outputs \( M_1 \) and another honest node \( P_j \) outputs \( M_2 \) such that \( M_1 \neq M_2 \) and prove the
correctness by contradiction. If \( P_1 \) holds \( M_1 \), it must have output \( \phi_1 \) for slot \( s \) in \( \text{PBC}_\kappa \) such that \((\phi_1, d, \pi) \leftarrow \text{ecprove}_{pp}(M_1)\). In the following, we first show that if \( \text{PBC}_\kappa^* \) outputs a commitment \( \phi_1 \), then at least one honest node \( P_k \) has received message \( M_1 \) from \( P_s \). Then we show that if another honest node \( P_j \) outputs \( M_2 \), \( M_1 = M_2 \).

We begin with the first statement. According Lemma 1, \( Q(\phi_1) \) holds for at least one honest node. Therefore, the honest node has received \( M_1 \).

We then show the second statement. Here, there are two cases: \( P_j \) receives \( M_2 \) from \( P_s \); \( P_j \) does not receive any value from \( P_s \) and outputs \( M_2 \) after it receives \( n - f \) fragments. For the first case, if \( M_2 \neq M_1 \), the commitment of \( M_2 \) is \( \phi_2 \neq \phi_1 \), so either the \( f \)-consistency property of PBC is violated or the EC-correctness property of ECP is violated. For the second case, we already know that an honest node \( P_k \) holds \( M_1 \), so \( P_k \) will broadcast a fragment to each node. According to the EC-correctness property of ECP, every honest node \( P_\ell \) is able to fix its \( d_\ell \) and then sends a message \((\text{Echo}, d_\ell, \pi_\ell)\) to all nodes. Accordingly, node \( P_j \) will receive \( n - f \) valid fragments and reconstruct the message \( M_2 \). If \( M_2 \neq M_1 \), the EC-consistency property is violated.

**Theorem 2.** The \( \text{PBC}_L^* \) protocol achieves \( O(n^2 L + \mathcal{P}(\kappa)) \) communication and the round complexity is asymptotically the same as \( \text{PBC}_\kappa^* \), where \( \mathcal{P}(\kappa) \) is the communication complexity of \( \text{PBC}_\kappa^* \).

## 4 \( \text{PBC}_\kappa^\text{static} \): Communication-Efficient PBC under a Static Adversary

\( \text{PBC}_\kappa^\text{static} \) is a two-layer protocol that reduces the PBC problem to best effort broadcast and a \( \mathcal{C}(\cdot) \) protocol among \( \lambda \) committee members. Although \( \text{PBC}_\kappa^\text{static} \) itself can clearly generalized to an \( L \)-bit PBC, when the protocol is used as a \( \kappa \)-bit PBC and integrated with our extension protocol presented above, we obtain a more communication-efficient \( L \)-bit PBC. In this section, we present the \( \mathcal{C}(\cdot) \) protocol that we use as a building block for our the concrete construction of \( \text{PBC}_\kappa^\text{static} \) that we present after.

### 4.1 The \( \mathcal{C}(\cdot) \) Protocol

As mentioned in the introduction, the \( \mathcal{C}(\cdot) \) protocol achieves agreement among committee members. After running the \( \mathcal{C}(\cdot) \) protocol, if an honest committee member sees some value for the first time, any other honest committee member also sees the value for the first time. We consider the following scenario: each committee member \( P_i \) receives \( M_j \) from each node \( P_j \), where \( j \in [n] = \{1, 2, \ldots, n\} \) and \( M_j \) is an \( n \)-value vector in the form of \([M_1^j, \ldots, M_n^j]\). Each \( M_k^j \) is either \( \bot \) or consists of up to two valid \( (r - 1) \)-s batches. To facilitate the exposition of our protocol, we first provide some definitions.
Definition 4. (Valid r-s batch). A valid r-s batch on a message/slot pair \((u, s)\) for (some round) \(r \geq 0\) is in the form of \(u||s||\text{SIG}_e\), where \(u \in \{0, 1\}^L\), \(s \in [n]\), and \(\text{SIG}_e\) is a set of signatures that contains one signature from \(P_i\) and \(3r(\epsilon - \mu)(1 - \epsilon) \log \frac{1}{\delta}\) signatures on \([u, s]\) from members in the committee, where \(\mu\) is a small constant such that \(0 < \mu < \epsilon\) and \(\delta\) is the desired failure probability.

Definition 5 (Valid tuple for round \(r\)). A valid tuple \(M_i\) for round \(r \geq 1\) is in the form of \([M_i^1, \cdots, M_i^n]\) where each \(M_i^j\) is either \(\perp\), or consists of at most two valid \((r - 1)-s\) batches, one for a pair \((u, j)\) and one for a pair \((u', j)\).

Definition 6. (Part-of relationship). Given two valid tuples \(M_i\) and \(M_j\), \(M_i\) is part of \(M_j\) if the following conditions are satisfied: For any \(l \in [n]\), if \(rs \in M_i^l\) where \(rs\) is a valid r-s batch on \([u, l]\), then \(rs' \in M_j^l\), where \(rs'\) is a valid r-s batch on \([u, l]\). In addition, any signature in \(rs\) is also included in \(rs'\).

The part-of relationship is transitive: if \(M_i\) is part of \(M_j\) and \(M_j\) is part of \(M_k\), then \(M_i\) is part of \(M_k\).

We now specify the input and output of the \(C()\) protocol as follows. The protocol is executed among \(c\) nodes, among which at most \(t\) are corrupt. To allow honest nodes to share the same view about the messages they receive, the input of \(C()\) needs to be validated and the output needs to be verifiable [14]. In some round \(r\), the input of each node \(P_i\) for the \(C()\) protocol is \(M\) which consists of up to \(n\) vectors \(\{M_1, \cdots, M_n\}\). Any \(M_j \in M\) is sent by node \(P_j\). Each \(M_j\) is validated if it is a valid tuple for round \(r\). After running the \(C()\) protocol, each honest committee member \(P_i\) outputs an \(n\)-value vector \(\text{Merged}_i\). \(\text{Merged}\), is verified if it is a valid tuple for round \(r\). We further consider that the total number of messages provided by any node for each slot \(s\) is bounded by a constant, i.e., \(\sum_{j=1}^n M_j^s\) is a constant number.

Definition 7 (t-Secure \(C()\)). Let \(C()\) be a protocol executed by \(c\) nodes \(\{P_1, \cdots, P_c\}\), as specified above. \(C()\) should satisfy the following properties with probability \(1 - \negl(\kappa)\) whenever at most \(t\) nodes are corrupted.

- t-Validity: If an honest node \(P_i\) provides \(M\) as input, any valid tuple \(M_j \in M\) for \(r\) is part of \(\text{Merged}_k\) for any honest node \(P_k\).
- t-Consistency: For each slot \(s \in [n]\), if an honest node \(P_i\) outputs \(\text{Merged}_i^s\), another honest node \(P_j\) outputs \(\text{Merged}_j^s\), \(\text{Merged}_i^s = \text{Merged}_j^s\).

The workflow. As our goal is essentially for all honest committee members to share the same view of the received valid \((r-1)-s\) batches, an interesting observation is that the \(C()\) protocol can be reduced to a PBC protocol with \(L\)-bit inputs among \(c\) committee members, denoted as \(\text{PBC}_{L,c}^*\). Namely, each node aggregates its input into a valid tuple, and then broadcasts it via \(\text{PBC}_{L,c}^*\). After completing \(\text{PBC}_{L,c}^*\), each node aggregates the outputs into a valid tuple.

We present a construction of \(C(M)\) in Figure 3. Upon \(C(M)\), each node \(P_i\) first filters the vectors that can not be validated. Then \(P_i\) compiles a union of \(M\) into a valid tuple \(\text{Aggregated}_i\) for \(r\). Namely, for each slot \(s \in [n]\), if any node \(P_j\)
\( \mathcal{C}(\cdot) \)

**Input:** Round \( r \), \( M = \{M_1, \ldots, M_n\} \), for each \( M_j \in M \), \( M_j = [M_1^j, \ldots, M_n^j] \).

**Output:** An \( n \)-value vector \( \text{Merged}_i \).

**Upon \( \mathcal{C}(M) \)**
- Filter any \( M_j \) such that \( M_j \) is not a valid tuple for round \( r \).
- For each slot \( s \in \{n\}:
  - \( \text{Aggregated}_s^j \leftarrow \cup_{j=1}^n M_s^j \).
  - Provide \( \text{Aggregated}_s \) as input to a \( L \)-bit PBC \( \text{PBC}^*_L,c \).
  - Wait until \( \text{PBC}^*_L,c \) completes, let the output be \( m \).
- For each slot \( s \in \{n\} \) and for each valid tuple \( m_j \) for round \( r \) that \( m_j \in m \):
  - \( \text{Merged}_s^j \leftarrow \cup_j m_s^j \).

**Output conditions**
- After \( \text{PBC}^*_L,c \) terminates, return \( \text{Merged}_i \).

![Fig. 3: The \( \mathcal{C}(\cdot) \) protocol.](image)

provides a valid tuple \( M_j \) for \( r \), \( P_i \) compiles a union of \( M_j^s \) for any \( j \in \{n\} \) and updates \( \text{Aggregated}_s^j \). After this procedure, \( P_i \) holds a valid tuple \( \text{Aggregated}_s^j \) for \( r \). Then, \( P_i \) provides \( \text{Aggregated}_s^j \) as input to \( \text{PBC}^*_L,c \).

Let \( m \) denote the set of outputs of \( \text{PBC}^*_L,c \). Node \( P_i \) compiles a union of all valid tuples in \( m \). Namely, for any valid tuple \( m_j \), \( P_i \) sets its \( \text{Merged}_s^j \) as the union of \( m_s^j \) for \( s \in \{n\} \). Finally, \( P_i \) outputs \( \text{Merged}_i \) and the \( \mathcal{C}(\cdot) \) protocol completes.

As we use \( \text{PBC}^*_L,c \) as a sub-protocol for \( \mathcal{C}(\cdot) \), we need a committee size that meets the requirement for \( \text{PBC}^*_L,c \). As we assume \( f < (1-\epsilon)n \) in our work and the upper bound any PBC protocol can achieve is \( f < n \) (i.e., [18]), it is natural to consider a committee size \( c \) such that the number of corrupt committee members is bounded by \( t < (1-\epsilon+\mu)c \), where \( \mu \) is a small constant such that \( 0 < \mu < \epsilon \). As we show later in Lemma 2, the committee size can be set as \( \frac{3(1-\epsilon)}{\mu^2} \ln \frac{1}{\delta} = O(\kappa) \) to satisfy the requirement. The number of signatures required for a valid \( r \)-s batch for each \( r \) is then provided.

**Lemma 2.** Let \( \alpha \) denote the fraction of Byzantine nodes in the entire system, i.e. \( \alpha = 1 - \epsilon \), where \( \epsilon \in (0, 1) \). Then for any small constant \( \mu \) such that \( 0 < \mu < \epsilon \), if the number of the nodes in the committee is greater than \( \frac{3\alpha}{\mu^2} \ln \frac{1}{\delta} \), then the number of Byzantine nodes in the committee is less than \( (1-\epsilon+\mu)c \) with probability \( 1 - \negl(\lambda) \).

**Corollary 1.** If the number of the nodes in the committee is greater than \( \frac{3\alpha}{\mu^2} \ln \frac{1}{\delta} \), the number of honest nodes in the committee is greater than \( (\epsilon - \mu)c \) with probability \( 1 - \negl(\lambda) \).

**Example 1.** Let \( \mu = \frac{\epsilon}{2} \), if the number of the nodes in the committee is greater than \( \frac{12\alpha}{\epsilon^2} \ln \frac{1}{\delta} \), the number of honest nodes in the committee is greater than \( \frac{\epsilon^2}{2} = \frac{6(1-\epsilon)}{\epsilon} \ln \frac{1}{\delta} \) with probability \( 1 - \negl(\lambda) \).
Lemma 3. Consider a committee with c nodes, among which fewer than t = (1−ε+μ)c are faulty. An L-bit PBC protocol satisfies t-validity and t-consistency properties of PBC.

