# Towards Optimal Parallel Broadcast under a Dishonest Majority

Daniel Collins[1], Sisi Duan[2], Julian Loss[3], Charalampos Papamanthou[4], Giorgos Tsimos[5], and Haochen Wang[6]

[1] Texas A&M University, danielpatcollins@gmail.com
[2] Tsinghua University and State Key Laboratory of Cryptography and Digital Economy Security, duansisi@tsinghua.edu.cn
[3] CISPA Helmholtz Center for Information Security, lossjulian@gmail.com
[4] Yale University, charalampos.papamanthou@yale.edu
[5] University of Maryland, tsimos@umd.edu
[6] Tsinghua University, whc20@mails.tsinghua.edu.cn

**Abstract.** The parallel broadcast (PBC) problem generalizes 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. The state-of-the-art PBC protocol, TrustedPBC, is due to Tsimos, Loss, and Papamanthou (CRYPTO 2022), which is secure under an adaptive adversary assuming $f < (1-\epsilon)n$, where $f$ is the number of Byzantine failures and $\epsilon \in (0,1)$. TrustedPBC focuses on single-bit inputs and achieves $\tilde{O}(n^2\kappa^4)$ communication and $O(\kappa \log n)$ rounds.

In this work, we propose three PBC protocols for $L$-bit messages, for any size $L$, that significantly improve TrustedPBC. First, we propose a new extension protocol that uses a $\kappa$-bit PBC as a black box and achieves i) communication complexity of $O(Ln^2 + n^3\kappa + \mathcal{P}(\kappa))$, where $\mathcal{P}(\kappa)$ 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 $O(n^2\kappa^{1+K} + n\kappa^3 + \kappa^4)$ 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 $\tilde{O}(n^2\kappa^2 + n\kappa^3)$ communication overhead and $O(\kappa \log n)$ round complexity. 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.

## 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,

| $\|m\|$ | Protocol | Model | Adv. | $f <$ | Communication | Rounds |
|---|---|---|---|---|---|---|
| 1 | BulletinBC$^{\ddagger}$ [20] | bulletin | static | $(1-\epsilon)n$ | $\tilde{O}(n^3\kappa^2)$ $(=\tilde{O}(\mathbf{C}^5))$ | $O(n)$ |
| | FloodBC$^{\star}$ [6] | trusted | static | $(1-\epsilon)n$ | $\tilde{O}(n^2\kappa^3)$ $(=\tilde{O}(\mathbf{C}^5))$ | $O(\kappa)$ |
| | BulletinPBC [20] | bulletin | adaptive | $(1-\epsilon)n$ | $\tilde{O}(n^3\kappa^2)$ $(=\tilde{O}(\mathbf{C}^5))$ | $O(n\log n)$ |
| | TrustedPBC [20] | trusted | adaptive | $(1-\epsilon)n$ | $\tilde{O}(n^2\kappa^4)$ $(=\tilde{O}(\mathbf{C}^6))$ | $O(\kappa\log n)$ |
| | $\mathsf{PBC}_1^{\mathsf{static}}$ (§4) | trusted | static | $(1-\epsilon)n$ | $O(n^2\kappa^{1+K}+n\kappa^3+\kappa^4)=O(\mathbf{C}^4))$ | $O(\kappa)$ |
| | $\mathsf{PBC}_1^{\mathsf{adaptive}}$ (§5) | trusted | adaptive | $(1-\epsilon)n$ | $\tilde{O}(n^2\kappa^2+n\kappa^3)=\tilde{O}(\mathbf{C}^4))$ | $O(\kappa\log n)$ |
| $L$ | ANS [3] | trusted | adaptive | $n/2$ | $O(n^2L+n^3\kappa)=O(\mathbf{C}^2L+\mathbf{C}^4))$ | $O(1)$ |
| | NRSVX [19] | $\star$ | $\star$ | $(1-\epsilon)n$ | $O(n^2L+\mathcal{P}(\kappa)+n^3\kappa+n^4)$ $(=O(\mathbf{C}^2L+\mathcal{P}(\mathbf{C})+\mathbf{C}^4))$ | $O(n)$ |
| | TLP [20] | trusted | adaptive | $(1-\epsilon)n$ | $\tilde{O}(n^2\kappa^4L)(=\tilde{O}(\mathbf{C}^6L))$ | $O(\kappa\log n)$ |
| | AC [5] | trusted | static | $n/3$ | $O(n^2L+n^3\log^2 n)$ $(=O(\mathbf{C}^2L)+\tilde{O}(\mathbf{C}^3)\ )$ | $O(1)$ |
| | $\mathsf{PBC}_L^{\star}$ (§3) | SRS+$\star$ | $\star$ | $(1-\epsilon)n$ | $O(n^2L+n^3\kappa+\mathcal{P}(\kappa))$ $(=O(\mathbf{C}^2L+\ \mathcal{P}(\mathbf{C})+\mathbf{C}^4))$ | $O(\mathcal{T}(\kappa))$ |
| | $\mathsf{PBC}_L^{\mathsf{static}}$ (§3 & §4) | trusted | static | $(1-\epsilon)n$ | $O(n^2L+n^3\kappa+n^2\kappa^{1+K}+n\kappa^3+\kappa^4)$ $(=O(\mathbf{C}^2L+\mathbf{C}^4))$ | $O(\kappa)$ |
| | $\mathsf{PBC}_L^{\mathsf{adaptive}}$ (§3 & §5) | trusted | adaptive | $(1-\epsilon)n$ | $O(n^2L+n^3\kappa)+\tilde{O}(n^2\kappa^2+n\kappa^3)$ $(=O(\mathbf{C}^2L)+\tilde{O}(\mathbf{C}^4))$ | $O(\kappa\log n)$ |

Table 1: Comparison of the PBC protocols where honest nodes broadcast messages length $\leq |m|$. ‡PBC that runs $n$ parallel instances. $\star$The assumptions (*bulletin* board PKI, *trusted* PKI and/or structured reference string (*SRS*)) 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 $O(f(n)\cdot \mathsf{poly}(\log n))$ for some polynomial $\mathsf{poly}$. $\mathbf{C}$ captures practical settings where $\mathbf{C} = O(n) \approx O(\kappa)$.

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 [20] gave an efficient designated parallel broadcast (PBC) protocol under dishonest majority, TrustedPBC. Denoting $n$ as the number of nodes and $\kappa$ as the length of a signature, TrustedPBC achieves $\tilde{O}(n^2\kappa^4)$ 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 improved substantially the communication with respect to $n$. While TrustedPBC already achieves improved communication, its communication is still high, especially when $n$ and $\kappa$ are close. Additionally, TrustedPBC is limited to single-bit inputs.

