# Secure Causal Atomic Broadcast, Revisited

Sisi Duan
Oak Ridge National Laboratory
Email: duans@ornl.gov

Michael K. Reiter
University of North Carolina at Chapel Hill
Email: reiter@cs.unc.edu

Haibin Zhang
University of Connecticut
Email: haibin.zhang@uconn.edu

***Abstract*—We revisit the problem of preserving causality in Byzantine fault-tolerant (BFT) atomic broadcast protocols, a requirement first proposed by Reiter and Birman (TOPLAS 1994). While over the past three decades, this requirement has been met through the deployment of expensive public-key threshold cryptosystems, we propose three novel, secure causal BFT protocols without using public-key cryptography. We implement and evaluate these protocols, showing that they significantly outperform existing constructions that use threshold cryptosystems.**

## I. Introduction

State machine replication [48, 61] is a fundamental software approach to enabling highly available services in practical distributed systems and cloud computing platforms (*e.g.,* Google's Chubby [18] and Spanner [28], Apache ZooKeeper [42]). Despite its great success, it is extremely difficult to enable confidentiality—an essential goal for secure outsourcing and computation—in such an approach (without sacrificing functionality). In fact, the confidentiality problem has become a major obstacle to a wider adoption of cloud storage and cloud computing where replication plays a central goal [54].

In this paper, we consider one specific confidentiality goal—causality preservation, a notion that dates back to early '90s [57] and is particularly relevant to modern cloud computing. Let us briefly describe the notion below.

In a replicated service, when a client issues a request, some faulty replicas may create a new request which will be executed before the original request of the client. This will violate the *causal order* of client requests. For example, Reiter and Birman [57] considered a trading service that trades stocks. When a client issues a request to purchase stock shares, a faulty replica may be able to collude with another corrupt client to issue a derived request for the same stock. If the correct replicas deliver the new request before the request of the correct client, the new request may adjust the demand for the stock and the service may raise the price to the correct client. Another example is to consider a service that registers names (*e.g.,* DNS service) in a "first come, first served" manner [19, 24]. A faulty replica may observe an interesting name being registered, and it may then register the name for another client, thereby violating the causality of the client requests.

It is important to note that causality preservation with malicious parties is a confidentiality related notion, and in fact a rare one that only makes sense for a replicated service, but not for one provided by a centralized server. Moreover, causality preservation is indeed desirable in practical distributed systems: ZooKeeper [42] is one such system that achieves both total order and causal order in the crash failure model.

Reiter and Birman's construction [57] combines an atomic broadcast protocol and a public-key threshold cryptosystem [64]. The clients encrypt their requests with the threshold cryptosystem. Replicas run the underlying atomic broadcast protocol to reach an agreed total order on the ciphertexts. Then each replica uses its corresponding decryption key to generate its decryption share which is broadcast to the rest of the replicas. Each replica waits for enough decryption shares to recover and then deliver the client request. Cachin, Kursawe, Petzold, and Shoup (CKPS) [21] refined and formalized the notion of secure causal atomic broadcast and built a provably secure construction from a labeled chosen-ciphertext-attack (CCA) secure threshold cryptosystem [64]. Recently, Duan and Zhang [35] presented a confidential BFT protocol that also achieves causality preservation using only symmetric cryptography. However, their construction lies outside our model, because it relies on a specific architecture (that separates agreement from execution) and requires a larger number of nodes.

While these existing constructions use encryption as the key component, we propose three novel and efficient constructions without explicitly using encryption. When designing these protocols, we keep three goals in mind—*provable security*, *generality*, and *efficiency*. All of our protocols are provably secure in the definitions that we formalize: we provide not only general and expressive frameworks based on generic primitives, but efficient and secure instantiations.

**Our contributions.** We make the following contributions:

- We extend the definitional framework of CKPS [21] to include more general scenarios for secure causal BFT protocols.
- We provide a generic framework for a secure causal protocol. It can be based on any fair BFT protocol [27, 34] and any *non-malleable commitment with associated-data*, a new primitive we define. We offer an efficient instantiation (CP1) which is secure in the random oracle model (ROM).
- We go on to study the case of benign clients subject to crash failures. In this case, we also provide a generic framework that is built on top of a new and generic distributed cryptographic primitive—*asynchronous robust*

*secret sharing*. We suggest two concrete instantiations (CP2 and CP3) that are even more efficient than CP1.
- We implemented and evaluated all these protocols. We show that our protocols significantly outperform the existing constructions based on threshold cryptosystems [21, 23].