Proof. In a synchronous system, the upper bound for t and c for L-bit BC is t < c [18]. As t = (1−ε+μ)c < c, an L-bit PBC protocol satisfies t-validity and t-consistency properties of PBC.

Theorem 3. The C() protocol presented in Figure 3 satisfies t-validity and t-consistency.

Theorem 4. Let the length of each Mi ∈ M be L and c = λ, the communication complexity and the round complexity of the C() protocol is the same as a L-bit PBC among c committee members, i.e., PBC^L,c.

To build the L-bit PBC, we can use the extension protocol by Nayak, Ren, Shi, Vaidya, and Xiang (NRSVX) [34]. NRSVX reduces L-bit BC to κ-bit BC. If we use Dolev-Strong as the κ-bit BC, the communication complexity of C() is O(Lκ^3 + κλ^3 + λ^4) and the round complexity is O(λ). If we use our extension protocol mentioned in §3, the C() protocol achieves O(Lκ + κλ^3 + λ^4) communication and O(λ) rounds. This is because the κ-bit BC oracle is the bottleneck.

4.2 The PBC^static Protocol

Based on the C() protocol, we are now ready to present PBC^static. The protocol is round-based, starting from round 0 to round R where R = ⌈(1−ε+μ)c−1/(ε−μ)c⌉ = O(1/μ), i.e., a constant number.

State. Each node Pi has a value ui as input to the protocol. Each node also maintains two global parameters: ExtractedSet = [ExtractedSet_1, · · · , ExtractedSet_n] and VotedSet = [VotedSet_1, · · · , VotedSet_n], used to store the received and voted values. ExtractedSet_i and VotedSet_i denote the values for slot s where s ∈ [n]. The two global maps are accumulative, i.e., they are not cleared throughout the protocol. In each round r ≥ 1, every node also maintains Received_i = [Received_1, · · · , Received_i] and Merged_i = [Merged_1, · · · , Merged_i] to keep track of the received valid (r−1)-s batches. The Received_i and Merged_i are not accumulative, i.e., they are local parameters for each round.

The workflow. We present the workflow of PBC^static in Figure 4. In round 0, each node Pi puts its input ui in ExtractedSet_i, creates a digital signature of σ_i for [ui, i], and sends a (SIGN, ui, σ_i) message to all nodes. Meanwhile, each node queries the F_mine(static, i) function to discover whether it belongs to the committee. At the end of round 0, all honest nodes are aware of the identities of all honest committee members.

From round r = 1 to r = R, each round r consists of three mini-rounds. In the first mini-round, each node Pi sends an (Echo, Received_i) message to all committee members. If r = 1, Received_i is the aggregated n-value vector obtained from the (SIGN) messages. If r > 1, Received_i is the aggregated n-value
Global Parameters:
- Let $u_i$ be the input of $P_i$.
- Let $\epsilon$ be the fraction of honest nodes.
- Let $\delta$ be the desired failure probability.
- Let $\mu$ be a small positive constant such that $0 < \mu < \epsilon$.
- Let $R = \lceil \frac{1 - \epsilon + \mu}{\epsilon \mu} \rceil$ be the total number of rounds.
- Let $\text{ExtractedSet}_i \leftarrow [0]^n$, $\text{VotedSet}_i \leftarrow [0]^n$.

**Round 0:**
Every node $P_i$ performs the following:
- $\text{ExtractedSet}_i \leftarrow u_i$.
- Send $(\text{SIGN}, u_i, \sigma_i)$ to all nodes where $\sigma_i$ is a signature on $[u_i, i]$.
- Upon receiving a $(\text{SIGN}, u_j, \sigma_j)$ from $P_j$, add $\sigma_j$ to $\text{Received}_j$.
- Query $b \leftarrow \mathcal{F}_{\text{mine}}(\text{static}, i)$, if $b = 1$, broadcast $(\text{COM}, i)$ to all nodes.
- Upon receiving a valid $(\text{COM}, j)$ such that $\mathcal{F}_{\text{mine}}.\text{verify}(\text{static}, j)=1$, add $P_j$ to committee.

**Round $r = 1, \cdots, R$:** each round has three mini-rounds.
In the first mini-round, every node $P_i$ performs the following:
- Send $(\text{ECHO}, \text{Received}_i)$ to all committee members, where $\text{Received}_i$ is obtained from round $r - 1$.
- Upon receiving $(\text{ECHO}, \text{Received}_j)$ from $P_j$, $M[j] \leftarrow \text{Received}_j$.
In the second mini-round:
- Every committee member runs $C(M)$ and obtains $\text{Merged}_i$.
In the third mini-round, set $\text{Received}_i \leftarrow [\bot]^n$, for each $s \in [n]$:
Every committee member $P_i$ performs the following:
- Send $(\text{SEND}, \text{Merged}_i)$ to all nodes.
- If $\text{Merged}_i^s$ is non-empty and a valid $(r - 1)$-s batch on $[u_i^s, s]$ is included in $\text{Merged}_i^s$ but $u_i^s \notin \text{VotedSet}_i^s$:
  - Set $\text{VotedSet}_i^s \leftarrow \text{VotedSet}_i^s \cup u_i^s$, create a signature for $[u_i^s, s]$ and send to all nodes.
- If at least two valid $(r - 1)$-s batches on different pairs $[u_i^s, s]$ and $[v_i^s, s]$ are included in $\text{Merged}_i^s$ and $u_i^s, v_i^s \notin \text{VotedSet}_i^s$:
  - Send the first two of valid $(r - 1)$-s batches on different pairs $[u_i^s, s]$ and $[v_i^s, s]$ to all nodes.
For every node $P_i$, upon receiving $(\text{SEND}, \text{Merged}_i)$ and signatures:
- Merge the received $(\text{SEND})$ messages and signatures into an $n$-value vector $\text{Received}_i$, s.t. each non-empty $\text{Received}_i^s$ contains at most two valid $r$-s batches.
- If there exists any $u_i^s$ such that a valid $r$-s batch for $[u_i^s, s]$ is included in $\text{Received}_i^s$ and $u_i^s \notin \text{ExtractedSet}_i^s$ and $|\text{ExtractedSet}_i^s| < 2$, $\text{ExtractedSet}_i^s \leftarrow \text{ExtractedSet}_i^s \cup u_i^s$.

**Output conditions**
- At the end of round $R$, for each slot $s \in [n]$:
  (Event 1) If $|\text{ExtractedSet}_i^s| = 1$ and $\text{ExtractedSet}_i^s = \{u\}$, $v_i[s] \leftarrow u$.
  (Event 2) If $|\text{ExtractedSet}_i^s| = 0$ or $2$, $v_i[s] \leftarrow \bot$.
- Output $v_i$.

Fig. 4: The $\text{PBC}_{\text{static}}^\kappa$ protocol.
vector obtained from the union of the (Send) message and signatures signed by committee members in round $r - 1$.

The second mini-round is executed only by committee members. At the end of the first mini-round, each committee member $P_i$ receives $M$, which consists of up to $n$ $\text{Received}_j$ for $j \in [n]$. Then $P_i$ executes the $C(M)$ protocol and outputs $\text{Merged}_i$, an $n$-value vector according to the specification in Figure 3. According to the $C()$ protocol discussed in §4.1, $\text{Merged}_i$ is a valid tuple that consists of a vector of valid $(r - 1)$-s batches received in the first mini-round.

In the third mini-round, each committee member $P_i$ first sends a $(\text{Send, Merged}_i)$ message to all nodes. Meanwhile, $P_i$ iterates the $(r - 1)$-s batches in $\text{Merged}_i$. If there exists a valid $(r - 1)$-s batch on $[u^*_i, s]$ in any $\text{Merged}_i$ and $P_i$ has not previously created a signature for $[u^*_i, s]$ (i.e., $u^*_i$ is not included in $\text{VotedSet}_i$), $P_i$ creates a digital signature for $[u^*_i, s]$ and sends to all nodes. Note that a Byzantine node can send different values to different committee members in the first mini-round. To ensure that the communication complexity of the committee members does not grow in scenarios like this, we also require that each honest committee member only sends at most two values for each slot. Two conflicting signatures from the same sender $P_s$ serves as an evidence that $P_s$ equivocate. Every honest node that receives the evidence will output ⊥ for slot $s$.

After receiving the $(\text{Send})$ messages and the digital signatures, each node $P_i$ first compiles the union of signatures from the $(\text{Send})$ messages. If $P_i$ obtains a valid $r$-s batch $rs$ for any value, it adds $rs$ to $\text{Received}_i$. Additionally, $P_i$ also verifies whether it has collected any valid $r$-s batch on $[u^*_i, s]$ and whether the size of $\text{ExtractedSet}_i$ is lower than two. If so, it adds $u^*_i$ to $\text{ExtractedSet}_i$.

At the end of round $R$, each node $P_i$ is ready to output. For each slot $s \in [n]$, if there is only one value in $\text{ExtractedSet}_i$, $P_i$ sets $v_i[s] = \text{ExtractedSet}_i$. Otherwise, $P_i$ sets $v_i[s]$ as a default symbol ⊥. Then $P_i$ outputs the vector $v_i$.

**Optimizing the 1st mini-round via StaticPropagate().** We optimize the first mini-round via a sampling protocol, denoted as $\text{StaticPropagate}()$. We use sampling for each node to propagate its $\text{Received}_i$ value to all committee members. Our optimization ensures that if an honest node holds a message $M_i$, at least one honest committee member receives $M_i$ at the end of the first mini-round.

To generalize the function, we use $\text{StaticPropagate}(M_i)$ to denote the function, where $M_i$ is a set of messages queried by $P_i$ and $|M_i| = n$. The pseudocode of $\text{StaticPropagate}(M_i)$ is shown in Figure 5. The $\text{StaticPropagate}(M_i)$ protocol has $\lambda K$ rounds, where $0 < K < 1$ is an arbitrarily small positive constant. In each round, node $P_i$ first samples $n/\lambda$ messages from $M_i$ and then sends the messages to the committee members.

**Lemma 4.** If an honest node $P_i$ provides $M_i$ as input to $\text{StaticPropagate}()$, the probability that none of honest committee members receive $M_i$ is $\text{negl}((\lambda))$.

**Proof.** If none of honest committee members received $M_i$ at the end of $\text{StaticPropagate}()$, there exists at least one message $x \in M_i$ such that none of honest committee members received $x$. According to Corollary 1, the number of honest committee members in the committee $d$ satisfies $d \geq (\epsilon - \mu)c_i$. Let $X^i$
**StaticPropagate**($M_i$)

**Input:** A set of messages $M_i$.

**Output:** A set of messages $O_i$.

For round $r = 1, \ldots, \lambda^K$:

- For all $x \in M_i$:
  - For any node $P_j$ in the committee:
    - Add $x$ to list $L_j$ with probability $1/\lambda$.
  - For any node $P_j$ in the committee:
    - Send $L_j$ to $P_j$.
  - For a committee member $P_i$, upon receiving a valid list $L_j$ from $P_j$:
    - $O_i \leftarrow O_i \cup L_j$

Output

- Return the set of received messages $O_i$.

---

**Fig. 5:** The static propagation process.

denote the event that the $i$-th honest committee member has not received $x$ by the end of round $\lambda^K$. The events $X^1, X^2, \ldots, X^d$ are independent since $P_i$ adds $x$ to lists $L_1, L_2, \ldots, L_d$ independently. Therefore, the probability that none of the honest committee members received $x$ is:

$$(1 - \frac{\lambda}{n}) \cdot \Pr(X^1X^2\cdots X^d) \leq \Pr(X^1X^2\cdots X^d)$$

$$= \Pr(X^1) \cdot \Pr(X^2) \cdots \Pr(X^d)$$

$$\leq \Pr(X^1) \cdot \Pr(X^2) \cdots \Pr(X^{(\epsilon-\mu)c})$$

$$= \left(1 - \frac{1}{\lambda}\right)^{\lambda^K (\epsilon-\mu)c} \leq e^{- (\epsilon-\mu)\lambda^K} = \text{negl}(\lambda).$$

**Lemma 5.** Let $|M|$ upper bound the length of the input message of each honest node. The **StaticPropagate**() protocol achieves $O(n|M|\log \lambda)$ communication.