**Our contributions.** In this work, we study 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 and Figure 1, we provide three protocols with improved communication: a new extension protocol $\mathsf{PBC}_L^{\star}$ that achieves

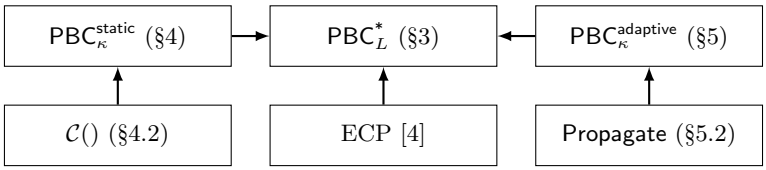


Fig. 1: Overview of our results.

both improved communication and round; a $\kappa$-bit PBC $\mathsf{PBC}_\kappa^{\mathsf{static}}$ in the static adversary model that achieves improved communication and round; a $\kappa$-bit PBC $\mathsf{PBC}_\kappa^{\mathsf{adaptive}}$ with improved communication in the adaptive adversary model. 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, $\mathsf{PBC}_L^*$, that reduces the $L$-bit PBC problem to a $\kappa$-bit PBC oracle. Compared to prior extension protocols, e.g., running $n$ BC instances of the protocol of Nayak, Ren, Shi, Vaidya, and Xiang (NRSVX) [19], $\mathsf{PBC}_L^*$ achieves both improved communication and round complexity. The adversarial assumption of $\mathsf{PBC}_L^*$ depends on the underlying $\kappa$-bit PBC oracle. In particular, if the $\kappa$-bit PBC is adaptively secure, then so is $\mathsf{PBC}_L^*$.

We then present $\mathsf{PBC}_\kappa^{\mathsf{static}}$, a $\kappa$-bit PBC protocol in the static adversarial setting. $\mathsf{PBC}_\kappa^{\mathsf{static}}$ can be generalized to $L$-bit PBC. However, using it as a $\kappa$-bit PBC in our extension protocol results in a more communication-efficient PBC. Compared to the state-of-the-art protocols BulletinBC [20] and FloodBC [6], $\mathsf{PBC}_\kappa^{\mathsf{static}}$ 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, $\mathsf{PBC}_\kappa^{\mathsf{static}}$ achieves $O(n^2\kappa^{1+K} + n\kappa^3 + \kappa^4)$ communication and $O(\kappa)$ rounds for $\kappa$-bit broadcast, where $K$ is an arbitrarily small constant such that $0 < K < 1$.

Finally, we present $\mathsf{PBC}_\kappa^{\mathsf{adaptive}}$, 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. [20], 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, a $O(\kappa^3)$ improvement over that of TrustedPBC for $\kappa$-sized messages. Similarly to $\mathsf{PBC}_\kappa^{\mathsf{static}}$, we can use $\mathsf{PBC}_\kappa^{\mathsf{adaptive}}$ as a $\kappa$-bit PBC oracle in our extension protocol to obtain a more communication-efficient $L$-bit PBC.

## 2 Preliminaries

**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 and the adaptive adversary models. In the static model, the adversary corrupts nodes prior to the start of the protocol. In the adaptive model, the adversary can choose the set of corrupted nodes at any moment during the execution of the protocol based on its current state. 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 node sent before they become corrupted. Additionally, we restrict the adversary by assuming *atomic sends* [7] 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.

**Normalizing security parameters.** Let $\kappa$ denote the cryptographic security parameter, i.e., the length of hashes or digital signatures. In this work, we also use $\lambda$ as the statistical parameter. We may consider $\lambda = O(\kappa)$, as typically $\lambda < \kappa$. We can also say that the security parameter of the system is the maximum of $\lambda$ and the cryptographic security parameter (e.g., length of digital signatures). When we discuss the concrete complexities in the main body of the paper, we differentiate $\lambda$ and $\kappa$. Also note that $\kappa$ and $\lambda$ are independent of $n$ and we usually assume $n \gg \kappa$ and $n \gg \lambda$. In Table 1, we provide $\mathbf{C}$ assuming $O(n) \approx O(\kappa) \approx O(\lambda)$ for the ease of understanding.

### 2.1 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 $\boldsymbol{v}_i$. Each slot $s$ in $\boldsymbol{v}_i$ is dedicated for the value broadcast by $P_s$, the output of which is denoted as $\boldsymbol{v}_i[s]$. 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 $\boldsymbol{v}_i$. $\Pi$ should achieve the following properties with probability $1 - \mathsf{negl}(\kappa)$ whenever at most $f$ nodes are corrupted.*

- *$f$-**Validity**: If $P_s$ is honest, the output $\boldsymbol{v}_i$ at any honest node $P_i$ satisfies $\boldsymbol{v}_i[s] = v_s$.*
- *$f$-**Consistency**: All honest nodes output the same vector $\boldsymbol{v}'$.*

We will need an *external validity* property for some of our constructions defined as follows. There exists a predicate $Q()$ known by all nodes. Given a value $v$, every node can query $Q(v)$ to validate $v$. In the literature, the value $v$ can be validated by via some additional data such as digital signatures [9]. Alternatively, $v$ can be validated according to the local state of some nodes [1,13]. In this case, we may call the predicate a *locally validated predicate*. We use the state-based predicate in this paper.

**Initialization**:
- Mining probability $p_{mine}$.
- Let $call_i \leftarrow \perp$ for any $i \in [n]$

---

On input $\mathcal{F}_{mine}(\mathsf{type}, val, i)$ from node $P_i$:
- If $call_i = \perp$, output $\mathsf{b} = 1$ with probability $p_{mine}$, or $\mathsf{b} = 0$ with probability $1 - p_{mine}$ and set $call_i = \mathsf{b}$.
- Else output $call_i$.

On input $\mathcal{F}_{mine}.\mathsf{verify}(\mathsf{type}, val, j)$ from node $P_i$:
- If $call_j = 1$, output 1, otherwise output 0.

Fig. 2: Functionality $\mathcal{F}_{mine}$. *val* can be $\perp$ or consists of multiple values.

– ***f*-External validity**: *Given a predicate $Q$, any honest node $P_i$ that terminates outputs a value $\boldsymbol{v_i}$ such that for each $\boldsymbol{v}_i[s] \neq \perp$, $Q(\boldsymbol{v_i}[s])$ holds by at least one honest node.*

**Protocol naming convention $\mathsf{PBC}_x^{\mathsf{y}}$.** To differentiate the protocols we study in this paper, we use the $\mathsf{PBC}_x^{\mathsf{y}}$ to denote a PBC protocol where each node provides an $x$-size input that is secure secure under $\mathsf{y}$ model. For example, $\mathsf{PBC}_1^{\mathsf{static}}$ denotes a 1-bit PBC assuming a static adversary.