While our primary goal is to build efficient secure causal BFT protocols, our work also provides a firm foundation for this primitive. Existing secure causal BFT protocols based on threshold cryptosystems can only be realized using specific number-theoretic assumptions (*e.g.,* the Decisional Diffie-Hellman (DDH) assumption). However, because of our general frameworks, such protocols can be built from generic primitives such as a one-way function, or be even information-theoretically secure.

## II. Related Work

**Atomic broadcast and Byzantine fault-tolerant protocols.** According to the recent literature, the concept of atomic broadcast in the presence of Byzantine failures is more used to describe consensus-based protocols that do not explicitly distinguish clients and servers [21, 52], while the notion of Byzantine fault-tolerant (BFT) protocols is more used to describe state machine replication with Byzantine replicas (and Byzantine clients) [1, 4, 26, 27, 29, 33, 44]. Throughout the paper, we do not distinguish atomic broadcast protocols from BFT protocols and use them interchangeably.

Our constructions (CP1, CP2, and CP3) are all general and can be built from any types of BFT protocols. Namely, the underlying BFT protocols may be consensus-based or sequencer-based, and they may work in synchronous, partially asynchronous, or asynchronous environments. (See [30] for a comprehensive survey.) However, our constructions have more visible performance improvements for the recent sequencer-based BFT state machine replication protocols using symmetric cryptography [1, 4, 26, 27, 29, 33, 44], compared to consensus-based ones that employ threshold cryptography [21].

**Causal order in crash fault-tolerant distributed systems.** The notion of causality in distributed systems in the crash failure model was first introduced by Lamport [47] and later extended by Lamport [49]. The formalization is centered on the "happened before" or "precedes" relation. Since then, a number of causal broadcast protocols that tolerate crash failures [14, 36, 46, 62] were proposed.

**Confidentiality in distributed systems.** Confidentiality is a central goal in dependable distributed programs. There are many types of confidentiality notions though.

A line of work aims at achieving confidentiality in distributed file systems or storage systems that only support *store* and *retrieve* operations [2, 12, 25, 38, 43, 45, 60]. This is achieved by letting clients apply encryption or secret sharing to the data before the data is uploaded to the system. The servers neither see the underlying data nor process the data. Belisarius [53] additionally supports *add* operations by leveraging the additive homomorphic property of the underlying secret sharing scheme.

However, it is highly challenging to support arbitrary operations while maintaining confidentiality and availability of the user data in distributed systems. Despite significant progress in secure outsourcing and computation, it has largely focused on the case of a single server (*e.g.,* fully homomorphic encryption [39], ORAM [40]), or generic and inefficient multi-party computation (*e.g.* [11]). There are two notable works [35, 65] that achieve reliability and confidentiality simultaneously, yet requiring a special architecture support. The first is a confidential BFT protocol by Yin, Martin, Venkataramani, Alvisi, and Dahlin [65]. Their protocol leverages the idea of separating agreement from execution and uses expensive threshold signatures. Assuming the same architecture, Duan and Zhang [35] provided a much more efficient construction that uses only symmetric encryption. Moreover, both protocols require a lot more nodes than a conventional BFT protocol. Duan and Zhang also proved that their confidential BFT protocol implies a secure casual BFT protocol. Nevertheless, the implication applies only to their specific architecture.

**Symmetric cryptography *vs.* threshold cryptography.** Cachin and Poritz [23] built a system, SINTRA, which consists of an implementation of a secure causal atomic broadcast protocol [21]. As reported by the authors [23, pp. 9], their atomic broadcast protocol using extensive threshold cryptography is several orders of magnitude slower than PBFT [26] (in the LAN setting). The evaluation on a confidential BFT [35] and a BFT storage system [53] demonstrated a similar result: the protocols using only symmetric cryptography are several orders of magnitude faster than the ones using expensive threshold signatures [65]. Our evaluation is in line with these prior findings: our constructions CP1, CP2, and CP3 add little overhead to the underlying BFT protocol, and they all significantly outperform the one based on threshold cryptosystems (CP0) in both the LAN and WAN settings.

## III. Preliminaries and System Model

**Notation.** If $n$ is an integer then $[1..n]$ denotes the set $\{1, \cdots, n\}$. If $S$ is a set then $s \xleftarrow{\$} S$ denotes the operation of selecting an element $s$ of $S$ uniformly at random. If $\mathcal{A}$ is a randomized algorithm then we write $z \xleftarrow{\$} \mathcal{A}(x, y, \cdots)$ to indicate the operation that runs $\mathcal{A}$ on inputs $x, y, \cdots$ and fresh and uniformly random coins, and outputs $z$.