**Optimizing the 3rd mini-round using erasure coding and a matching rule.** We optimize the communication of the third mini-round using ECP. The subtle challenge we address here is that even if we use ECP for the outputs of all committee members, we cannot lower the communication compared to the existing protocol. To see why, let the length of each committee member’s message be $|M|$. The communication of the third mini-round is $n\lambda|M|$, as $\lambda$ committee members send messages to $n$ nodes. If we use an $(f + 1, n)$ ECP scheme, each committee member can first disseminate the data fragments and then all nodes exchange their data fragments. As the nodes need to exchange the fragments for $\lambda$ messages, the communication complexity is $O(n\lambda|M| + n^2\lambda\kappa)$.

We provide a more communication-efficient version in Figure 6. The idea is that since $C()$ already makes honest committee members reach an agreement on
Replace the processing of \((\text{Send}, \text{Merged})\) in Figure 4 as follows:
- Every committee member \(P_i\) performs the following:
  - \((\phi_i, d_i, \pi_i) \leftarrow \text{ecprove}_{pp}(\text{Merged}_i)\).
  - For all \(j \in [n]\):
    - Send \((\text{Send}, \phi_i, d_j, \pi_j)\) to \(P_j\).
- For every node \(P_i\), upon receiving \(c(\epsilon - \mu)\) matching \((\text{Send}, \phi, d_i, \pi_i)\) messages such that \(\text{ecverify}_{pp}(\phi, d_i, \pi_i) = 1\)
  - Send \((\text{Echo}, \phi, d_i, \pi_i)\) to all nodes.
  - Upon receiving \((\text{Echo}, \phi, d_j, \pi_j)\) from \(P_j\)
    - If \(\text{ecverify}_{pp}(\phi, d_j, \pi_j) = 1\), \(B_{\phi}.\text{add}(d_j)\).
  - Upon \(|B_{\phi}| \geq n - f\):
    - \(M \leftarrow \text{decode}(B_{\phi})\).
    - \(\text{Received}_i \leftarrow \text{Received}_i \cup M\).

Fig. 6: A communication-efficient instantiation for the third mini-round.

the output, we can also group the fragments so nodes do not have to exchange so many fragments. As shown in Figure 6, we only need to modify the process for the \((\text{Send})\) message with a two-round protocol. First, every committee member \(P_i\) applies an \((f + 1, n)\) ECP scheme on \(\text{Received}_i\) and sends a data fragment to each node. For every node \(P_i\) in the system, upon receiving \(c(\epsilon - \mu)\) matching fragments, \(P_i\) forwards the fragments to all nodes. Finally, each node waits for \(n - f\) valid fragments and then decodes the messages. Other procedures of the third mini-round remain the same as those shown in Figure 4.

**Lemma 6.** Given the protocol in Figure 6, if all honest committee members send a message \(M\), the \(\text{Received}_i\) value by any honest node \(P_i\) at the end of the protocol satisfies \(M \subseteq \text{Received}_i\).

**Proof.** We first show that \(P_i\) will set its \(\text{Received}_i\) as some value. As honest committee members send \(M\), it is not difficult to see that every honest node in the system receives \(c(\epsilon - \mu)\) matching data fragments for \(M\) and then sends to all nodes in the \((\text{Echo})\) round. Therefore, every node receives \(n - f\) data fragments and decodes them into some value and updates \(\text{Received}_i\).

We now show that \(M \subseteq \text{Received}_i\). As honest committee members send the fragments for \(M\), any honest node is able to decode the fragments and include them in \(\text{Received}_i\), as otherwise the EC-correctness property of ECP is violated.

**Lemma 7.** Given the protocol in Figure 6, if every committee members sends a message \(M\) to all nodes, the protocol in Figure 6 achieves \(O(n|M| + \lambda|M| + n\lambda\kappa + n^2\kappa)\) communication.

**Proof.** In the \((\text{Send})\) round, \(\lambda\) nodes send a message to all nodes. In the \((\text{Echo})\) round, every node sends an \((\text{Echo})\) message upon receiving \(c(\epsilon - \mu)\) matching messages. Therefore, every node sends at most \(\frac{c}{\epsilon - \mu}\) \((\text{Echo})\) messages. The communication of the protocol is thus \(O\left(\lambda n\left(\frac{|M|}{f + 1} + \kappa\right) + \frac{1}{\epsilon - \mu} n^2\left(\frac{|M|}{f + 1} + \kappa\right)\right) = O(n|M| + \lambda|M| + n\lambda\kappa + n^2\kappa)\).
Discussion. We briefly discuss why the number of rounds $R$ is a constant. Specifically, all nodes send their received $(r-1)$-s batches to the committee members in the first mini-round. In the second mini-round, committee members exchange their received values. The $t$-consistency property of $C$ protocol guarantees that all honest committee members maintain the same Merged vector. Furthermore, the $t$-validity property and Lemma 4 together guarantee that if one honest committee member receives a valid $(r-1)$-s batch on $[b,s]$ from any node in the first mini-round, all honest committee members include $rs$ in their Merged at the end of the second mini-round. Furthermore, by Lemma 6, if an honest committee member $P_i$ sees a valid $(r-1)$-s batch $rs$ for the first time in the third mini-round, every honest committee member also sees $rs$ for the first time. Recall that the committee has $c$ nodes, among which $t$ are faulty. Therefore, in every round, every node can expect to receive $c-t \geq (\epsilon-\mu)c$ matching signatures instead of only one! As there are $c$ committee members in total and $\epsilon$ and $\mu$ are constants, the protocol can complete in $R = \mathcal{O}(1)$ rounds.

Theorem 5. Assuming a trusted PKI and CRS, $\text{PBC}^\kappa_{\text{static}}$ is an $f$-Secure Parallel Broadcast protocol with probability $1-\text{negl}(\lambda)$.

Theorem 6. Assuming a trusted PKI and CRS, the $\text{PBC}^\kappa_{\text{static}}$ protocol has $\mathcal{O}(\lambda)$ round complexity and $\mathcal{O}(n^2\kappa\lambda^4 + n\kappa\lambda^2 + \kappa\lambda^3 + \lambda^4)$ communication complexity.

5 $\text{PBC}^\kappa_{\text{adaptive}}$: Communication-Efficient PBC under an Adaptive Adversary

In this section, we present an adaptively-secure PBC protocol that, for $\kappa$-sized messages, has a communication complexity of $\mathcal{O}(n^2\kappa\lambda^4 + n\kappa\lambda^2 + \kappa\lambda^3 + \lambda^4)$, i.e., $\mathcal{O}(n^2\kappa^4)$ for single-bit PBC. By direct application of our adaptive extension protocol $\text{PBC}^\kappa_{\text{L}}$ from Section 3, we obtain an $L$-bit protocol with a communication complexity of $O(n^2L + C)$ bits where $C = \mathcal{O}(n^2\kappa^2 + n\kappa^3)$.

5.1 $\mathcal{M}$-DistinctConverge

Following Tsimos et al. [38], we define the $\mathcal{M}$-DistinctConverge problem. Later, we use our $\mathcal{M}$-DistinctConverge protocol to build PBC. In $\mathcal{M}$-DistinctConverge for set $\mathcal{M}$, honest nodes begin with a set of messages $M_i \subseteq \mathcal{M}$ and a constraint set $C_i \subseteq \mathcal{M}$. At the end of the protocol, all honest nodes output a superset of the difference between the union of all sets $M_j$ input by honest nodes and the union of all constraint sets $C_j$ input by honest nodes. Intuitively, this allows us to build a constrained ‘all-to-all’ multicast functionality that helps to achieve PBC.

Definition 8 (distinct function). For any set $M$, $\text{distinct}_k(M)$ is a subset of $M$ that contains all messages in $M$ with distinct $k$-bit prefixes.
For example, for \( M = \{01001, 01111, 11000, 10000\} \) we have that \( \text{distinct}_2(M) = \{01001, 11000, 10000\} \). Note that \( \text{distinct}_k \) is an one-to-many function, e.g., \( \text{distinct}_2(M) \) is also \( \{01111, 11000, 10000\} \).

**Definition 9 (t-secure \( M \)-DistinctConverge protocol).** Let \( M \subseteq \{0,1\}^\ast \) be an efficiently recognizable set. A protocol \( \Pi \) executed by \( n \) nodes, where every honest node \( P_i \) initially holds input set \( M_i \subseteq M \) and constraint set \( C_i \subseteq M \), is a t-secure \( M \)-DistinctConverge protocol if all remaining honest nodes upon termination, with probability \( 1 - \text{negl}(\kappa) \), output a set

\[
S_i = \text{distinct}_k \left( \bigcup_{P \in \mathcal{H}} M_i - \bigcup_{P \in \mathcal{H}} C_i \right),
\]

when at most \( t \) nodes are corrupted and where \( \mathcal{H} \) is the set of honest nodes at the beginning of the protocol.

**Propagate protocol.** We first present a sub-protocol called Propagate. In the work of Tsimos et al. [38], Propagate is abstracted into an ideal functionality \( F_{\text{prop}} \), but here we present the protocol and prove properties directly that we later use to prove security for our \( M \)-DistinctConverge protocol.

The aim of Propagate is to capture one step of all-to-all gossip. The protocol needs to protect against an adaptive adversary who tries to e.g., prevent one message from being received by any honest node by observing all sent messages.

Tsimos et al. solve this in TrustedPBCFor a set of messages \( M_i \), they create lists of messages for each node and include each message from \( M_i \) in \( O(\lambda) \) lists, ensuring that at least one honest node receives each message except with negligible probability. To prevent the adversary from corrupting nodes during this process causing to block some message, each list is 1) encrypted under a fresh public key sampled by all honest nodes and 2) padded to be of the same length.

Intuitively, this prevents the adversary from learning anything about who received what from observing messages alone since we assume atomic sends (i.e, the adversary cannot corrupt while a node is sending to several nodes at once) and from stopping communication (since we assume messages cannot be taken back when sent by previously honest nodes). Concretely, suppose node \( P_i \) has input message \( M \). Without encryption, the adversary could corrupt the \( O(\kappa) \) recipients of \( M \) and prevent it from reaching all parties.

To improve communication complexity, our Propagate protocol captures several of the key improvements to TrustedPBC described in the introduction. In TrustedPBC, all lists are padded to size \( A_p = \lceil 2m|M_i|/n \rceil \) for input set \( M_i \) (where \( |M_i| \) denotes the cardinality of \( M_i \)). This ensures that, except with negligible probability, all lists will be the same size, which results in an overall communication complexity in [38] of \( O(m \cdot \max\{n, |M_i|\} \cdot s) \) since \( A_p \) is at least of size \( O(\lambda) \) independent of the size of \( M_i \).

Our improved protocol achieves \( O(m \cdot |M_i| \cdot s) \) communication complexity. To do so, we instead set \( A_p = 2m|M_i|/n \). In doing this, we can no longer rely on the lists being bounded in size by \( A_p \). Therefore, we make some changes to the protocol (and provide new analysis) to accommodate for it:
For a sufficiently small set of messages ($|M_i| \leq \log n$), the caller $P_i$ will simply multicast their set of messages to all nodes.

Otherwise ($|M_i| > \log n$) the caller will sample lists of the form $L_j$, one for each node $P_j$.

- For $\log n < |M_i| \leq (n \log n)/\lambda$, if $|L_j| > A_p$, resample $L_j$. We prove this happens a polynomial amount of times except with negligible probability.
- For larger $|M_i|$, we show except with negligible probability padding to $2m|M_i|/n$ is sufficient for all lists.

Finally, note that Propagate is a one-round protocol, which is a reduction in half from the protocol of Tsimos et al. [38]. We achieve this by using forward-secure public-key encryption (FS-PKE) instead of regular PKE to encrypt messages. Namely, instead of sampling a new public/secret key pair at the beginning of each invocation of Propagate, nodes simply (non-interactively) update their FS-PKE secret key at the end of each Propagate call.