### 2.2 Building Blocks

We review the building blocks. Due to space limitations, we provide detailed definitions and descriptions where relevant in our full paper.

**Aggregate signatures.** We assume recursively combinable aggregate signatures of size $O(\kappa + S \log n)$ when signed by $S$ nodes or $O(\kappa + n)$ using a bitmask, e.g., using BLS signatures based on pairings in the random oracle model [8] or a signature scheme and generic zero-knowledge proofs.

**The $\mathcal{F}_{mine}$ oracle.** We follow prior work [2, 11, 20] and define the $\mathcal{F}_{mine}$ ideal functionality that we use for random committee selection. $\mathcal{F}_{mine}$ is parameterised by the total number of nodes and a *mining* probability $p_{mine}$. $\mathcal{F}_{mine}$ provides two interfaces: $\mathcal{F}_{mine}$ and $\mathcal{F}_{mine}.\mathsf{verify}()$, as illustrated in Figure 2. For our static PBC, this can be implemented by nodes multicasting $O(\kappa)$-sized proofs using an SRS [2, 14]. For our adaptive PBC, we assume generic zero-knowledge proofs for composing signature aggregation and $\mathcal{F}_{mine}$ proofs (or for our protocols, proving committee membership), which also can be instantiated using an SRS [14].

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

**Erasure coding proof (ECP) system.** The idea of ECP [4] is to allow the encoder to prove succinctly and non-interactively that an erasure-coded fragment is consistent with a commitment to the original data block. Consider an $(m, n)$

erasure code that encodes a message $M$ into a set of $n$ fragments $d_1, d_2, \cdots, d_n$. An 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 $\pi_i$ that is around size $O(|M|/m + \kappa)$. Together, $\phi$ and $\pi_i$ convince node $P_i$ that $d_i$ is the correct data fragment for the message committed to by $\phi$.

An ECP system consists of three algorithms:

- **setup**. The **setup** algorithm receives a security parameter $\kappa$ and sets up the system parameters $\mathsf{pp}$.
- $\mathbf{prove}_{\mathsf{pp}}$. The $\mathbf{prove}_{\mathsf{pp}}$ algorithm takes as input a block of data $M$ and outputs $(\phi, \mathbf{d}, \boldsymbol{\pi})$ where $|\mathbf{d}| = |\boldsymbol{\pi}| = n$. Here, $\phi$ is a (computationally) binding commitment to all erasure-coded fragments $\mathbf{d} \leftarrow \mathsf{encode}(M)$, 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 $\phi$.
- $\mathbf{verify}_{\mathsf{pp}}$. The $\mathbf{verify}_{\mathsf{pp}}$ algorithm takes as input $(\phi, d_i, \pi_i)$ and outputs a bit. If $\mathbf{verify}_{\mathsf{pp}}(\phi, d_i, \pi_i) = 1$, then we say $d_i$ is a valid fragment w.r.t. $\phi$.

A secure ECP system achieves *EC-correctness* and *EC-consistency*. We use ECP-1 in this work, one of the two constructions provided in the paper. Under the trusted setup assumption (relying on a trusted setup to generate a powers-of-tau structured reference string [15] or SRS hereafter) and when the data block is of at least length $O(\kappa)$, 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 [10], 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. It consists of algorithms **gen**, **enc**, **dec** and **upd**, the first three as in standard PKE, and the last updating the secret key into a new *epoch*. We require an appropriate IND-CCA security notion where a challenge is made in epoch $j$, and the adversary has access to the secret key in any epoch $i > j$. FS-PKE can be implemented using pairings with $O(\kappa \log E)$-sized keys and constant-sized ciphertexts to support $E$ epochs [10].

## 3 $\mathsf{PBC}^*_L$: An Extension Protocol for $L$-bit PBC

### 3.1 Technical Overview

We present a new extension protocol for $L$-bit PBC. $\mathsf{PBC}^*_L$ reduces $L$-bit PBC to a $\kappa$-bit PBC $\mathsf{PBC}^*_\kappa$ and uses an ECP system. We only require that the $\kappa$-bit PBC protocol is transformed into a *validated* PBC by adding a locally validated predicate to $\mathsf{PBC}^*_\kappa$. We show that such a validated PBC can be easily achieved in our protocol (cf. Lemma 1). As described above, ECP works like an accumulator scheme for erasure coding and can be used to determine if a given fragment corresponds to the original data block. 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

**Global Parameters**:
- $M_i$ is the input of $P_i$. $\mathsf{ExtractedSet}_i \leftarrow [\emptyset]^n$.

---

**Phase 1**:
Every node $P_i$ performs the following:
- $\mathsf{ExtractedSet}_i^i \leftarrow M_i$. $(\phi_i, \mathbf{d}^i, \boldsymbol{\pi}^i) \leftarrow \mathbf{prove}_{\mathsf{pp}}(M_i)$.
- Send $(\textsc{Disseminate}, M_i)$ to all nodes.
- Query $\mathsf{PBC}_\kappa^*$ with predicate $Q$ and use $\phi_i$ as input.

**Phase 2**:
- Upon output $\phi_j$ for slot $j$ in $\mathsf{PBC}_\kappa^*$
  - If $P_i$ has previously received $(\textsc{Disseminate}, M_j)$ from $P_j$
    - $(\phi', \mathbf{d}, \boldsymbol{\pi}) \leftarrow \mathbf{prove}_{\mathsf{pp}}(M_j)$.
    - If $\phi' = \phi_j$,
      - For $\ell = 1, 2, \cdots, n$,
        - Send $(\textsc{Send}, d_\ell, \pi_\ell)$ to $P_\ell$.
- Upon receiving $(\textsc{Send}, d_i, \pi_i)$ from $P_\ell$,
  - If $\mathbf{verify}_{\mathsf{pp}}(\phi_j, d_i, \pi_i) = 1$, fix $d_i^*$ as $d_i$ and send $(\textsc{Echo}, d_i^*, \pi_i)$ to all nodes.
- Upon receiving $(\textsc{Echo}, d_\ell, \pi_\ell)$ from $P_\ell$,
  - If $\mathbf{verify}_{\mathsf{pp}}(\phi_j, d_\ell^*, \pi_\ell) = 1$, $B_j.add(d_\ell^*)$.
  - Upon $|B_j| \geq n - f$, set $M_j \leftarrow \mathsf{decode}(B_j)$ and $\mathsf{ExtractedSet}_i^j \leftarrow M_j$
- Output $\mathsf{ExtractedSet}_i$

Fig. 3: The $\mathsf{PBC}_L^*$ protocol. $Q$ is the locally validated-predicate evaluated within $\mathsf{PBC}_\kappa^*$ defined as follows: $Q(\phi_i)$ is valid for a node $P_j$ if, during execution, $P_j$ has previously received $M_i$ from $P_i$ such that for $(\phi, \mathbf{d}, \boldsymbol{\pi}) \leftarrow \mathbf{prove}(M_i)$ it holds that $\phi_i = \phi$.

rounds of communication. In contrast, the most communication-efficient extension protocol known so far (for BC rather than PBC) [19] incurs $O(n)$ rounds on top of the underlying $\kappa$-bit BC oracle.