**BFT protocols.** We consider Byzantine fault-tolerant (BFT) protocols (*i.e.* atomic broadcast protocols), where $f$ out of $n$ replicas can behave arbitrarily and a computationally bounded adversary can coordinate faulty replicas to compromise the system. A BFT protocol may be sequencer-based [26, 44] or consensus-based [21, 52]. A secure BFT protocol should satisfy the following properties:

*Agreement.* If any correct replica delivers a message $m$, then every correct replica delivers $m$.

*Total order.* If a correct replica has delivered $m_1, m_2, \cdots, m_s$ and another correct replica has delivered $m'_1, m'_2, \cdots, m'_{s'}$, then $m_i = m'_i$ for $1 \leq i \leq \mathit{min}(s, s')$.

*Liveness.* If a message $m$ is input to $n - f$ correct replicas, then all correct replicas will eventually deliver $m$.

**Secure causal BFT protocols.** Input causality prevents the faulty replicas from creating a derived request that is delivered and executed before the client's request. It is important to note that it is equally unacceptable for the derived request to be identical to the client's request.

The problem of preserving input causality in BFT atomic broadcast protocols was first introduced by Reiter and Birman (RB) [57]. The notion was later refined by Cachin, Kursawe, Petzold, and Shoup (CKPS) [21].

We note that the definition of CKPS is not general enough to cover all possible scenarios or constructions. Specifically, CKPS explicitly models input causality using threshold encryption (as in RB), but there may exist constructions that do not exploit threshold encryption (as we show in this paper). To put it sightly differently, while CKPS uses an ("indistinguishability-style") secrecy notion for encryption, causality preservation seems more of a non-malleability notion [32] for a primitive that is not necessarily related to encryption.[1]

Therefore, we will extend the CKPS definitional framework to capture more general scenarios, and introduce appropriate security definitions tailored for different constructions.

**Fair BFT protocols** For one of our constructions, *fairness* plays a vital role. Fairness prevents the BFT service from unfairly delaying or dropping some clients' requests but not others. For instance, Aardvark [27] implements a fair and efficient sequencer-based BFT protocol. (The idea is that if the sequencer fails to enforce fairness, a view change will be triggered.) ByzID [34] is another fair BFT protocol (that uses small trusted components).

**Authenticated and private channels.** Throughout the paper, we assume authenticated channels which can be easily realized using message authentication codes (MACs). In our secret sharing based constructions, we additionally require private channels for the communication carrying secret shares. Authenticated and private channels only add very slight overhead to authentication-only channels, and the most efficient instantiation is to use authenticated encryption (with associated-data) [58, 59].

**ROM.** For some constructions in the paper, we use the random oracle model (ROM) [8], where scheme algorithms and adversary algorithms will have access to a random oracle.

## IV. Building Blocks

In this section, we review labeled threshold cryptosystems (for CP0), and introduce two new cryptographic primitives—non-malleable commitment with associated-data (for CP1) and asynchronous robust secret sharing (for CP2 and CP3).

[1]For encryption schemes, message secrecy and message non-malleability are equivalent (if considering chosen ciphertext attacks) [7, 10]. This may not be the case for other primitives. For instance, the non-malleability notion for a commitment scheme is orthogonal to its secrecy notion.

### A. Labeled Threshold Cryptosystems

We review robust labeled threshold cryptosystem (*i.e.* threshold encryption) [64] where a public key is associated with the system and a decryption key is shared among all the servers. Syntactically, a $(t, n)$ threshold encryption scheme ThreshEnc consists of the following algorithms. A probabilistic key generation algorithm $\mathsf{TGen}$ takes as input a security parameter $l$, the number $n$ of total servers, and threshold parameter $t$, and outputs (*pk*, *vk*, *sk*), where *pk* is the public key, *vk* is the verification key, and $sk = (sk_1, \cdots, sk_n)$ is a list of private keys. A probabilistic encryption algorithm $\mathsf{TEnc}$ takes as input a public key *pk*, a message $m$, and a label $lb$, and outputs a ciphertext $c$. A probabilistic decryption share generation algorithm $\mathsf{ShareDec}$ takes as input a private key $sk_i$, a ciphertext $c$, and a label $lb$, and outputs a decryption share $\sigma$. A deterministic share verification algorithm $\mathsf{Vrf}$ takes as input the verification key *vk*, a ciphertext $c$, a label $lb$, and a decryption share $\sigma$, and outputs $b \in \{0, 1\}$. A deterministic combining algorithm $\mathsf{Comb}$ takes as input the verification key *vk*, a ciphertext $c$, a label $lb$, a set of $t$ decryption shares, and outputs a message $m$, or $\perp$ (a distinguished symbol).