**Lemma 8.** If $M$ is the input set of node $P$ in Propagate, the size of each list $L_j$ is:

\[
\begin{cases}
O(\log n), & \text{if } |M| < \log n \\
\leq 2m|M|/n, & \text{else}
\end{cases}
\]

with probability $1 - \text{negl}((\lambda))$ if $m = \Theta(\lambda)$.

**Lemma 9.** Let $s$ be the length in bits of each message in $M$ for node $P$. Then, the communication complexity of $P$ during Propagate is with prob. $1 - \text{negl}(\lambda)$:

\[
\begin{cases}
O(n \log n \cdot s), & \text{if } |M| < \log n \\
O(m \cdot |M| \cdot s), & \text{else}
\end{cases}
\]

We use an idealized functionality of Propagate, namely $F_{\text{prop}}$, which is a slightly modified version of the same-named functionality from [38].

**Lemma 10.** Assuming a FS-PKE scheme (per Definition 3), Propagate is a secure instantiation of the $F_{\text{prop}}$ functionality.

### 5.2 Implementing $\mathcal{M}$-DistinctConverge

We provide an implementation of $\mathcal{M}$-DistinctConverge in Figure 9, which is the same as $\mathcal{M}$-DistinctCV presented in [38, Fig. 7]. As described above, nodes input a set of messages $M_i$ and a constraint set $C_i$. Running for $O(\log n)$ rounds, in each round, nodes first call Propagate with $M_i \setminus C_i$ as input, which outputs a set $O_i$. The output is appended to a set Local$_i$. $M_i$ is then added to the constraint set, since there is no need for the caller to propagate them again, and then the set of messages $M_i$ is set to Local$_i \cap \mathcal{M}$. Ultimately, the set $M_i$ is returned.
Propagate()

**Input:** A set of messages $M_i$.

**Output:** A set of messages $O_i$.

**Global Parameters:**
- PK is a set of FS-PKE public keys by the caller protocol.
- $sk_i$ is a FS-PKE secret key inherited by the caller protocol.

Upon Propagate($M_i$)
- If $|M_i| \leq \log n$:
  - For $\ell \in [n]$:
    - Send (Noenc, $M_i$) to $P_\ell$.
  - Else:
    - Let $A_p = 2m|M_i|/n$.
    - For $j \in [n]$:
      - For $x \in M_i$:
        - Add $x$ to list $L_j$ with probability $m/n$.
      - If $|M_p| \leq n\log n/\lambda$ and $|L_j| > A_p$:
        - $L_j \leftarrow \perp$
        - $j \leftarrow j - 1$ // resample $L_j$.
    - For $j \in [n]$:
      - Pad $L_j$ to size $A_p$.
      - If $(pk, j) \in PK$: $ct_j \leftarrow \text{fspkeenc}(pk, e, L_j)$.
      - Erase $L_j$ from memory.
      - If $(pk, j) \in PK$: Send $ct_j$ to $P_j$.

Upon $\Delta$ time after invoking Propagate($M_i$):
- For all $ct_j$ received from $P_j$:
  - $L_j \leftarrow \text{fspkedec}(pk, e, sk_c, ct)$
  - If $L_j \neq \perp$: Add $L_j$ to $O_i$.
  - $sk_{c+1} \leftarrow \text{fspkeupd}(pk_i, e + 1, sk_c)$.
  - Erase $sk_c$ from memory.
- For all (Noenc, $M_j$) received from $P_j$ s.t. $ct_j$ was not received from $P_j$:
  - Add $M_j$ to $O_i$.
- Output $O_i$.

Fig. 7: Propagate, an instantiation of the propagation process.

**Lemma 11.** DistinctCV is an adaptively $f$-secure $M$-DistinctConverge protocol, for $f < (1 - \epsilon)n$. The number of bits sent over all nodes is

$$O\left(\sum_{l=1}^{\log n} \cdot \sum_{i\in[n]} CC(\text{Propagate}(M_i^l \setminus C_i^l)),\right)$$

where $CC(\text{Propagate}(M_i^l \setminus C_i^l))$ denotes the communication cost of node $P_i$ calling Propagate($M_i^l \setminus C_i^l$).

**5.3 Our Adaptive Protocol**

We first recall two useful definitions from Tsimos et al. [38].
Let \( n \) be the number of nodes and \( m = 10/\epsilon + \kappa \). For every node \( i \in [n] \), \( F_{\text{prop}} \) keeps a set \( O_i \) which is initialized to \( \emptyset \). Let \( M_i \) be node \( i \)'s input messages’ set.

- On input \((\text{SendRandom}, M_i)\) by honest node \( i \):
  - If \(|M_i| < \log n\), then for all \( j \in [n] \) set \( O_j = M_i \).
  - Else, for all \( x \in M_i \) and for all \( j \in [n] \) add \((i, x)\) to \( O_j \) with probability \( m/n \);
  - return \( M_i \) to adversary \( A \);
  - return \( O_i \) to node \( i \).

- On input \((\text{SendDirect}, x, J)\) by adversary \( A \) (for a corrupted node \( i \)):
  - Add \((i, x[j])\) to \( O_j \) for all \( j \in J \);
  - return \( O_i \) to adversary \( A \).

Fig. 8: Functionality \( F_{\text{prop}} \).

\[
\{M, k\}\text{-DistinctCV}(M_i, C_i)
\]

**Input:** Sets of messages \( M_i, C_i \) and a parameter \( k > 0 \).

**Output:** A set of messages \( O_i \).

**Global Parameters:**
- \( \text{Local}_i \) is a set inherited by the caller protocol.

For round \( r = 1, \ldots, \lceil \log cn \rceil \):
- \( O_i \leftarrow \text{Propagate}(\text{distinct}_r(M_i \setminus C_i)) \)
- \( \text{Local}_i \leftarrow \text{Local}_i \cup O_i \)
- \( C_i \leftarrow C_i \cup M_i \)
- \( M_i \leftarrow \text{Local}_i \cap M_i \)

**Output**
- Return \( M_i \).

Fig. 9: The \( \text{DistinctCV} \) protocol, an implementation of \( M\)-DistinctConverge.

**Definition 10 ((u, j)-committee).** For each message/slot pair \((u, j)\), the \((u, j)\)-committee is a subset of nodes such that for each node \( P_i \) in the \((u, j)\)-committee, whenever \( F_{\text{mine}} \) is queried on input \( F_{\text{mine}}.\text{verify}(\text{adaptive}, u, j) \), \( F_{\text{mine}} \) outputs 1.

Note that we have previously defined the notion of a valid \( r \)-batch for our static PBC protocol in Section 4. Here, we define the notion of a valid \( r \)-batch that is more in line with a Dolev-Strong-style protocol [18] below.

**Definition 11 (Valid \( r \)-batch).** A valid \( r \)-batch on pair \((u, j)\) is the element \( u || j || S\Gamma_r \), where \( S\Gamma_r \) is a set of at least \( r \) signatures (or aggregate signature) on \([u, j] \) consisting of one signature from node \( P_j \) and at least \( r - 1 \) signatures from nodes in the \((u, s)\)-committee (resp. or an aggregate signature with the contributions of \( P_j \) and at least \( r - 1 \) other nodes in the \((u, j)\)-committee).

**Definition 12.** Let \( M_r \) denote the set of all possible valid \( r \)-batches for all \( m \in \{0, 1\}^\kappa \) and for all \( s \in [n] \).
Lemma 12. Let $k^*$ be the number of bits required to describe $s||m$, where $s \in [n]$ and $m \in \{0, 1\}^{\kappa-1}$ is such that distinguishes between exactly 2 messages in $\mathcal{M}_r$ and where $\text{distinct}_{k^*}$ is defined in Definition 8. Then $|\text{distinct}_{k^*}(_r\mathcal{M})| = 2 \cdot n$.

Proof. Follows from the fact that $\mathcal{M}_r$ contains $2 \cdot n$ elements with unique $s||m$ prefixes ($s \in [n]$) and $m$ leads to outputting any but exactly 2 messages in $\mathcal{M}_r$.

We present our adaptively secure $\kappa$-valued PBC. The protocol (Figure 10) follows the template of TrustedPBC of [38], which itself follows the template of the broadcast protocol of Chan et al. [16], save for the following notable changes.

First, TrustedPBC is defined only for single-bit PBC. Therefore, we generalise it for multiple nodes, the main difference coming from our use of $\mathcal{F}_{\text{mine}}$. Abstractly, there are an exponential number of possible committees, one per message/slot pair (this number is quadratic in $n$ for TrustedPBC), but since $\mathcal{F}_{\text{mine}}$ can be evaluated on-demand for a given input, this is not an issue for complexity. Also, in our protocol, we guarantee that each node will forward at most two messages from the same sender, since they are sufficient to show that the sender is dishonest. Therefore, the size of the message space does not affect the total communication, except for the message length.

At the beginning of protocol execution, nodes send to all nodes their input value $u_i$ and a signature $\sigma_i$. Recall in Propagate that we use FS-PKE instead of regular public-key encryption. To bootstrap keys, each node therefore sample a FS-PKE key pair and send to all nodes their public key. Recall that we use the DistinctConverge protocol in a single round so that each honest node $P_i$ dissembles its message and all honest nodes receive it at the end of this round; TrustedPBC does not use constraint sets to this end.

Otherwise, the protocol at described at the level of abstraction of Figure 10 is comparable to previous Dolev-Strong-style protocols. Each round $r$ is divided into two phases. In the distribution phase, nodes propagate $r$-batches of messages associated with a given node $P_j$ that they have not previously propagated (using DistinctConverge), and for any such $r$-batches, they add the corresponding message to $\text{ExtractedSet}_j^r$. In the voting phase, nodes check, for each $r$-batch that they have received in the distribution phase, whether they are in the committee or not for the corresponding message/slot pair using $\mathcal{F}_{\text{mine}}$. If so, and they have not previously added their signature to the $r$-batch, they do so. Finally, at the end of $R$ rounds, nodes output a vector of values for each node $P_j$, which is $\perp$ if $|\text{ExtractedSet}_j^r| \neq 1$ and the message in $\text{ExtractedSet}_j^r$ otherwise.

Theorem 7. Protocol $\text{PBC}_{\kappa}^{\text{adaptive}}$ satisfies $f$-consistency with probability $1 - \text{negl}(\kappa) - \text{negl}(\lambda)$.

Theorem 8. Protocol $\text{PBC}_{\kappa}^{\text{adaptive}}$ satisfies $f$-validity with probability $1 - \text{negl}(\kappa)$.

Theorem 9. Protocol $\text{PBC}_{\kappa}^{\text{adaptive}}$ has $O(\log cn(n^2\lambda \kappa + n^2 \lambda^2 \kappa + n \lambda^3 \log n + n^2 \lambda^2 \log n))$ communication complexity.
Input: Each node $P_i$ inputs $\kappa$-bit value $u_i$.
Output: Each node $P_i$ outputs an $n$-valued vector $\text{out}_i$.

Global Parameters:
- Let $\epsilon$ be the fraction of honest nodes.
- Let $R$ be the total number of rounds.
- $\text{ExtractedSet}_i \leftarrow [\emptyset]^n$, $\text{VotedSet}_i \leftarrow [\emptyset]^n$, $\text{Local}_i \leftarrow \emptyset$.
- $\text{PK} \leftarrow \emptyset$, a set of FS-PKE public keys.
- $\text{sk}_e \leftarrow \bot$, $P_i$’s FS-PKE secret key.

**Round 0:**
Every node performs the following:
- $(\text{pk}_i, \text{sk}_i) \leftarrow \text{fspkegen} \ (\epsilon = 0)$.
- Send $(\text{SIGN}, u_i, \sigma_i)$ and $(\text{KEY}, \text{pk}_j)$ to all nodes where $\sigma_i$ is a signature on $[u_i, i]$.
- Upon receiving $(\text{SIGN}, u_j, \sigma_j)$ from $P_j$, add $\sigma_j$ to $\text{Received}_j$.
- Upon receiving $(\text{KEY}, \text{pk}_j, \sigma_j)$ from $P_j$, add $(\text{pk}_j, j)$ to $\text{PK}$.