Our extension protocol is secure under an adaptive adversary, as long as $\mathsf{PBC}_\kappa^*$ is adaptively secure. The same paradigm can also be extended to obtain a communication-efficient $L$-bit extension protocol for (non-parallel) BC.

### 3.2 The Extension Protocol

The pseudocode of our extension protocol is shown in Figure 3. Briefly speaking, each node first disseminates its input to all nodes. Then, it queries $\mathsf{PBC}_\kappa^*$ and uses the ECP commitment $\phi_i$ as the input. The predicate we add to $\mathsf{PBC}_\kappa^*$ ensures that if $\mathsf{PBC}_\kappa^*$ completes, at least one honest node holds $M_i$. Finally, after $\mathsf{PBC}_\kappa^*$ outputs some value, nodes reconstruct the data via two communication rounds.

It is worth mentioning that in our construction, we view ECP as a tailored and computation-efficient *proof* that proves the correctness of the encoding function of erasure coding. There exists more generic approach that achieves the same communication complexity as our approach, e.g., using zero knowledge proofs [12] to prove the correctness of the encoding function.

Our protocol has the following properties. Note that proofs of claims made hereafter are available in the full version of this work.

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

**Theorem 1.** *Assuming an SRS, the* $\mathsf{PBC}^*_L$ *protocol presented in Figure 3 satisfies* $f$*-validity and* $f$*-consistency with probability* $1 - \mathsf{negl}(\lambda)$.

**Theorem 2.** *The* $\mathsf{PBC}^*_L$ *protocol achieves* $O(n^2L + n^3\kappa + \mathcal{P}(\kappa))$ *communication and the round complexity is asymptotically the same as* $\mathsf{PBC}^*_\kappa$*, where* $\mathcal{P}(\kappa)$ *is the communication complexity of* $\mathsf{PBC}^*_\kappa$.

## 4 $\mathsf{PBC}^{\mathsf{static}}_\kappa$: Efficient PBC under a Static Adversary

### 4.1 Technical Overview

We present $\mathsf{PBC}^{\mathsf{static}}_\kappa$, a two-layer protocol that reduces the PBC problem to best effort broadcast and a $\mathcal{C}()$ protocol among $\lambda$ committee members. Although $\mathsf{PBC}^{\mathsf{static}}_\kappa$ itself can clearly be generalized to an $L$-bit PBC, we obtain a more efficient $L$-bit PBC integrating $\mathsf{PBC}^{\mathsf{static}}_\kappa$ with our extension protocol.

Our motivation is a tempting solution for committee sampling based protocols: the committee members can *reach an agreement* on some value and then convey the results to all nodes. While this is feasible for a system in the honest majority setting [2,16–18], there is no straightforward way to properly convey the results under a corrupt majority, an observation also made in prior works [11,21].

Our $\mathsf{PBC}^{\mathsf{static}}_\kappa$ protocol makes the above tempting solution work under a corrupt majority. We use a reduction from PBC to a so-called $\mathcal{C}()$ protocol, among $\lambda = O(\kappa)$ committee members. By carefully defining the security properties of $\mathcal{C}()$ and building an efficient construction from $L$-bit PBC among $\lambda$ committee members, we ensure that if an honest committee member sees some value output by $\mathcal{C}()$ for the first time, so does any other honest committee member. Accordingly, the maximum number of interactions between an honest node and committee members is bounded by a constant. In particular, $\mathcal{C}()$ takes as input a vector of sets of 'valid' (defined below) messages, and outputs a vector of sets corresponding to the union of valid messages that were input.

Briefly speaking, our protocol roughly works as follows. Each node first disseminates its input to all nodes. Then they proceed in a constant number of rounds. In each round, every node first sends its current received values to the committees. Then, the committees query the $\mathcal{C}()$ protocol. After $\mathcal{C}()$ terminates, committee members create signatures for the values they have seen and send them to all nodes. Finally, nodes merge the signatures they receive. At the end of the protocols, for each $P_s$, every honest node $P_i$ delivers some value from $P_s$ only after $P_i$ has received a sufficiently large number of signatures from the committee.

### 4.2 The $\mathcal{C}()$ Protocol

To achieve the goals mentioned above, the input of $\mathcal{C}()$ needs to be *validated* and the output needs to be *verifiable* [9]. For the protocol to be validated, the input

message $\boldsymbol{M}$ must satisfy a global predicate. In our case, for $\boldsymbol{M}$ to be validated, it must consist of $n$ vectors of valid $(r-1)$-s batches, as defined below. For the protocol to be verifiable, the output message, once sent to an honest node, should also be a valid $(r-1)$-s batches.

In our PBC protocol, each committee member $P_i$ receives $M_j$ from each node $P_j$, where $j \in [n] = \{1, 2, \cdots, n\}$ and $M_j$ is an $n$-value vector in the form of $[M_j^1, \cdots, M_j^n]$. Each $M_j^k$ is either $\perp$ or consists of up to two valid $(r-1)$-s batches. To facilitate the exposition of our protocol, we provide some definitions.

**Definition 2. (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 $u||s||\mathsf{SIG}_r$, where $u \in \{0,1\}^L$, $s \in [n]$, and $\mathsf{SIG}_r$ is a set of signatures that contains one signature from $P_s$ and $\frac{3r(\epsilon-\mu)(1-\epsilon)}{\mu^2} \log \frac{1}{\delta}$ 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 3 (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 $(u', j)$.*

We now specify the input and output of the $\mathcal{C}()$ protocol as follows. In some round $r$, the input of each node $P_i$ for the $\mathcal{C}()$ protocol is $\boldsymbol{M}$ which consists of up to $n$ vectors $\{M_1, \cdots, M_n\}$. Any $M_j \in \boldsymbol{M}$ is sent by node $P_j$. Each $M_j$ is validated if it is a valid tuple for round $r$. After running the $\mathcal{C}()$ protocol, each honest committee member $P_i$ outputs an $n$-value vector $\mathsf{Merged}_i$. $\mathsf{Merged}_i$ is verified if it is a valid tuple for round $r$. An interesting finding is that we can build $\mathcal{C}()$ from a $\kappa$-bit PBC among $\lambda$ committee members.