We require the threshold encryption scheme to be chosen ciphertext attack (CCA) secure against an adversary that controls up to $t - 1$ servers. We also require consistency of decryptions, *i.e.*, no adversary that controls up to $t-1$ servers can produce a ciphertext and two $t$-size sets of valid decryption shares such that they yield different plaintexts.

There are a few efficient constructions for (labeled) CCA secure threshold cryptosystems [5, 16, 64].

### B. Commitment Schemes

We will use a number of commitment schemes for different purposes, including conventional commitments, non-malleable commitments (NMC), non-malleable commitments with associated-data (NM-CAD), and concurrent non-malleable commitments.

**Conventional commitment scheme.** A commitment scheme is a protocol that allows one to commit to a chosen value. The value will remain hidden until the moment when the committer decides to open the commitment. A conventional commitment requires the regular hiding and binding properties.

**Non-malleable commitment with associated-data.** A scheme for (non-interactive) *non-malleable commitment with associated-data*, or NM-CAD, is a triple $\Pi = (\mathsf{Cgen}, \mathsf{Commit}, \mathsf{Open})$. $\mathsf{Cgen}$ takes as input a security parameter $l$ and outputs a commitment key $ck$. $\mathsf{Commit}$ takes as input the commitment key $ck$, a message $m \in \mathcal{M}$ (message space) and a header $h \in \mathcal{H}$ (header space) and outputs $(c, d)$, where $c$ is the commitment and $d$ is the decommitment or the opening; we write $\mathsf{Commit}^h_{ck}(m)$ for $\mathsf{Commit}(ck, h, m)$. $\mathsf{Open}$ takes as input $ck$, $h$, $c$, $m$, and $d$, and outputs a decision bit; we write $\mathsf{Open}^h_{ck}(c, m, d)$ for $\mathsf{Open}(ck, h, c, m, d)$.

We impose the conventional correctness requirement: for any $ck \in \mathsf{Cgen}(1^l)$, $m \in \mathcal{M}$, and $h \in \mathcal{H}$, if $\mathsf{Commit}^h_{ck}(m) = (c, d)$, then $\mathsf{Open}^h_{ck}(c, m, d) = 1$.

A syntactic difference between NM-CAD and the conventional commitment scheme is that NM-CAD supports associated-data (*i.e.,* header). The function of the associated-data resembles that of the label in threshold encryption—to distinguish instances of the protocols.

For our purpose, we require computational hiding, computational binding, and non-malleability with respect to opening and associated-data.

*Hiding.* It is computationally infeasible for any adversary $\mathcal{A}$ to output two messages $m_0, m_1 \in \mathcal{M}$ such that $\mathcal{A}$ can distinguish between their corresponding commitment values $c_0$, $c_1$. Formally, for any probabilistic polynomial-time (PPT) adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, we define $\mathbf{Adv}^{\text{hiding}}_{\Pi,\mathcal{A}}(1^l) = \Pr[b = b' | ck \xleftarrow{\$} \mathsf{Cgen}(1^l), (m_0, m_1, h, st) \leftarrow \mathcal{A}_1(ck), b \xleftarrow{\$} \{0,1\}, (c,d) \xleftarrow{\$} \mathsf{Commit}^h_{ck}(m_b), b' \xleftarrow{\$} \mathcal{A}_2(h, c, st)]$, where *st* is the state information.

*Binding.* It is computationally infeasible for any adversary $\mathcal{A}$ to output two different messages $m \neq m'$ to the same commitment $c$ and header $h$. Formally, for any PPT adversary $\mathcal{A}$, we define $\mathbf{Adv}^{\text{binding}}_{\Pi,\mathcal{A}}(1^l) = \Pr[(m \neq m') \wedge (m, m' \neq \perp) | ck \xleftarrow{\$} \mathsf{Cgen}(1^l), (h, c, m, m', d, d') \xleftarrow{\$} \mathcal{A}(ck), \mathsf{Open}^h_{ck}(c, m, d) = 1, \mathsf{Open}^h_{ck}(c, m', d') = 1]$.

*Non-malleability.* We require the commitment scheme to be non-malleable. There are two notions of non-malleability though: non-malleability *with respect to commitment* [32] and non-malleability *with respect to opening* [31]. The former requires that it is infeasible to find a commitment to a message which is related to another committed value. The latter requires that it is infeasible to open the modified commitment given the decommitment of the original commitment. The former is a strictly stronger notion [37].

Our application only needs the weaker notion—non-malleability with respect to opening. However, our specific construction meets the stronger one.