**Round $r = 1, \ldots, R + 1$:**

**Distribute:**
- Add all received valid messages into $\text{Local}_i$.
- Find all $u_j \notin \text{ExtractedSet}_i$ in $\text{Local}_i$ with valid $r$-batches.
- If $|\text{ExtractedSet}_i| \leq 1$, add $u_j$ in $\text{ExtractedSet}_i$, else disregard.
- Find all $u_j \notin \text{ExtractedSet}_i$ in $\text{Local}_i$ with valid $r$-batches.
- If $r \leq R$, then let $C_i$ contain all messages that $P_i$ has propagated exactly twice through $\text{DistinctCV}_i$. If for some $j$, $|\text{ExtractedSet}_i| > 1$, then $C_i$ implicitly contains all messages of the form $[u_j, j]$.
- $\text{Local}_i \leftarrow M^{r\times}_{\text{DistinctCV}_i}(\text{Local}_i, C_i, k^*)$.

**Vote:**
If $r \leq R$:
- Find all $u_j \notin \text{ExtractedSet}_i$ in $\text{Local}_i$ with valid $r$-batches.
- For each such $u_j$ s.t. $\mathcal{F}_{\text{mine}}(\text{adaptive}, u_j, i) = 1$ (and $|\text{ExtractedSet}_i| \leq 1$):
  - Add $u_j$ to $\text{VotedSet}_i$ and $\text{ExtractedSet}_i$.
  - Extend the $r$-batch to include the new signature.
  - Send the message with the updated batch to all nodes.

**Output conditions**
- At the end of round $R$, return for each $j$ for which a message is received $u_j \in \text{ExtractedSet}_i$ if $|\text{ExtractedSet}_i| = 1$, or $\bot$ else.

---

Fig. 10: The $\text{PBC}_{\kappa}^{\text{adaptive}}$ protocol.

**References**

Appendices

A Proofs of $\text{PBC}^*_L$

Proof of Theorem 2.

Proof. In the dissemination phase, each node sends $M_i$ to all nodes so the communication complexity is thus $O(n^2L)$. In the reconstruction phase, each node sends SEND or ECHO messages to each other, where each message consists of a fragment and a witness. According to ECP, the length of both fragment and the witness is $O(L/n)$. The communication complexity of $\text{PBC}^*_L$ is thus $O(n^2L + P(\kappa))$.

As we only introduce three communication rounds on top of $\text{PBC}^*_\kappa$, the round complexity is the same as that for $\text{PBC}^*_\kappa$.

B Proofs of $\text{PBC}^\text{static}_\kappa$

Proof of Lemma 2.

Proof. We model the committee election process as a $c$-times independent and repeated experiments, where $c$ is the size of the committee: in one-time experiment, a determinate node is chosen randomly to be a committee member. This is equivalent to the process that each node calls the committee election oracle $F_{\text{mine}}$ to check whether it is a member of the committee.

To analyze the number of Byzantine nodes in the committee, let $P_i$ be the node selected in the $i$-th experiment, and the random variable $X_i = 1$ if $P_i$ is Byzantine, and $X_i = 0$ otherwise.

For the determinate committee member chosen in the $i$-th experiment, it is either honest or corrupt. Since $n$ is a sufficiently large number, a Byzantine node is chosen in a single experiment with a fixed probability $\alpha$, since the fraction of all Byzantine nodes in total $n$ nodes is $\alpha$. We thus have $Pr(X_i = 1) = \alpha$, for each $i = 1, 2, \cdots, c$. Let $Y = X_1 + \cdots + X_c$. $Y$ represents the number of Byzantine nodes chosen in these experiments. Based on the above analysis and probability theory, we have $E(Y) = \alpha c$.

According to Fact 1, we have:

$$Pr \left( Y \geq (\alpha + \mu)c \right) = Pr \left( Y \geq (1 + \frac{\mu}{\alpha})E(Y) \right) \leq e^{-\frac{\alpha^2E(Y)}{3\sigma^2}} = e^{-\frac{\alpha^2}{3\sigma^2}}$$

If $c > \frac{3\alpha^2}{\mu^2} \ln \frac{1}{\delta}$, 

$$Pr \left( Y \geq (\alpha + \mu)c \right) \leq e^{-\frac{\alpha^2}{3\sigma^2}} \leq \delta.$$ 

We now discuss the value of $\delta$. Typically, the failure probability of the protocol $\delta$ is a negligible function in some statistical security parameter. As a special
case, assuming that $\epsilon$ is any arbitrarily small positive constant, $0 < \mu < 1 - \epsilon$ and the mining difficulty parameter is $p_{\text{mine}} = \frac{32\mu}{\mu^2 \ln 2}$, then the failure probability $\delta = e^{-\omega(\log \lambda)}$ would be a negligible function, so with probability $1 - \text{negl}(\lambda)$.

Proof of Theorem 3.

Proof. $t$-Validity. If an honest node $P_i$ provides $M$ as input to $C()$, $P_i$ first compiles a union for the valid tuples in $M$ into $\text{Aggregated}_i$. We first show that if any valid tuple $M_j$ is part of $M$, $M_j$ is part of $\text{Aggregated}_i$. Then we show that $M_j$ is part of $\text{Merged}_k$ for every honest node $P_k$.

If any valid tuple $M_j$ is part of $M$, $M_j$ is part of $\text{Aggregated}_i$. Namely, we consider that $rs \in M_j$ where $rs$ is a valid $(r-1)$-s batch. We assume that $rs \notin \text{Aggregated}_i$ (i.e., $M_j$ is not part of $\text{Aggregated}_i$) and prove the correctness by contradiction. When $P_i$ compiles a union for $M$, there are two cases: 1) there does not exist any $M_k \in M$ such that a valid $(r-1)$-s batch is included in $M_k$, i.e., no node sends a valid $(r-1)$-s batch to the committee members; 2) there exists a $M_k \in M$ such that $rs' \in M_k$ and $rs'$ is a valid $(r-1)$-s batch, i.e., node $P_k$ sends a valid $(r-1)$-s batch to the committee member. In case 1, $rs \in \text{Aggregated}_i$, as $\text{Aggregated}_i$ is a union of $M$. In case 2, when $P_i$ compiles the union of $M$ into $\text{Aggregated}_i$, it takes a union of the signatures in both $rs$ and $rs'$ and include them in $\text{Aggregated}_i$.

Now we prove that $M_j$ is part of $\text{Merged}_k$ for any honest node $P_k$. We consider that the input of $\text{PBC}^*_L.c$ for node $P_i$ is $m_i = \text{Aggregated}_i$. According to Lemma 3, any honest node $P_k$ outputs $m_k[i] = m_i$. As any honest node $P_k$ further compiles a union of any valid tuple $m_i$ into $\text{Merged}_k$, $m_i$ is part of $\text{Merged}_k$. As $M_j$ is part of $m_i$, $M_j$ is part of $\text{Merged}_k$, according to the transitivity of the part-of relationship.

$t$-Consistency. According to the $t$-consistency property of PBC, if an honest node $P_i$ outputs $m_i[k] = m_k$, any honest node $P_j$ also outputs $m_j[k] = m_k$. Any honest node $P_i$ first filters invalid tuples in $m_i$ and then compiles a union of $m_i$ into $\text{Merged}_i$. Hence, for a slot $s \in [n]$ such that $\text{Merged}^i_s \neq \text{Merged}^j_s$, there must exist some valid tuple $m_k$ such that $m_k^s \in \text{Merged}^i_s$ but $m_k^s \notin \text{Merged}^j_s$, i.e., $m_k^s$ is output by $P_i$ but not $P_j$, violating the $t$-consistency property shown in Lemma 3.

Proof of Theorem 4.

Proof. As each node compiles a union of its input $M$ into an $n$-value vector and the total number of messages provided by any node for each slot $s \in [n]$ is a constant, the length of input for $\text{PBC}^*_L.c$ is $L$. The communication thus depends on the $\text{PBC}^*_L.c$ oracle. The lemma thus holds.

Proof of Lemma 5.

Proof. For round $r = 1, 2, \ldots, \lambda^K$, each node $P_i$ sends a list $L_j$ to every committee member $P_j$. Since every message of $M_i$ has been added to $L_j$ with probability
outputs a value \( v \), the size of \( \mathcal{L} \) is \(|M|/\lambda \). The communication complexity of \texttt{StaticPropagate()} is thus \( O \left( \left( |M|/\lambda \right) \cdot n \lambda \cdot \lambda^K \right) = O(n |M| \lambda^K) \).

**Proof of Theorem 5.** According to Lemma 4, if an honest node holds a message \( M \), at least one honest committee member receives \( M \). Furthermore, according to Lemma 6, if all committee members send the same message \( M \), \( M \) is part of \texttt{Received}, by any honest node \( P_i \). We therefore prove that the protocol \( \text{PBC}^\text{static} \) shown in Figure 4 achieves \( f \)-validity and \( f \)-consistency.

**\( f \)-Consistency.** We assume that for some \( s \in [n] \), an honest node \( P_i \) outputs \( v_i[s] \), another honest node \( P_j \) outputs \( v_j[s] \), and \( v_i[s] \neq v_j[s] \), and we prove the correctness by contradiction. In particular, if \( v_i[s] \neq v_j[s] \), there are two cases:

- **Case 1:** \( P_i \) outputs \( v_i[s] = u_i^s \) and \( P_j \) outputs \( v_j[s] = \bot \).
- **Case 2:** \( P_i \) outputs \( v_i[s] = u_i^s \) and \( P_j \) outputs \( v_j[s] = u_j^s \) such that \( u_j^s \neq u_i^s \).

We prove that either of the above two cases happens with probability at most \( \text{negl}(\kappa) \).

We focus on case 1 as case 2 can be proved similarly. If an honest node \( P_i \) outputs a \( v_i[s] = u_i^s \), \([\text{ExtractedSet}^s_i]\) = 1. There are two sub-cases for \( P_j \):

- **Sub-case 1:** \(|\text{ExtractedSet}^s_j| = 0 \), i.e., \(\text{ExtractedSet}^s_j = \emptyset\).
- **Sub-case 2:** \(|\text{ExtractedSet}^s_j| = 2 \), i.e., there exists two values \( u \) and \( u' \) such that \( u, u' \in \text{ExtractedSet}^s_j \).

In sub-case 1, \( u_i^s \) is added to \( \text{ExtractedSet}^s_i \) for \( P_i \) but not \( \text{ExtractedSet}^s_j \) for \( P_j \). In sub-case 2, at least one value (e.g., \( u' \)) is added to \( \text{ExtractedSet}^s_j \) but not \( \text{ExtractedSet}^s_i \). Without loss of generality, we consider that value \( u \) is added to \( \text{ExtractedSet}^s_i \) of an honest node \( P_i \) but is not added to \( \text{ExtractedSet}^s_j \) of another honest node \( P_j \). In this way, both sub-cases are covered.

We classify the following two types of scenarios and then show that for either type of scenario, at the end of the protocol, it is impossible that an honest \( P_i \) adds a value \( u \) to its \( \text{ExtractedSet}^s_i \) while another honest node \( P_j \) does not include \( u \) in its \( \text{ExtractedSet}^s_j \):

- **Type 1:** The value \( u \) is first added to \( \text{ExtractedSet}^s_i \) in round \( r = 0, 1, \ldots, R-1 \) by \( P_i \) but is never added to \( \text{ExtractedSet}^s_j \) by \( P_j \).
- **Type 2:** The value \( u \) is first added to \( \text{ExtractedSet}^s_i \) in round \( R \) by \( P_i \) but is not added to \( \text{ExtractedSet}^s_j \) by \( P_j \).

**Lemma 13.** In some round \( r \), if an honest committee member \( P_i \) creates a signature for \([u, s]\) and sends to all nodes, any other honest committee member \( P_j \) also creates a signature for \([u, s]\) and sends to all nodes with probability \( 1 - \text{negl}(\kappa) \).