**Definition 4 ($t$-Secure $\mathcal{C}()$).** *Let $\mathcal{C}()$ be a protocol executed by $c$ nodes $\{P_1, \cdots, P_c\}$, as specified above. $\mathcal{C}()$ should satisfy the following properties for some round $r$ with probability $1 - \mathsf{negl}(\kappa)$ whenever at most $t$ nodes are corrupted.*

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

### 4.3 The $\mathsf{PBC}_\kappa^{\mathsf{static}}$ Protocol

We present the workflow of $\mathsf{PBC}_\kappa^{\mathsf{static}}$ in Figure 5. The protocol is round-based, starting from round 0 to round $R$ where $R = \lceil \frac{(1-\epsilon+\mu)c+1}{(\epsilon-\mu)c} \rceil = O(\frac{1}{\epsilon-\mu})$, i.e., a constant number. Each round consists of three *mini-rounds*, the first and the third using a constant number of message delays, and the second being $\mathcal{C}()$ executed by the committee ($O(\lambda)$ round complexity).

We optimize the 1st mini-round and reduce the communication from $O(n^2\kappa\lambda)$ to $O(n^2\kappa\lambda^K)$, where $0 < K < 1$ via a new sampling protocol. Additionally, using ECP as a building block, we can reduce the communication of the 3rd mini-round from $O(n^2\kappa\lambda)$ to $O(n^2\kappa + n\kappa\lambda)$. We provide a detailed description of the protocol and the optimizations in the full version of this work.

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

---

**Output conditions**
- After $\mathsf{PBC}_{L,c}^*$ terminates, return $\mathsf{Merged}_i$.

Fig. 4: The $\mathcal{C}()$ protocol.

**Theorem 3.** *Assuming a trusted PKI and SRS,* $\mathsf{PBC}_{\kappa}^{\mathit{static}}$ *is an* $f$*-Secure Parallel Broadcast protocol with probability* $1 - \mathsf{negl}(\lambda)$.

**Theorem 4.** *Assuming a trusted PKI and SRS, the* $\mathsf{PBC}_{\kappa}^{\mathit{static}}$ *protocol has* $O(\lambda)$ *round complexity and* $O(n^2\kappa\lambda^K + n\kappa\lambda^2 + \kappa\lambda^3 + \lambda^4)$ *communication complexity.*

## 5 $\mathsf{PBC}_{\kappa}^{\mathsf{adaptive}}$: Efficient PBC under an Adaptive Adversary

We present our adaptively-secure PBC protocol that, for $\kappa$-sized messages, has a communication complexity of $\tilde{O}(n^2\kappa^2 + n\kappa^3)$ given $\kappa = O(\lambda)$, or in general $\tilde{O}(n^2\kappa\lambda + n^2\lambda^2 + n\kappa\lambda^2 + n\lambda^3)$. By direct application of our adaptive extension protocol $\mathsf{PBC}_L^*$ from §3, we obtain an $L$-bit protocol with a communication complexity of $O(n^2L + n^3\kappa) + \tilde{O}(n^2\kappa^2 + n\kappa^3)$ bits.

### 5.1 Review of TrustedPBC

We briefly review TrustedPBC [20], our starting point of $\mathsf{PBC}_{\kappa}^{\mathsf{adaptive}}$. Every node $P_i$ holds a bit $b_i$ as input and outputs a vector of values $\boldsymbol{v}_i$. For each bit $b_j$ and slot $j$, $r$ signatures on $[b_j, j]$ (including one from $P_j$) from the committee members are collectively called a valid $r$-batch on $[b_j, j]$. The protocol is round-based and proceeds as follows.

- In round $r = 1$, every node $P_i$ creates a signature for $[b_i, i]$ and sends to all nodes. Each node now holds a vector of valid 1-batches, denoted as $M_i$, which continues to be updated throughout the protocol.
- In rounds $r = 2, \cdots, 2\kappa/\epsilon$, there are two mini-rounds in each round.
  - In the first mini-round, each node $P_i$ executes a $\mathcal{M}$-DistinctConverge protocol, the idea being for honest nodes to disseminate their received valid $r$-batches to all other nodes. Their implementation of $\mathcal{M}$-DistinctConverge uses $\tilde{O}(n^2\kappa^3)$ total communication and $\lceil \log(\epsilon \cdot n) \rceil$ rounds. It is an iterative gossiping protocol, where each step a node sends each message it propagates to $O(\kappa)$ other nodes, where the number of nodes that has seen a given message doubles each round (thus converging in a logarithmic number of rounds).

**Global Parameters**:
- Let $u_i$ be the input of $P_i$. Set $\mathsf{ExtractedSet}_i \leftarrow [\emptyset]^n$ and $\mathsf{VotedSet}_i \leftarrow [\emptyset]^n$.

**Round 0**:
- $\mathsf{ExtractedSet}_i^i \leftarrow u_i$. Send $(\textsc{Sign}, u_i, \sigma_i)$ to all where $\sigma_i$ is a signature on $[u_i, i]$.
- Upon receiving a $(\textsc{Sign}, u_j, \sigma_j)$ from $P_j$, add $\sigma_j$ to $\mathsf{Received}_i^j$.
- Query $b \leftarrow \mathcal{F}_{mine}(\mathsf{static}, i)$, if $b = 1$, broadcast $(\textsc{Com}, i)$ to all nodes.
- Upon receiving $(\textsc{Com}, j)$ s.t. $\mathcal{F}_{mine}.\mathsf{verify}(\mathsf{static}, j)$=1, add $P_j$ to committee.

**Round** $r = 1, \cdots, R$: each round has three mini-rounds.

**1st mini-round**: Every node $P_i$ performs the following:
- Send $(\textsc{Echo}, \mathsf{Received}_i)$ to all committee members
- Upon receiving $(\textsc{Echo}, \mathsf{Received}_j)$ from $P_j$, $\boldsymbol{M}[j] \leftarrow \mathsf{Received}_j$.