Our notion of non-malleability with respect to opening is an extension and generalization of the conventional one, because we require that non-malleability hold with respect to associated-data as well.

We provide a simulation-based definition as depicted in Fig. 1. We fix a PPT relation $\mathsf{R}$ which takes inputs from a space $\mathcal{M} \times \mathcal{M}$ and outputs a bit.

We consider a three-stage adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$. In the first stage, adversary $\mathcal{A}_1$ selects a message space $\mathcal{M}$ and a header space $\mathcal{H}$ from which a message $m$ and a header $h$ are sampled respectively. In the second stage, adversary $\mathcal{A}_2$ is given a hint $h_m$ on the message $m$ (as an output of a PPT function $hint(ck, \cdot)$) that represents some a-prior information that may be gathered by the adversary from other executions of the protocols. Note that we do not need to define a similar function for the header space $\mathcal{H}$, as the target header will be given to the adversary in cleartext. $\mathcal{A}_2$ now has to output a "new" commitment. In the last stage, adversary $\mathcal{A}_3$ is given the committed value and the opening of the original commitment and must output the corresponding values for the new commitment.

| $\mathbf{Exp}^{\text{nm-oad-0}}_{\Pi,\mathcal{A}}(l)$ | $\mathbf{Exp}^{\text{nm-oad-1}}_{\Pi,\mathcal{S}}(l)$ |
|---|---|
| $ck \xleftarrow{\$} \mathsf{Cgen}(1^l)$ | $ck \xleftarrow{\$} \mathsf{Cgen}(1^l)$ |
| $(\mathcal{M}, \mathcal{H}, \mathsf{s}_1) \xleftarrow{\$} \mathcal{A}_1(ck)$ | $(\mathcal{M}, \mathcal{H}, \mathsf{s}_1) \xleftarrow{\$} \mathcal{S}_1(ck)$ |
| $m \xleftarrow{\$} \mathcal{M}(1^l)$ | $m \xleftarrow{\$} \mathcal{M}(1^l)$ |
| $ht_m \xleftarrow{\$} hint(ck, m)$ | $ht_m \xleftarrow{\$} hint(ck, m)$ |
| $h \xleftarrow{\$} \mathcal{H}(1^l)$ | $h \xleftarrow{\$} \mathcal{H}(1^l)$ |
| $(c,d) \xleftarrow{\$} \mathsf{Commit}^h_{ck}(m)$ | $(c,d) \xleftarrow{\$} \mathsf{Commit}^h_{ck}(m)$ |
| $(h^*, c^*, \mathsf{s}_2) \xleftarrow{\$} \mathcal{A}_2(h, c, \mathsf{s}_1, ht_m)$ | $(h^*, c^*, \mathsf{s}_2) \xleftarrow{\$} \mathcal{S}_2(\mathsf{s}_1, ht_m)$ |
| $(m^*, d^*) \xleftarrow{\$} \mathcal{A}_3(m, d, \mathsf{s}_2)$ | $(m^*, d^*) \xleftarrow{\$} \mathcal{S}_3(m, d, \mathsf{s}_2)$ |
| **return** 1 iff | **return** 1 iff |
| $\mathsf{Open}^{h^*}_{ck}(c^*, m^*, d^*) = 1 \wedge$ | $\mathsf{Open}^{h^*}_{ck}(c^*, m^*, d^*) = 1 \wedge$ |
| $(m, h) \neq (m^*, h^*) \wedge$ | $(m, h) \neq (m^*, h^*) \wedge$ |
| $(m^* \neq \perp) \wedge (h^* \neq \perp) \wedge$ | $(m^* \neq \perp) \wedge (h^* \neq \perp) \wedge$ |
| $\mathsf{R}(m, m^*) = 1$ | $\mathsf{R}(m, m^*) = 1$ |

Fig. 1. Game for non-malleability with respect to opening and associated-data (NM-OAD).

For the relation $\mathsf{R}$, unlike prior works, we do not need to rule out the possibility of "message copying" (by insisting $\mathsf{R}(m, m) = 0$ with probability 1), as the adversary may well win the game by simply modifying the header.

We require that for any adversary which, on input the commitment $(h, c)$, finds a new commitment to a related message, there exists a simulator which can simulate the commitment and the decommitment just as well without the original commitment.