**Proof.** If \( P_i \) creates a signature for \([u, s]\), a valid \((r - 1)\)-s batch is included in \( \text{Merged}^s_j \) and \( u \notin \text{VotedSet}^s_j \). We assume that \( P_j \) does not create a signature for \([u, s]\) and prove the correctness by contradiction.

If \( P_j \) does not create a signature for \([u, s]\), there are two cases: a valid \((r - 1)\)-s batch on \([u, s]\) is not included in \( \text{Merged}^s_j \); a valid \((r - 1)\)-s batch on \([u, s]\) is included in \( \text{Merged}^s_j \) but \( u \notin \text{VotedSet}^s_j \).
The scenario for type 1 will not happen with probability 1

In this case, the t-consistency property of the C() protocol is violated.

Case 2: An honest committee member \( P_j \) only adds \( u \) to its VotedSet\(^\epsilon\)_\( j \) in round \( r \) after it sees a valid \((r - 1)\)-s batch on \([u, s]\) in Merged\(^\epsilon\)_\( j \). If \( P_j \) has already added \( u \) to VotedSet\(^\epsilon\)_\( j \), it must have seen a valid \((r' - 1)\)-s batch in some round \( r' \) such that \( r' < r \), i.e., a valid \((r' - 1)\)-s batch is included in Merged\(^\epsilon\)_\( j \) in \( r' \) for \( P_i \) but not included in Merged\(^\epsilon\)_\( j \) for \( P_j \). This violates the t-consistency property of the C() protocol.

**Lemma 14. (Type 1).** For any value \( u \) and any slot \( s \in [n] \), if the scenario for type 1 happens, \( P_j \) adds \( u \) to ExtractedSet\(^\epsilon\)_\( j \) in round \( r + 1 \) with probability \( 1 - \text{negl}(\lambda) \).

**Proof.** In type 1, an honest node \( P_i \) adds \( u \) to ExtractedSet\(^\epsilon\)_\( j \) in round \( r \leq R - 1 \) but another honest node \( P_j \) has not added \( u \) to ExtractedSet\(^\epsilon\)_\( j \). We show that \( P_j \) will add \( u \) to ExtractedSet\(^\epsilon\)_\( j \) in round \( r + 1 \) with probability \( 1 - \text{negl}(\lambda) \).

If \( P_i \) adds \( u \) to ExtractedSet\(^\epsilon\)_\( j \) in round \( r \), \( P_i \) must have seen a valid \( r\)-s batch \( rs \) on \([u, s]\) for the first time. Moreover, all of the signatures consist in the valid \( r\)-s batch \( rs \) must be signed by corrupt committee members. This is because if an honest committee member \( P_k \) creates a signature on \([u, s]\) and sends to all nodes in round \( r \), according to Lemma 13, any other honest committee members also create a signature on \([u, s]\) and send to all nodes with probability \( 1 - \text{negl}(\lambda) \). Additionally, \( P_k \) already holds a valid \((r - 1)\)-s batch. Therefore, any honest node \( P_i \) will receive the signatures created by all honest committee members and the valid \((r - 1)\)-s batch, after which \( P_j \) will add \( u \) in ExtractedSet\(^\epsilon\)_\( j \), a violation of the scenario for type 1.

At the beginning of round \( r + 1 \), node \( P_i \) will send an (Echo, Received\(_j\)) message to all committee members where \( rs \in \text{Received\(_j\)} \). After each honest committee member \( P_i \) receives the (Echo) message, it sets \( M_i = \text{Received\(_i\)} \) where \( rs \in M_i \) and \( M_i \in M \). Following the t-validity property of the C() protocol, any honest committee member \( P_k \) outputs Merged\(_k\) such that \( M_i \) is part of Merged\(_k\). According to Definition 6 on the part-of relationship, \( rs \in \text{Merged\(_k\)} \) for any honest committee member \( P_k \).

Additionally, the honest committee member \( P_k \) sees \( u \) for the first time (i.e., \( u \not\in \text{VotedSet\(_k\)} \)), \( P_k \) creates a signature for \([u, s]\) and sends to all nodes. According to Lemma 13, any other honest committee member also creates a signature for \([u, s]\) and sends to all nodes with probability \( 1 - \text{negl}(\lambda) \). Thus, any honest node \( P_j \) receives at least \( c(\epsilon - \mu) \) signatures for \([u, s]\) and also a valid \( r\)-s batch. Accumulatively, \( P_j \) receives \( c(\epsilon - \mu) + r(c(\epsilon - \mu)) = c(r + 1)(\epsilon - \mu) = \frac{3(r + 1)(\epsilon - \mu)(1 - \epsilon)}{\mu^2} \log \frac{1}{\delta} \) signatures. That is, \( P_j \) receives a valid \((r + 1)\)-s batch in round \( r + 1 \) on \([u, s]\) and adds \( u \) to ExtractedSet\(^\epsilon\)_\( j \).

As \( P_j \) adds \( u \) to ExtractedSet\(^\epsilon\)_\( j \) in round \( r + 1 \) with probability \( 1 - \text{negl}(\lambda) \), the scenario for type 1 will not happen with probability \( 1 - \text{negl}(\lambda) \).

**Lemma 15. (Type 2).** Let \( R = \lceil \frac{\log 4}{(\epsilon - \mu)^2} \rceil \). For any value \( u \) and any slot \( s \in [n] \), the scenario for type 2 will not happen with probability \( 1 - \text{negl}(\lambda) \).
Proof. In type 2, an honest node \( P_i \) adds \( u \) to its ExtractedSet\(^i\) in round \( R \) for the first time but another honest node \( P_j \) has not added \( u \) to its ExtractedSet\(^j\). We show that this scenario occurs with probability at most \( \text{negl}(\lambda) \).

If \( P_i \) adds \( u \) to ExtractedSet\(^i\), it must have seen a valid \( R\)-s batch on \([u, s]\). According to Definition 4 on valid \( r\)-s batch, there are at least \( Rc(\epsilon - \mu) = c(1 - \epsilon + \mu) + 1 \) signatures on \([u, s]\) from committee members. As there are less than \( c(1 - \epsilon + \mu) \) corrupt committee members according to Lemma 2, at least one honest committee member \( P_k \) creates a signature on \([u, s]\). Let \( r \leq R \) be the round when the honest committee member \( P_k \) creates a signature on \([u, s]\). According to the protocol, \( P_k \) must have seen a valid \((r-1)\)-s batch on \([u, s]\) at the end of the second mini-round of round \( r \), i.e., the valid \((r-1)\)-s batch on \([u, s]\) is included in ExtractSet\(^k\). According to our protocol, \( P_k \) sends ExtractedSet\(^k\) to all nodes in the system. Additionally, according to Lemma 13, any honest committee member also creates a signature on \([u, s]\) and sends to all honest nodes with probability \( 1 - \text{negl}(\lambda) \). Thus, every honest node will receive a valid \((r-1)\)-s batch and signatures on \([u, s]\) from all committee members.

Accumulatively, \( P_i \) receives \( c(\epsilon - \mu) + rc(\epsilon - \mu) = rc(\epsilon - \mu) \) signatures. That is, every honest node including \( P_j \) sees a valid \( r\)-s batch on \([u, s]\) and adds \( u \) to ExtractedSet\(^j\) with probability \( 1 - \text{negl}(\lambda) \). Thus, the scenario for type 2 will not happen with probability \( 1 - \text{negl}(\lambda) \).

**Theorem 10.** PBC\(_\kappa\)\(^{\text{static}}\) satisfies f-consistency with probability \( 1 - \text{negl}(\lambda) \).

**Proof.** \( f \)-consistency follows from Lemma 14 and Lemma 15.

**f-Validity.** We now show that \( f \)-validity holds for PBC\(_\kappa\)\(^{\text{static}}\).

**Theorem 11.** PBC\(_\kappa\)\(^{\text{static}}\) satisfies \( f \)-validity with probability \( 1 - \text{negl}(\lambda) \).

**Proof.** If node \( P_i \) is honest, in round 0, every honest node will receive a \((\text{Sign}, u_s, \sigma_s)\) message from node \( P_s \), where \( \sigma_s \) is a signature on \([u_s, s]\). Accordingly, in round 1, each honest node \( P_j \) will send a message \((\text{Echo}, \text{Received}, j)\) to all committee members where \( \sigma_s \in \text{Received}, j \). Hence, in the valid tuple \( M_j = \text{Received}, j \) sent by each honest node \( P_j \), the signature \( \sigma_s \) on \([u, s]\) is included in \( M_j^\circ \) and also the input \( M \) for any honest committee member. According to the \( t \)-validity property of the \( C() \) protocol, as an honest committee member provides \( M_j \) as its input, \( M_j \) is part of \( \text{Merged}, i \) for any honest committee member \( P_i \). In the third mini-round, \( P_i \) will send \( \text{Merged}, i \) to all nodes. As the signature \( \sigma_s \) on \([u, s]\) from \( P_s \) is included in \( \text{Merged}, i \), \( P_i \) also creates a signature for \([u_s, s]\) and sends to all nodes. According to Lemma 13, any honest committee member creates a signature fore \([u_s, s]\) with probability \( 1 - \text{negl}(\lambda) \). It is then straightforward to see that every honest node sees a valid 1-s batch on \([u_s, s]\) and then adds \( u \) to ExtractedSet\(^j\).

Additionally, according to the unforgeability of the digital signature scheme, except with probability \( \text{negl}(\lambda) \), a signature on different value \( v_s \neq u_s \) such that \([v_s, s]\) cannot be forged by an adversary. Therefore, none of the honest nodes will receive a valid 0-s batch on \([v_s, s]\) in round 0. As a valid \( r \)-s batch must include a
digital signature from \( P_i \), none of the honest nodes will see a valid \( s \)-batch on \([v_s, s]\) for any round \( r = 0, 1, \ldots, R \). At the end of round \( R \), every honest node \( P_i \) thus has \( |\text{ExtractedSet}[^*]_i| = 1 \) and outputs \( v_i[s] = u_s \).

**Proof of Theorem 6.**

**Proof.** The round complexity of \( \text{PBC}^{\text{static}}_\kappa \) depends on both \( R \) and the round complexity of \( C(\cdot) \). The total round number is \( R = \frac{c(3\mu + \rho) + 1}{\epsilon} = O\left( \frac{n}{\epsilon} \right) \) according to Lemma 15. Additionally, as discussed in §4.1 and §4.2, the round complexity of the first mini-round and \( C(\cdot) \) are \( O(\lambda^K) \) and \( O(\lambda) \) respectively, where \( 0 < K < 1 \) is an arbitrarily small constant. Thus, the round complexity of \( \text{PBC}^{\text{static}}_\kappa \) is \( O\left( \frac{n}{\epsilon} \cdot \max(\lambda^K, \lambda) \right) = O(\lambda) \).

We now discuss the communication complexity. In round 0, every node sends its input (length \( L \)) and a signature to every other node so the communication is \( O(n^2 L + n^2 \lambda) \). In round \( r = 1, \ldots, R \), each round has three mini-rounds and the communication complexity of each mini-round is shown below.

- **First mini-round:** Every node sends an \( n \)-value vector to every committee member and each component has up to two valid \((r - 1)\)-s batch. Each valid \((r - 1)\)-s batch consists of one value and up to \( c(r - 1)(\epsilon - \mu) = O(\lambda) \) digital signatures. Accordingly, the length of the \( n \)-value vector is \( n\kappa + n\kappa\lambda(r - 1)(\epsilon - \mu) = O(n\kappa + n\kappa\lambda) \). According to Lemma 5, the communication complexity of this mini-round is \( O(n^2 \kappa \lambda^{1+K}) \).
- **Second mini-round:** The length of input of each node is \( O(n\kappa + n\kappa\lambda) \). According to §4.1, the communication complexity is \( O(n\kappa\lambda^3 + \lambda^4) \).
- **Third mini-round:** Every committee member sends an \( n \)-value vector to every node, i.e., with length \( O(n\kappa + n\kappa\lambda) \). According to Lemma 7, the communication complexity is \( O(n^2 \kappa \lambda + n\kappa\lambda^2) \).