**2nd mini-round**: Every committee member queries $\mathcal{C}(\boldsymbol{M})$ and obtains $\mathsf{Merged}_i$.

**3rd mini-round**: Set $\mathsf{Received}_i \leftarrow [\perp]^n$. For each $s \in [n]$:

Every committee member $P_i$ performs the following:
- Send $(\textsc{Send}, \mathsf{Merged}_i)$ to all.
- If a valid $(r-1)$-s batch on $[u_i^s, s]$ is included in $\mathsf{Merged}_i^s$ but $u_i^s \notin \mathsf{VotedSet}_i^s$:
  - Set $\mathsf{VotedSet}_i^s \leftarrow \mathsf{VotedSet}_i^s \cup u_i^s$, create a signature for $[u_i^s, s]$ and send to all.
- If at least two valid $(r-1)$-s batches on different pairs $[u_i^s, s]$ and $[v_i^s, s]$ are included in $\mathsf{Merged}_i^s$ and $u_i^s, u_j^s \notin \mathsf{VotedSet}_i^s$:
  - Extract the first two of valid $(r-1)$-s batches and send to all.

For every node $P_i$, upon receiving valid $(\textsc{Send}, \mathsf{Merged}_j)$
- Merge the $(r-1)$-s batches from the $(\textsc{Send})$ messages into an $n$-value vector $\mathsf{Received}_i$ s.t. each $\mathsf{Received}_i^s$ contains at most two valid $r$-s batches.
- If there exists $u_j^s$ s.t. a valid $r$-s batch for $[u_j^s, s]$ is included in $\mathsf{Received}_j^s$ and $u_j^s \notin \mathsf{ExtractedSet}_i^s$ and $|\mathsf{ExtractedSet}_i^s| < 2$, $\mathsf{ExtractedSet}_i^s \leftarrow \mathsf{ExtractedSet}_i^s \cup u_j^s$.

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

Fig. 5: The $\mathsf{PBC}_\kappa^{\mathsf{static}}$ protocol. $R = \lceil \frac{(1-\epsilon+\mu)c+1}{(\epsilon-\mu)c} \rceil$.

  - In the second mini-round, whenever an honest committee member $P_i$ observes a valid $r$-batch in $M_i$ for the first time for slot $j$, it creates a digital signature for $[b_j, j]$, appends the signature to the valid $r$-batch, and sends to all nodes. Upon receiving a valid $(r+1)$-batch on $[b_j, j]$, each node $P_i$ updates its $M_i$ and adds $b_j$ to $\mathsf{ExtractedSet}_i^s$.
- At the end of the protocol, for each slot $j$, if there is only one value $b_j$ in $\mathsf{ExtractedSet}_i^j$, $P_i$ sets $\boldsymbol{v}_i[j]$ as $b_j$. Otherwise, $P_i$ sets $\boldsymbol{v}_i[j]$ as a canonical bit $\perp$. Finally, $P_i$ outputs the vector $\boldsymbol{v}_i$.

### 5.2 An Improved $\mathcal{M}$-DistinctConverge Protocol

Towards describing our protocol, we first formally define the aforementioned $\mathcal{M}$-DistinctConverge problem below.

**Definition 5 ($\texttt{distinct}_k$ function).** *For any set $M$, $\texttt{distinct}_k(M)$ is a subset of $M$ that contains all messages in $M$ with distinct $k$-bit prefixes.*[1]

**Definition 6 ($t$-secure $\mathcal{M}$-DistinctConverge protocol).** *Let $\mathcal{M} \subseteq \{0,1\}^*$ be an efficiently recognizable set. A protocol $\Pi$ executed by $n$ nodes, where every honest node $P_i$ initially holds input set $M_i \subseteq \mathcal{M}$ and constraint set $C_i \subseteq \mathcal{M}$, is a* $t$-secure $\mathcal{M}$-DistinctConverge protocol *if all remaining honest nodes upon termination, with probability $1 - \mathsf{negl}(\kappa)$, output a set*

$$S_i \supseteq \texttt{distinct}_k \left( \bigcup_{P_i \in \mathcal{H}} M_i - \bigcup_{P_i \in \mathcal{H}} C_i \right),$$

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

We build a new $\mathcal{M}$-DistinctConverge protocol (that improves the same function in TrustedPBC) to reduce the overall communication from $\tilde{O}(n^2\kappa^4)$ to $\tilde{O}(n^2\kappa^3)$. The novelty is to provide a new way for nodes to disseminate the messages in each round via a more efficient *sampling* approach. We first present a Propagate sub-protocol, and define its ideal functionality in Figure 6.

Let $n$ be the number of nodes and $m = 10/\epsilon + \kappa$. For every node $i \in [n]$, $\mathcal{F}_{\mathsf{prop}}$ keeps a set $O_i$ which is initialized to $\emptyset$. Let $M_i$ be node $i$'s input messages' set.

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

Fig. 6: Functionality $\mathcal{F}_{\mathsf{prop}}$.

We show the pseudocode of our Propagate protocol in Figure 7. In Propagate, nodes send in a given round a set of messages to $O(\lambda)$ nodes. Like [20], to achieve adaptive security, message lists are encrypted and padded to the same size, preventing the adversary from learning who sent what and, e.g., blocking a single message from being propagated. [20] implement Propagate in two rounds, where in the first round nodes sample fresh public keys that are then used in the second round. To improve concrete efficiency, our protocol reduces this to one round using *forward-secure public-key encryption* [10], since the public key is fixed at initialisation and only secret keys are evolved in a one-way fashion each time Propagate is called, which suffices for security.

[1] For example, for $M = \{01001, 01111, 11000, 10000\}$ we have that $\texttt{distinct}_2(M) = \{01001, 11000, 10000\}$. Note that $\texttt{distinct}_k$ is an one-to-many function, e.g., $\texttt{distinct}_2(M)$ is also $\{01111, 11000, 10000\}$.

**Input:** A set of messages $M_i$.
**Output:** A set of messages $O_i$.
**Global Parameters**:
- $\mathsf{PK}$ is a set of FS-PKE public keys inherited from the higher-level protocol.
- $\mathsf{sk}_i$ is a FS-PKE secret key inherited by the higher-level protocol.

---

**Upon** $\mathsf{Propagate}(M_i)$
- If $|M_i| \leq \log n$:
  - For $\ell \in [n]$: Send $(\textsc{Noenc}, M_i)$ to $P_\ell$.
- Else: // $\{\text{let } \Lambda_p = 2m|M_i|/n\}$
  - For $j \in [n]$:
    - For $x \in M_i$:
      - Add $x$ to list $\mathcal{L}_j$ with probability $m/n$.
    - If $|M_p| \leq n \log n/\lambda$ and $|\mathcal{L}_j| > \Lambda_p$, set $\mathcal{L}_j \leftarrow \perp$ and $j \leftarrow j-1$ // $\{\text{resample}\}$
  - For $j \in [n]$:
    - Pad $\mathcal{L}_j$ to size $\Lambda_p$.
    - If $(\mathsf{pk}, j) \in \mathsf{PK}$: $\mathsf{ct}_j \leftarrow \mathbf{enc}(\mathsf{pk}, e, \mathcal{L}_j)$.
    - Erase $\mathcal{L}_j$ from memory.
    - If $(\mathsf{pk}, j) \in \mathsf{PK}$: Send $\mathsf{ct}_j$ to $P_j$.