Let $\Pi$ be an NM-CAD, let $\mathsf{R}$ be a relation, let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ be any PPT adversary, and let $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3)$ be the simulator. We define the NM-OAD advantage of $\mathcal{A}$, $\mathbf{Adv}^{\text{NM-OAD}}_{\Pi,\mathcal{A}}(l)$, as $\Pr[\mathbf{Exp}^{\text{nm-oad-0}}_{\Pi,\mathcal{A}}(l) = 1] - \Pr[\mathbf{Exp}^{\text{nm-oad-1}}_{\Pi,\mathcal{S}}(l) = 1]$.

*Remark.* In defining NM-OAD, we chose to use a notion similar to non-malleability with respect to opening. But one can easily modify the experiment to define a stronger notion that resembles the notion of non-malleability with respect to commitment, by requiring the simulator to output $m^*$ and $d^*$ given $ck$ and $ht_m$ only. But as we will see, our NM-OAD definition suffices for our application.

A definitional choice we made is that in order to win the game, adversary should produce a pair $(m, h) \neq (m^*, h^*)$. In fact, according to our protocol (CP1), it is sufficient to require $(m \neq m^*, h \neq h^*)$ or $(m = m^*, h \neq h^*)$.[2] Our definition yields a stronger one.

Our definition echoes some other cryptographic primitives such as authenticated encryption with associated-data [58] and adaptive one-way function [55].

*Extensions and simplifications on commitment with associated-data.* A conventional commitment scheme can be easily obtained by removing the header in its algorithms. The definition can be also easily extended to the concurrent

[2]This is because according to our protocol, the recipient of the commitment will also verify the authenticity of the message.

setting [56] where the adversary is receiving commitments to multiple messages $m_1, \cdots, m_s$, while attempting to commit to related messages $m'_1, \cdots, m'_s$. This is a stronger notion than non-malleable commitment.

**Generic constructions and efficient instantiations.** It was shown that concurrent NMC can be built from any one-way function [51]. It is easy to show that concurrent NM-CAD can be built from any adaptive one-way function [55].

Building NMC is trivial in the ROM; one just needs to commit to a random coin in addition to the message itself.

Concurrent NM-CAD can be built likewise but with a small tweak. For simplicity, let $\mathcal{M} = \{0,1\}^l$ and $\mathcal{H} = \{0,1\}^l$. Fix a hash function $\mathsf{H}\colon \mathcal{K} \times \{0,1\}^{3l} \to \{0,1\}^l$. Cgen simply returns a uniformly chosen hash key $k \xleftarrow{\$} \mathcal{K}$. Given a message $m$ and a header $h$, Commit selects a uniformly random coin $r$, computes $c = \mathsf{H}_k(h, m, r)$, and returns $(c, d)$, where $d = r$. Given $h$, $c$, $m$, and $d$, Open checks if $c = \mathsf{H}_k(h, m, d)$ and returns a decision bit. The scheme can be easily shown to be NM-OAD in both the stand-alone and concurrent settings.

**A brief explanation.** NM-CAD is designed specifically for CP1. NMC also works for CP1, but it requires slightly higher bandwidth. Since we consider concurrent requests, we will correspondingly need concurrent non-malleable commitments (with associated-data). For CP2, a conventional commitment scheme is sufficient (explanation coming shortly).

### C. Secret Sharing Schemes

In a $(t, n)$ secret-sharing scheme (SS), the dealer distributes shares of the secret to $n$ servers such that: 1) any group of $t$ (or more) servers can reconstruct the secret, and 2) any group of fewer than $t$ servers cannot obtain any partial information on the secret. One well-known SS is Shamir's secret sharing scheme [63] which is based on polynomial interpolation.

**Asynchronous robust secret sharing.** We consider *asynchronous robust secret sharing*, where the dealer is correct, up to $t-1$ servers may be Byzantine, and the environments may be asynchronous. To formalize this, we extend *robust secret sharing* (RSS) by Bellare and Rogaway (BR) [9] to asynchronous settings.

A $(t, n)$ asynchronous robust secret-sharing (ARSS) protocol consists of a *distribute* stage and *reconstruct* stage. In the *distribute* stage, a dealer runs Share on $s$ and outputs a $n$-vector $S = S[1..n]$, and sends the server $i$ the share $S[i]$. In the *reconstruct* stage, each server broadcasts its share to the rest of the servers. Each server waits for *enough* shares in a set $S'$ and runs Rec on $S'$ to return a secret or a distinguished symbol $\perp$.