As \( R \) is a constant, \( \text{PBC}^{\text{static}}_\kappa \) achieves \( O(n^2 \kappa \lambda^{1+K} + n\kappa\lambda^3 + \lambda^4) \) communication. If we use an aggregate signature scheme, the length of the \( n \)-value vector is \( n\kappa + n\kappa\lambda(r - 1)(\epsilon - \mu) = O(n\kappa) \) instead of \( O(n\kappa\lambda) \), so the communication complexity of \( \text{PBC}^{\text{static}}_\kappa \) is \( O(n^2 \kappa \lambda^K + n\kappa\lambda^2 + \kappa\lambda^3 + \lambda^4) \).

## C Proofs of \( \text{PBC}^{\text{adaptive}}_\kappa \)

**Proof of Lemma 8.**

**Proof.** The first case is trivial; if \( |M| < \log n \), then \( L_j = M \). For the second case, we distinguish two subcases depending on whether \( \log n \leq |M| < n \log n/\lambda \) or \( |M| \geq n \log n/\lambda \). For the first subcase, \( P \) randomly samples each lists \( L_j \) until each is of size at most \( 2m|\lfloor M \rfloor|/n \). We claim that with probability \( 1 - \text{negl}(\lambda) \), this will occur after polynomially many resamplings of each list. Fix a node who samples lists and fix a target node. Then, let \( X_i, i \in |M| \) be i.i.d. Bernoulli r.v.s denoting whether the \( i \)-th value of set \( M \) is added in the recipient’s list. Clearly, \( \Pr[X_i = 1] = m/n \). Let \( X = \sum_{i=1}^{|M|} X_i \). Then, \( \mathbb{E}[X] = m|\lfloor M \rfloor|/n \). The probability
that a list requires resampling is \( \Pr[X > 2m|M|/n] = \Pr[X > 2\mathbb{E}[X]] \). By Chernoff (Fact 1), we can bound this probability as follows:

\[
\Pr[X > 2\mathbb{E}[X]] = \Pr[X > (1 + 1)\mathbb{E}[X]] < e^{-\frac{4}{3}\mathbb{E}[X]} = e^{-\frac{m|M|}{3n}}.
\]

Assume that each list is resampled \( S = 3n/\log n \) times. Let \( Y_{j,r}, j \in [n], r \in [S], \) be i.i.d Bernoulli r.v.s denoting whether the \( r \)-th list sampled for \( P_j \) is of size at most \( 2m|M|/n \). Then, the probability that after at most \( S \) many samples of each of the \( n \) lists, there was some list that still required resampling, is bounded via a union bound as follows:

\[
\Pr[\bigcup_{j=1}^{S} Y_{j,r} = 0] \leq n \cdot \Pr[\bigcup_{r=1}^{S} Y_{j,r} = 0] = n \cdot \Pr[\bigcap_{r=1}^{S} Y_{j,r} = 0] = n \cdot \Pr[Y_{j,r} = 0]^S < n \cdot (e^{-\frac{m|M|}{3n}})^S = n \cdot e^{-m}.
\]

In case where \( m = \Theta(\lambda) \) and since \( n = \text{poly}(\lambda) \), then \( n \cdot e^{-m} = \text{poly}(\lambda) \cdot \text{negl}(\lambda) = \text{negl}(\lambda) \).

For the second subcase \((|M| \geq n \log n/\lambda)\), fix a recipient \( P_j \). As previously, let \( X_i, i \in [|M|] \) be i.i.d. Bernoulli r.v.s denoting whether the \( i \)-th value of set \( M \) is added in the recipient’s list. Clearly, \( \Pr[X_i = 1] = m/n \). Let \( X = \sum_{i=1}^{m|M|} X_i \). Then, \( \mathbb{E}[X] = m|M|/n \). The probability that a list is larger than \( \Lambda = 2m|M|/n \) is \( \Pr[X > 2m|M|/n] = \Pr[X > 2\mathbb{E}[X]] \). By Chernoff, we can bound this probability as follows:

\[
\Pr[X > 2\mathbb{E}[X]] = \Pr[X > (1 + 1)\mathbb{E}[X]] < e^{-\frac{4}{3}\mathbb{E}[X]} = e^{-\frac{m|M|}{3n}}.
\]

Since \( |M| \geq n \log n/\lambda \), then \( \Pr[X > \Lambda] < e^{-\frac{m \log n}{3}} = \text{negl}(\lambda) \), if \( m = \Theta(\lambda) \).

**Proof of Lemma 9.**

*Proof.* For the first case, it suffices to observe that \( M \) is of size \( O(\log n) \) and, since \( P \) sends \( M \) to all, the communication is \( O(n \cdot |M| \cdot s = O(ns \log n) \).

Similarly, for the second case we observe from Lemma 9 that with probability \( 1 - \text{negl}(\lambda) \) each list \( L_j, j \in [n] \) is of size \( \leq 2m|M|/n \). \( P \) sends a list to every node, so the communication for \( P \) is \( O(n \cdot |L_j| \cdot s) = O(n(2m|M|/n)s) = O(m \cdot |M| \cdot s) \).

**Proof of Lemma 10.**

*Proof.* The proof follows closely to proof of Lemma 8 from [38]. The sole two differences are the list construction, and the FS-PKE scheme instead of CPA-secure PKE scheme. As in the cited proof, for the list construction, we denote that Propagate still follows the same distribution of propagation of messages as in \( F_{\text{prop}} \), while it also still satisfies the property of each node sending lists of the same size to all parties, thus hiding the communication pattern. Therefore, the proof is straightforward to construct via a similar hybrid argument as in the proof of Lemma 8 from [38]. We note for completeness that composability should be preserved even under a weakly adaptive adversary as we consider here.
Proof of Lemma 11.

Proof. The security can be proven similarly as in Theorem 1 of [38]. The only difference in $F_{\text{prop}}$ is only that nodes might add the entire list, if the list is small enough, which can only help with the propagation of messages. The communication follows from the protocol. The number of bits sent by one node in round of $\text{DistinctCV}$ is the number of bits sent by the call to propagate. Thus, overall the communication over all nodes is:

$$O(\sum_{l=1}^{\log cn} \cdot \sum_{i \in [n]} \mathbb{C}(\text{Propagate}(M_i^l \setminus C_i^l))).$$

Proof of Theorem 7.

Proof. The proof follows similarly from [38, Lemma 14]. Suppose that for some slot $s$, an honest node $P_j$ adds message $b$ to $\text{ExtractedSet}_j^s$ at some round $r$. We prove that by the end of the protocol all honest nodes $P_i$ add $b$ to their $\text{ExtractedSet}_i^s$ sets with probability at least $1 - \text{negl}(\kappa)$. We distinguish cases depending on the step of the protocol during which $P_j$ added message $b$ to $\text{ExtractedSet}_j^s$:

1. $r \leq R$ and $P_j$ adds $b$ to $\text{ExtractedSet}_j^s$ during the Vote stage. Then $P_j$ sends a valid-$(r + 1)$ batch $v_i^r$ for $(b, s)$ to all parties during the Vote stage, and therefore all parties $P_i$ add $b$ to their $\text{ExtractedSet}_i^s$ sets during the Distribute stage of round $r + 1$.

2. $r \leq R$ and $P_j$ adds $b$ to $\text{ExtractedSet}_j^s$ during the Distribute stage. Then, $P_j$ has received, during the Distribute stage of round $r$, a valid $r$-batch $v$ for $(b, s)$. Valid $r$-batch $v$ belongs to the set $V$ provided as input to $\text{DistinctCV}$. From Lemma 11, all honest parties output a set that contains $v$ after $P_j$ calls $\text{DistinctCV}$ with $v$ as part of its input. By Lemma 2, there is at least one honest voter $P_\ell$ in the $(b, s)$-committee. We distinguish two cases.

   (a) $P_\ell$ has not voted before for $(b, s)$. Then, $P_\ell$ will send a valid-$(r + 1)$ batch $v_i^r$ for $(b, s)$ to all parties during Vote. Therefore all honest parties $P_i$ add $b$ to their $\text{ExtractedSet}_i^s$ sets during the Distribute phase of $(r + 1)$;

   (b) $P_\ell$ voted before for $(b, s)$. Then let $r' < r$ be the round in which $P_\ell$ voted for $(b, s)$. Then, $P_\ell$ forwarded a valid-$(r' + 1)$ batch $v_i^{r'}$ for $(b, s)$ to all parties during Vote of $r'$. Therefore all parties $P_i$ added $b$ to their $\text{ExtractedSet}_i^s$ sets during Distribute of $(r' + 1)$.

3. $P_j$ adds $b$ to $\text{ExtractedSet}_j^s$ during Distribute of round $(R + 1)$: In this case, $P_j$ observes a valid $(R + 1)$-batch for $(b, s)$. By Lemma 2, at least one of the voters, say voter $P_t$, is honest. Let $r' < R + 1$ be the round when $P_t$ voted for $(b, s)$. This means that $P_t$ sent a valid-$(r' + 1)$ batch $v_i^{r'}$ for $(b, s)$ to all parties during Vote of $(r')$ and therefore all honest parties $P_i$ added $b$ to their $\text{ExtractedSet}_i^s$ sets during Distribute of $(r' + 1)$.  

38
Proof of Theorem 8.

Proof. Assume that $P_j$ is honest and inputs $u_i$. Then, $P_j$ will send a valid 1-batch $(u_i, \sigma)$ to all nodes alongside its public key $pk_j$. All nodes will then add $(u_i, \sigma)$ to $\text{ExtractedSet}_j^i$, and, by the security of the signature scheme, no other signature for slot $j$ can exist (thus for all $P_i$, $|\text{ExtractedSet}_j^i| \leq 1$). The argument then follows from our argument for consistency.

Proof of Theorem 9.

Proof. To calculate the communication complexity of $\text{PBC}_\kappa^{\text{adapt}}$, we must first consider how aggregate signatures and $F_{\text{mine}}$ may be efficiently instantiated. Note $F_{\text{mine}}$ can be implemented by checking a $O(\kappa)$-sized digest, as is the case in [2] where $F_{\text{mine}}$ was instantiated from non-interactive zero-knowledge proofs (NIZKs). Recall that $F_{\text{mine}}$ is also used in $\text{PBC}_\kappa^{\text{static}}$. In $\text{PBC}_\kappa^{\text{static}}$, $F_{\text{mine}}$ is only invoked in round 0, and in particular several proofs need not be combined or sent together at any point. By contrast, our notion of $r$-batches depends on $F_{\text{mine}}$, and when using aggregate signatures, an $r$-batch can be the result of iterative signature aggregation.

Recall from Section 2 that an aggregate signature signed by $r$ nodes (committee members here) is of size $\min\{O(\kappa + r \log n), O(\kappa + n)\}$. Note that an $r$-batch, with our sketched instantiation of $F_{\text{mine}}$, must in general contain $r$ proofs, which naively uses $O(r\kappa)$ space. However, if we assume the existence of NIZKs that can be recursively combined, we assume that even if the proofs contain information about how the proofs were combined together (e.g., encoding the indices of the nodes that ‘combined’ the proofs in-order), that this information should not require more than $O(\kappa + r \log n)$ bits to encode.

We now analyze each step of communication of the protocol of Figure 10. At the first step, each node sends its input value, signature and FS-PKE public key to all nodes. Using [12] and [15] to instantiate the FS-PKE scheme, each public key is of size $O(\kappa \log \kappa)$. Thus, this incurs $O(n \cdot \kappa \log \kappa)$ communication. So the first round incurs $O(n^2 \cdot \kappa \log \kappa)$ communication over all nodes.