**Upon** $\Delta$ time after invoking $\mathsf{Propagate}(M_i)$:
- For all $\mathsf{ct}_j$ received from $P_j$:
  - $\mathcal{L}_j \leftarrow \mathbf{dec}(\mathsf{pk}, e, \mathsf{sk}_e, \mathsf{ct})$.
  - If $\mathcal{L}_j \neq \perp$: Add $\mathcal{L}_j$ to $O_i$.
- $\mathsf{sk}_{e+1} \leftarrow \mathbf{upd}(\mathsf{pk}_i, e+1, \mathsf{sk}_e)$. and erase $\mathsf{sk}_e$ from memory.
- For all $(\textsc{Noenc}, M_j)$ received from $P_j$ s.t. $\mathsf{ct}_j$ was not received from $P_j$:
  - Add $M_j$ to $O_i$.
- Output $O_i$.

Fig. 7: Our new $\mathsf{Propagate}$ protocol, an instantiation of the propagation process.

The challenge is then to ensure that honest nodes receive all messages with overwhelming probability while reducing communication. We use a *resampling* technique to address the issue. In particular, if any list exceeds a predetermined size (which is approximately $\kappa\times$|size of the set of messages to be propagated|$/n$), then the node resamples the list until it does not exceed that size. In our case, we allow nodes to resample their lists $O(n)$ times in the worst case to keep the padding size to a minimum. By contrast, [20] pads lists to a predetermined maximum size in all cases, thereby incurring $O(\lambda)$ more communication than us.

**Lemma 2.** *If $M$ is the input set of node $P$ in* $\mathsf{Propagate}$*, the size of each list $\mathcal{L}_j$ is:*

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

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

**Input:** Sets of messages $M_i \subseteq \mathcal{M}$ and $\mathcal{C}_i \setminus \mathcal{M}$, and integer $k$.
**Output:** A set of messages $S_i$.

---

**Round** $r = 1, \cdots, \lceil \log(\epsilon n) \rceil$:
- $O_i \leftarrow \mathcal{F}_{\mathsf{prop}}(\mathsf{SendRandom}, \mathtt{distinct}_k(M_i - \mathcal{C}_i))$.
- For $\ell \in [n]$: Send $(\textsc{Noenc}, M_i)$ to $P_\ell$.
- $\mathsf{Local}_i \leftarrow \mathsf{Local}_i \cup O_i$.
- $\mathcal{C}_i \leftarrow \mathcal{C}_i \cup M_i$.
- $M_i \leftarrow \mathsf{Local}_i \cap \mathcal{M}$.

Output $M_p$.

Fig. 8: $\mathcal{M}$-DistinctCV protocol [20, Fig. 7] implementing $\mathcal{M}$-DistinctConverge.

**Lemma 3.** *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.* $1 - \mathsf{negl}(\lambda)$*:*

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

**Lemma 4.** *Assuming a FS-PKE scheme,* Propagate *is a secure instantiation of the $F_{prop}$ functionality.*

**Implementing $\mathcal{M}$-DistinctConverge.** We use the $\mathcal{M}$-DistinctCV [20, Fig. 7] protocol in our work, besides that we now use our new Propagate protocol, which we provide pseudocode for in Figure 8. Here, 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 $\mathsf{Local}_i$. $M_i$ 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 $\mathsf{Local}_i \cap \mathcal{M}$. Ultimately, set $M_i$ is returned.

**Lemma 5.** DistinctCV *is an adaptively $f$-secure* $\mathcal{M}$-DistinctConverge *protocol, for $f < (1-\epsilon)n$. The number of bits sent over all nodes is*

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

*where $CC(\mathsf{Propagate}(M_i^l \setminus C_i^l))$ denotes the communication cost of node $P_i$ calling* $\mathsf{Propagate}((M_i^l \setminus C_i^l)$*. Moreover,* DistinctCV *has $O(\log n)$ round complexity.*

### 5.3 The $\mathsf{PBC}_\kappa^{\mathsf{adaptive}}$ Protocol

We are finally ready to present our $\mathsf{PBC}_\kappa^{\mathsf{adaptive}}$ protocol. Note that we have previously defined the notion of a valid $r$-s batch for our static PBC protocol in §4. We need an appropriate notion of a valid *$r$-batch* as defined below.

**Input:** Each node $P_i$ inputs $\kappa$-bit value $u_i$.
**Output:** Each node $P_i$ outputs an $n$-valued vector $\mathsf{out}_i$.
**Global Parameters**:
- $\mathsf{ExtractedSet}_i \leftarrow [\emptyset]^n$, $\mathsf{VotedSet}_i \leftarrow [\emptyset]^n$, $\mathsf{Local}_i \leftarrow \emptyset$.
- $\mathsf{PK} \leftarrow \emptyset$, a set of FS-PKE public keys. $\mathsf{sk}_e \leftarrow \perp$, $P_i$'s FS-PKE secret key.

---

**Round 0**:
- $(\mathsf{pk}_i, \mathsf{sk}_e) \leftarrow$ **gen** $(e = 0)$.
- Send $(\textsc{Sign}, u_i, \sigma_i)$ and $(\textsc{Key}, \mathsf{pk}_i)$ to all nodes ($\sigma_i$ is a signature on $[u_i, i]$).
- Upon receiving $(\textsc{Sign}, u_j, \sigma_j)$ from $P_j$, add $\sigma_j$ to $\mathsf{Received}_i^j$.
- Upon receiving $(\textsc{Key}, \mathsf{pk}_j, \sigma_j)$ from $P_j$, add $(\mathsf{pk}_j, j)$ to $\mathsf{PK}$.

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

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

**Vote**:

If $r \leq R$:
- Find all $u_j \notin \mathsf{ExtractedSet}_i^j$ in $\mathsf{Local}_i$ with valid $r$-batches.
- For each such $u_j$ s.t. $\mathcal{F}_{mine}(\mathsf{adaptive}, u_j, i) = 1$ (and $|\mathsf{ExtractedSet}_i^j| \leq 1$) :
  - Add $u_j$ to $\mathsf{VotedSet}_i^j$ and $\mathsf{ExtractedSet}_i^j$
  - 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 $u_j \in \mathsf{ExtractedSet}_i^j$ if $|\mathsf{ExtractedSet}_i^j| = 1$, or $\perp$ else.

Fig. 9: The $\mathsf{PBC}_\kappa^{\mathsf{adaptive}}$ protocol.