The above syntax is natural, simple, but rather restrictive: the distribute stage consists of a best-effort broadcast, and the reconstruction stage consists of best-effort broadcast among servers. We find, however, the simple syntax is sufficient to provide efficient constructions. Also, in our syntax, the number of shares needed for reconstruction, $u$, where $t \le u \le n$, does not have to be fixed. As we will see, typical choices of $u$ such as $t$, $t+f$, and $n-f$, may restrict the possibility of efficient constructions.

| **Experiment** $\mathbf{Exp}^{\text{priv}}_{\text{ARSS}}(\mathcal{A})$ | **Experiment** $\mathbf{Exp}^{\text{rec}}_{\text{ARSS}}(\mathcal{A})$ |
|---|---|
| $T \xleftarrow{\$} \mathcal{A}$; $(s_0, s_1, st) \xleftarrow{\$} \mathcal{A}(T)$ | $T \xleftarrow{\$} \mathcal{A}$; $(s, st) \xleftarrow{\$} \mathcal{A}(T)$ |
| $b \xleftarrow{\$} \{0,1\}$ | $S \xleftarrow{\$} \mathsf{Share}(s)$ |
| $S \xleftarrow{\$} \mathsf{Share}(s_b)$ | $S'_T \xleftarrow{\$} \mathcal{A}(S_T, st)$ |
| $b' \xleftarrow{\$} \mathcal{A}(S_T, st)$ | $S' \leftarrow S_{\overline{T}} \sqcup S'_T$; $R \leftarrow R \cup \{S'[i]\}$ |
| **if** $b' = b$ **then return** 1 | **if** $s \neq \mathsf{Rec}(R)$ **return** 1 |
| **return** 0 | **return** 0 |

Fig. 2. Privacy game (Left) and recoverability game (Right) with adversary $\mathcal{A}$ and an ARSS scheme.

We consider two security definitions for ARSS—privacy and recoverability in Fig. 2. We assume a static adversary who must decide at the beginning of its execution which players $T$ to corrupt.

In the privacy game, adversary $\mathcal{A}$ decides which servers to corrupt before its execution. $\mathcal{A}$ chooses two secrets $s_0$ and $s_1$ such that $|s_0| = |s_1|$. Then the game chooses a hidden bit $b$ and runs $\mathsf{Share}(s_b)$ to produce a $n$-vector $S$. The shares corresponding to the corrupted set $T$, $S_T$, are then given to $\mathcal{A}$ who will then output its guess $b'$ for the hidden bit. Formally, we define $\mathbf{Adv}^{\text{priv}}_{\text{ARSS}}(\mathcal{A}) = \Pr[\mathbf{Exp}^{\text{priv}}_{\text{ARSS}}(\mathcal{A}) = 1] - 1/2$.

In the recoverability game, adversary $\mathcal{A}$ statically corrupts a set of $T$ servers. $\mathcal{A}$ then chooses a secret $s$ and the game samples $S$ from $\mathsf{Share}(s)$. $\mathcal{A}$ replaces the shares $S_T$ with $S'_T$. The new $n$-vector of shares $S'$ is now $S_{\overline{T}} \sqcup S'_T$. The reconstructor collects shares in a set $R$ and runs $\mathsf{Rec}(R)$ to return a secret or a distinguished symbol. $\mathcal{A}$ wins the game if $s \neq \mathsf{Rec}(R)$. We define $\mathbf{Adv}^{\text{rec}}_{\text{ARSS}}(\mathcal{A}) = \Pr[\mathbf{Exp}^{\text{rec}}_{\text{ARSS}}(\mathcal{A}) = 1]$.

In BR's privacy game, adversary can corrupt up to $t-1$ servers, while in their recoverability game, adversary can corrupt up to $n-(t-1)$ servers. In both of our games, we require that adversary can only corrupt up to $t-1$ servers. On the one hand, this suffices for our purpose, as we will consider the BFT setting where adversary can corrupt at most $t-1$ servers. On the other, this relaxation makes it possible to design highly efficient constructions and information-theoretic constructions.

*Implying non-malleability.* One may consider defining non-malleability for ARSS just as in encryption and commitment schemes. Specifically, one may consider a malleability adversary that corrupts up to $t-1$ servers and attempts to produce shares that can be recovered to a new secret $s'$ that is "meaningfully related" to the original secret $s$. We claim that this is *not* needed, as our privacy definition simply implies non-malleability. Unlike encryption and commitment schemes, ARSS is an *unkeyed* primitive, where both Share and Rec algorithms do not take as input a key. Once some malleability adversary produces shares for some secret $s'$ related to some $s$, it essentially knows the entire $s'$. One can then construct another adversary that attacks the privacy game.

Note that in order for the implication to hold, the adversary can corrupt up to $t-1$ servers. This implication does not hold if the adversary corrupts more servers. In fact, the malleability property (in particular, the linear property) on

secret sharing can be used to build fault-tolerant multi-party computation [11].