Then, there are $O(\lambda)$ rounds $r$ of the protocol, where each node either calls $\text{DISTINCTCV}$, or votes. For the latter case first, each node will vote only for messages where it is in the corresponding committee, and for each committee it will vote only once. For each such committee $c$, let $I_{c,i}$ be an indicator variable that is 1 if and only if the node is in the corresponding committee. Then, each node will send $O(\sum_{c=1}^{2n} I_{c,i} \cdot n \cdot (\kappa + \lambda \log n))$. Overall, for all nodes, the communication will be

\[
O(\sum_{i=1}^{n} \sum_{c=1}^{2n} I_{c,i} \cdot n \cdot (\kappa + \lambda \log n)) = O(n \cdot \lambda \cdot n \cdot (\kappa + \lambda \log n)) = O(n^2 \lambda \kappa + n^2 \lambda^2 \log n),
\]

since $\sum_{i=1}^{n} \sum_{c=1}^{2n} I_{c,i} = O(n \cdot \lambda)$. Now, when a node calls $\text{DISTINCTCV}$, from Lemma 11 it incurs $O(\sum_{c=1}^{\log n} \cdot \sum_{i \in [n]} CC(\text{Propagate}(M_i^l \setminus C_i^l)))$ communication. Each node calls $\text{DISTINCTCV}$ once per each of the $O(\lambda)$ rounds of the protocol. Each node will forward for each sender $s$ at most two messages, and also
each node will forward each message at most twice. Therefore, over all calls of Propagate we have that 
\[ P \log \epsilon n \leq 2 \cdot n. \] Therefore, there can be at most two rounds for each party, therefore for this case the total communication is

\[ \sum_{r=1}^{O(\lambda)} \sum_{l=1}^{\log \epsilon n} X_{i \in [n]} X_{r=1}^{O(\lambda)} X_{l=1}^{\log \epsilon n} CC(Propagate(M_{r,l}^{r,l} \backslash C_{i}^{r,l})) \leq n \cdot O(\lambda \log n). \]

where the communication of one such step of Propagate is bounded from the first case of Lemma 9.

2. \[ \log n \leq |M_{r,l}^{r,l} \backslash C_{i}^{r,l}|. \] In that case, we have from the second case of Lemma 9:

\[ \sum_{r=1}^{O(\lambda)} \sum_{l=1}^{\log \epsilon n} X_{i \in [n]} X_{r=1}^{O(\lambda)} X_{l=1}^{\log \epsilon n} CC(Propagate(M_{r,l}^{r,l} \backslash C_{i}^{r,l})) \]

\[ = \sum_{r=1}^{O(\lambda)} \sum_{l=1}^{\log \epsilon n} O(\lambda \cdot |M_{r,l}^{r,l} \backslash C_{i}^{r,l}| \cdot (\kappa + \lambda \log n)) \]

\[ \leq O(n \lambda \cdot (\kappa + \lambda \log n) \cdot \sum_{r=1}^{O(\lambda)} \sum_{l=1}^{\log \epsilon n} |M_{r,l}^{r,l} \backslash C_{i}^{r,l}|) \leq O(n^2 \lambda \cdot (\kappa + \lambda \log n)), \]

where in the last inequality we used the inequality that we derived before, i.e. \[ \sum_{r=1}^{O(\lambda)} \sum_{l=1}^{\log \epsilon n} |M_{r,l}^{r,l} \backslash C_{i}^{r,l}| \leq 2 \cdot n. \]

If we add all the computed communications, we derive the upper bound of the theorem statement.

D Instantiation of Communication-Efficient Committee Description

In the paper, we mention that we can efficiently describe an \( r \)-batch of signatures from committee members by using aggregate signatures and recursive snark proofs to prove committee membership. In this section we describe the exact details. The goal as already explained is to lower the communication size of an \( r \)-batch from \( \Theta(r \cdot \kappa) \) to \( \Theta(\kappa + r \log n) \), while allowing for the batch to be updateable to an \( (r + 1) \)-batch from a new committee member. Aggregate signatures directly allow for the signature part of the batch to be described with that communication size. The remaining part describes such batches and their proof update.

We first provide the definition of Non-Interactive Zero Knowledge proofs (NIZKs).
Definition 13 (NIZKs). Let an NP relation \( R \). Let statement \( x \) and witness \( w \) s.t. \( (x,w) \in R \). Let \( L \) denote the language that consists of statements in \( R \). A non-interactive zero-knowledge proof is a triple of PPT algorithms (\( \text{Gen, Prove, Verify} \)) such that:

- \( \text{Gen}: \) given the security parameter \( \kappa \) (in unary) outputs public parameter \( pp \); 
  \( pp \leftarrow \text{Gen}(1^\kappa) \).

- \( \text{Prove}: \) given \( pp \), a statement \( x \) and a witness \( w \), outputs proof \( \pi \); 
  \( \pi \leftarrow \text{Prove}(pp, x, w) \).

- \( \text{Verify}: \) given \( pp \), statement \( x \) and proof \( \pi \), outputs a bit \( b \); 
  \( b \leftarrow \text{Verify}(pp, x, \pi) \).

The specific properties we require from the NIZK system we use are inherited by the construction of the committee-membership NIZK proofs that are extensively described in [2, 16]. Namely we require perfect completeness, on-erasure computational zero-knowledge, and perfect knowledge extraction, as defined in [16].

So far, in committee based approaches [16, 38], a valid \( r \)-batch is structured as a tuple

\[
(\sigma_s(v), \{(\sigma_i(v), \pi_{v,i}^{\text{com}})\}_{i \in C_r^v})
\]

where \( \sigma_s(v) \) denotes the signature of the designated sender \( P_s \) on value \( v \), \( C_r^v \) is a bitmap representation of a set of indices corresponding to some parties in the \( v \) committee and \( \pi_{v,i}^{\text{com}} \) corresponds to a proof – a NIZK hereafter – that party \( P_i \) indeed belongs to the \( v \) committee. For the exact description of such a NIZK, see [16], but for the rest of our description we refer to that NIZK as \( \text{nizk}_1 \). Such a batch is considered \( r \)-valid if all signatures are valid, the number of signatures (including the sender’s) is at least \( r \) and each signature is accompanied by a valid proof that the party is elected in the \( v \) committee. If a party \( P_t \) wants to update the batch with its own signature, it simply has to append a tuple \( (\sigma_t(v), \pi_{v,t}^{\text{com}}) \), such that i) \( \sigma_t(v) \) does not appear already in the batch and ii) \( \pi_{v,t}^{\text{com}} \) is also a valid proof that \( P_t \) is in the \( v \) committee. The updated batch will then be considered as a valid \( (r+1) \)-by any honest party receiving it. Notice that the communication complexity of an \( r \)-batch is \( O(r \cdot \kappa) \).

In our work, we propose two structural changes that allow for \( r \)-batches to be represented in a more communication-efficient way. As a first approach we propose that parties, instead of sending their respective signature, they can instead construct a multisignature \( \sigma_{C_r^v}(v) \) combining all respective signatures from parties in the set \( C_r^v \). Any party knowing the set \( C_r^v \) can efficiently verify that the multisignature is a representation of all signatures \( \{\sigma_i(v)\}_{i \in C_r^v} \), by running \( \text{Ver}(PK, \sigma_{C_r^v}(v), v, C_r^v) \), where \( PK \) corresponds to the list of all \( n \) public keys (see Aggregate signatures (Section 2.3)). An efficient representation of \( C_r^v \) can be given by a mapping of indices \( \{i_1, i_2, \ldots, i_{r-1}\} \) with \( O(r \cdot \log n) \) bits. Each index represents the binary description of value \( i_j \), which takes \( \log n \) bits. This

\footnote{Notice that the same structure is true for the Dolev-Strong protocol [18]; however, the committee membership proofs are not required since every party is an effective member of the \( n \) committee in that protocol.}
set can be easily updated by simply including the new party’s index. Still, this approach clearly does not tackle the issue of the independent proofs of committee membership that need to also be propagated in the \( r \)-batch.

Therefore, the last issue to tackle is how to more efficiently represent the set of all proofs of committee membership, that must accompany the signatures. For this, we also employ NIZKs.

Let \( \text{nizk}_2 \) denote the following relation:

- \( \text{nizk}_2 \) statements of the form \( x := (s, v, \sigma_s(v), C^r_v) \) and witnesses of the form \( w := (t, \pi^r_v, \sigma(t), \pi^\text{com}_{r,t}) \), such that:
  1. \( C^r_v \subset [n] \) and \( t \in C^r_v \);
  2. \( \sigma_s(v) \) is a valid signature from \( P_s \) on \( v \);
  3. either \( C^r_v = \{s\} \) and \( t = s \), or else \( \pi^r_v \) is a valid \( \text{nizk}_2 \) proof w.r.t. \( s, v, \sigma_s(v), C^r_v - \{t\} \);
  4. \( \sigma(t) \) is a valid signature from \( P_t \) on \( v \);
  5. Either \( t = s \) and \( C^r_v = \{s\} \), or else \( \pi^\text{com}_{r,t} \) is a \( \text{nizk}_1 \) proof that party \( P_t \) is in the \( v \) committee (as defined in [16]).

Notice that condition 3. does not lead to a circular argument, rather a recursive definition. The recursion has an initial condition, for the case where \( C^r_v = \emptyset \) and is thus valid. Also notice that the time it takes for any such check is polynomial and the relation is thus an NP relation.

A valid \( r \)-batch with our renewed approach is of the form

\[
(\sigma_s(v), C^r_v, \pi^r_v)
\]

where \( C^r_v \) is still a bitmap representation of \( r - 1 \) parties and \( \pi^r_v \) is a \( \text{nizk}_2 \) proof.

The updated construction has the following actions.

**Setup.** The trusted party additionally with all other actions, also runs the CRS generation algorithms of the NIZK scheme to obtain \( \text{crs}_2 \), which is added to the public parameters, say \( pp \).

**Designated Sender.** In order to forward its input value \( v \) to all parties in the initial round of the protocol, the designated sender \( P_s \) calls

\[
\text{nizk}_2.\text{Prove}(pp, x := (s, v, \sigma_s(v), \{\}), w := (s, \bot, \sigma_s(v), \bot))
\]

to obtain a valid initial \( \text{nizk}_2 \) proof \( \pi^1_v \). Then \( P_s \) will send \( (\sigma_s(v), \{s\}, \pi^1_v) \) to all parties as a valid 1-batch. Notice that if the designated sender is honest, no dishonest party can forge a 1-batch for a different value \( v' \), since it can not even compute \( \sigma_s(v') \) (and does not have access to \( \text{sk}_s \)).

**Honest parties.** A party \( P_t \) observing a batch \( (\sigma_s(v), C^r_v, \pi^r_v) \) accompanying value \( v \) in round \( r \), considers the batch as \( r \)-valid if 1. \( |C^r_v| \geq r \), 2. the corresponding designated sender matches \( P_s \), and 3. \( \text{nizk}_2.\text{Verify}((s, v, \sigma_s(v), C^r_v), \pi^r_v) = 1 \).
If the party \( P_t \) is also in the corresponding committee and has to update the received batch with its own signature, it calls

\[
\pi_v^{r+1} \leftarrow \text{nizk}_2.\text{Prove}(pp, x := (s, v, \sigma_s(v), C_v^r \cup \{t\}), w := (t, \pi_v^r, \sigma_t(v), \pi^{\text{com}}_{v,t})).
\]

\( P_t \) can then send \((\sigma_s(v), C_v^r \cup \{t\}, \pi_v^{r+1})\) as a valid \((r+1)\)-batch to any party in the protocol.

It is straightforward that any such NIZK proof for a set \( C \) requires for all honest parties in \( C \) to have already constructed one step of the recursive NIZK proof or to have propagated their own signature on the value; otherwise, an adversary who can simulate all the steps to construct a valid \( \text{nizk}_2 \) proof for a set \( C \) that includes honest parties, must be able to break the security of the signature scheme. Finally, notice that the bit size of the proposed \( r \)-batches is \( O(r \log n + \kappa) \), where the \( r \log n \) factor comes from the size of the bitmap representation of set \( C_v^r \) and the \( \kappa \) factor is the size of the signature \( \sigma_s(v) \) and the single NIZK proof \( \pi_v^r \).

We also observe that in this construction we do not require aggregate signatures, since the existence of the signatures of the respective parties is proven via our proposed NIZK proof.