**Definition 7 ($(u, j)$-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 $\mathcal{F}_{mine}$ is queried on input $\mathcal{F}_{mine}$.verify(adaptive,u,j), $\mathcal{F}_{mine}$ outputs 1.*

**Definition 8.** *Let $\mathcal{M}_r$ denote the set of all possible valid $r$-batches for all $m \in \{0,1\}^\kappa$ and for all $s \in [n]$.*

**Lemma 6.** *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 $\mathtt{distinct}_{k^*}$ is defined in Definition 5. Then $|\mathtt{distinct}_{k^*}(\mathcal{M}_r)| = 2{\cdot}n$.*

**Definition 9 (Valid $r$-batch).** *A valid $r$-batch on pair $(u, j)$ is the element*

$$u||j||\mathsf{SIG}_r,$$

*where $\mathsf{SIG}_r$ 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 $(u, j)$-committee (resp. or an aggregate signature with the contributions of $P_j$ and at least $r - 1$ other nodes in the $(u, j)$-committee).*

Our protocol (Figure 9) follows the template of TrustedPBC of [20], which itself follows the template of the broadcast protocol of Chan et al. [11], save for the following notable changes.

First, TrustedPBC is defined only for single-bit PBC. Therefore, we generalize it for multiple nodes, the main difference coming from our use of $\mathcal{F}_{mine}$. Abstractly, there are an exponential number of possible committees, one per message/slot pair (this number is quadratic in $n$ for TrustedPBC), but since $\mathcal{F}_{mine}$ 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.

Note that at the beginning of protocol execution, nodes send to all nodes their input value $u_i$ and a signature $\sigma_i$. Recall in $\mathsf{Propagate}$ 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.

Then, as shown in Figure 9, 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 a set $\mathsf{ExtractedSet}_i^j$. 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 $\mathcal{F}_{mine}$. If so, and they have not previously added their signature to the $r$-batch, they do so. Finally, at the end of $R = O(\lambda)$ rounds, nodes output a vector of values for each node $P_j$, which is $\perp$ if $|\mathsf{ExtractedSet}_i^j| \neq 1$ and the message in $\mathsf{ExtractedSet}_i^j$ otherwise.

**Theorem 5.** *Assuming a trusted PKI and SRS, protocol $\mathsf{PBC}_\kappa^{\mathsf{adaptive}}$ satisfies $f$-consistency with probability $1 - \mathsf{negl}(\kappa) - \mathsf{negl}(\lambda)$.*

**Theorem 6.** *Assuming a trusted PKI and SRS, protocol $\mathsf{PBC}_\kappa^{\mathsf{adaptive}}$ satisfies $f$-validity with probability $1 - \mathsf{negl}(\kappa)$.*

**Theorem 7.** *Protocol $\mathsf{PBC}_\kappa^{\mathsf{adaptive}}$ has $O(\kappa \log n)$ round complexity and a communication complexity of $O(\log \epsilon n(n^2\lambda\kappa + n\lambda^2\kappa + n\lambda^3 \log n + n^2\lambda^2 \log n))$.*

## Acknowledgments

Sisi and Haochen were supported in part by the Beijing Natural Science Foundation under M23015 and National Natural Science Foundation of China under

92267203. Daniel and Giorgos completed part of this work while visiting CISPA, and Daniel while at EPFL, Purdue University and Georgia Institute of Technology, and was supported in part by Sunday Group, Inc. and AnalytiXIN.

## References

1. Abraham, I., Asharov, G., Patra, A., Stern, G.: Perfectly secure asynchronous agreement on a core set in constant expected time. In: TCC (2024)
2. Abraham, I., Chan, T.H., Dolev, D., Nayak, K., Pass, R., Ren, L., Shi, E.: Communication complexity of byzantine agreement, revisited. In: PODC (2019)
3. Abraham, I., Nayak, K., Shrestha, N.: Communication and round efficient parallel broadcast protocols. Cryptology ePrint Archive (2023)
4. Alhaddad, N., Duan, S., Varia, M., Zhang, H.: Succinct erasure coding proof systems. Cryptology ePrint Archive (2021)
5. Asharov, G., Chandramouli, A.: Perfect (parallel) broadcast in constant expected rounds via statistical vss. In: EUROCRYPT. pp. 310–339. Springer (2024)
6. Blum, E., Boyle, E., Cohen, R., Liu-Zhang, C.D.: Communication Lower Bounds for Cryptographic Broadcast Protocols. In: DISC (2023)
7. Blum, E., Katz, J., Liu-Zhang, C.D., Loss, J.: Asynchronous byzantine agreement with subquadratic communication. In: TCC. Springer (2020)
8. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the weil pairing. In: International conference on the theory and application of cryptology and information security. pp. 514–532. Springer (2001)
9. Cachin, C., Kursawe, K., Petzold, F., Shoup, V.: Secure and efficient asynchronous broadcast protocols. In: CRYPTO. pp. 524–541. Springer (2001)
10. Canetti, R., Halevi, S., Katz, J.: A forward-secure public-key encryption scheme. Journal of Cryptology **20**, 265–294 (2007)
11. Chan, T.H.H., Pass, R., Shi, E.: Sublinear-round byzantine agreement under corrupt majority. In: PKC. pp. 246–265. Springer (2020)
12. Civit, P., Gilbert, S., Guerraoui, R., Komatovic, J., Monti, M., Vidigueira, M.: Every bit counts in consensus. DISC (2023)
13. Duan, S., Wang, X., Zhang, H.: Fin: Practical signature-free asynchronous common subset in constant time. In: ACM CCS (2023)
14. Groth, J.: On the size of pairing-based non-interactive arguments. In: Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35. pp. 305–326. Springer (2016)
15. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: ASIACRYPT (2010)
16. Katz, J., Koo, C.Y.: On expected constant-round protocols for byzantine agreement. In: CRYPTO. pp. 445–462 (2006)
17. King, V., Saia, J.: Breaking the $o(n^2)$ bit barrier: scalable Byzantine agreement with an adaptive adversary. JACM **58**(4), 18 (2011)
18. King, V., Saia, J., Sanwalani, V., Vee, E.: Scalable leader election. In: SODA (2006)
19. Nayak, K., Ren, L., Shi, E., Vaidya, N.H., Xiang, Z.: Improved extension protocols for byzantine broadcast and agreement. DISC (2020)
20. Tsimos, G., Loss, J., Papamanthou, C.: Gossiping for communication-efficient broadcast. In: CRYPTO (2022)
21. Wan, J., Xiao, H., Shi, E., Devadas, S.: Expected constant round byzantine broadcast under dishonest majority. In: TCC. pp. 381–411. Springer (2020)