*Remark.* We intentionally make the definitions not to depend on a security parameter, as the security may hold in an information-theoretic setting. However, it is easy to extend our definitions to both concrete and asymptotic settings. For the asymptotic setting, we should avoid using the length of the secret as the security parameter.

Our definition of ARSS is a strengthening of RSS and a simplification of asynchronous verifiable secret sharing (AVSS) [20]. ARSS strengthens RSS to handle asynchrony issues. In BR's definition, networks are synchronous and the reconstructor will mark each missing share using a distinguished symbol. But we consider asynchronous environments, where there is no known upper bound on processing and message transmission delays.

ARSS relaxes AVSS in the sense that in ARSS the dealer is correct but in AVSS the dealer may be malicious. In addition, our syntax is much simpler, but we believe this does not restrict the possibility of efficient constructions for our purpose. Last, our definitions on privacy and recoverability with general adversaries are more general (and formal) than those in [20].

**Efficient constructions.** We present two ARSS constructions—ARSS1 that has computational security and ARSS2 that is information-theoretically secure. Both of the two constructions are as efficient as a regular secret sharing scheme, and several orders of magnitude faster than the most efficient AVSS [20] for any reasonably large (practical) $n$. For our purpose, we assume that $f = t - 1$ servers may behave arbitrarily and $n \geq 3f + 1$.

*A computational construction—ARSS1.* ARSS1 is general: it uses any conventional secret sharing scheme and any conventional commitment scheme in a black-box manner.

Let SS be any $(t, n)$ secret sharing scheme with $(\mathsf{Share}', \mathsf{Rec}')$ algorithms and let $\mathsf{CS} = (\mathsf{Cgen}, \mathsf{Commit}, \mathsf{Open})$ be any (conventional) commitment scheme with a commitment key $ck$. We define a $(t, n)$ ARSS scheme ARSS1 with $(\mathsf{Share}, \mathsf{Rec})$ algorithms as defined below. In the distribute stage, given a secret $s$, $\mathsf{Share}$ runs $(c, d) \xleftarrow{\$} \mathsf{Commit}_{ck}(s)$ (where $c$ and $d$ are the commitment and the opening respectively), runs $\mathsf{Share}'(s, d)$ to get a $n$-vector $S' = S'[1..n]$, and returns $S = S[1..n]$ such that $S[i] = (c, S'[i])$ for $i \in [1..n]$.

In the reconstruct stage, the reconstructor keeps waiting for shares to come and maintains sets of shares tagged by the same $c$. It drops other sets once the size of some set $R$ reaches $t$. It stops accepting new shares once the size of the set $R$ reaches $2f + 1$. In the meanwhile, once the size of the set $R$ reaches $f + 1$, it attempts to recover the secret by first running $\mathsf{Rec}'$ to get some $(s, d)$ and then verifying if $\mathsf{Open}_{ck}(c, s, d) = 1$. In the worst case, the reconstructor needs to verify at most $\binom{2f+1}{f}$ possible combinations to recover $s$.

If the reconstructor is also a share holder (*i.e.,* server), the above algorithm can be simplified: each correct server does not need to send the reconstructor the commitment or maintain multiple sets.

Instead of requiring a NMC, a conventional commitment scheme is sufficient. We argued that as long as an ARSS scheme is secure with respect to the privacy game, it is also secure in the sense of non-malleability. Still, let us explain the underlying idea for this construction. The non-malleability adversary's goal is not to make the commitment malleable but to provide a related secret. In fact, if the non-malleability adversary wins the non-malleability game for ARSS1 then we can construct an efficient adversary that attacks the hiding property of the commitment scheme.

Note that $\mathsf{Share}$ is invoked on *both* the secret *and* the opening. Sharing the secret only does not suffice, because later on we will also need the opening to verify the correctness of the commitment scheme. To efficiently instantiate the scheme, one can use the hash based commitment scheme described earlier. We do not claim the originality of the scheme, as we are not sure if similar constructions appeared elsewhere. Rather, we show that ARSS can be rather easily obtained both generically and efficiently.

*An information-theoretical construction—ARSS2.* ARSS2 can be adapted from a construction by Harn and Lin (HL) [41]. Padilha and Pedone [53] first used it in a BFT storage system for a purpose that is different from ours. In their scheme, the clients run both the $\mathsf{Share}$ and $\mathsf{Rec}$ algorithms, but in our scheme the $\mathsf{Rec}$ algorithm is invoked among the servers and reconstructors are also share holders. Moreover, in their system, the data is shared among servers and the servers are unable to see or process the data. We just regard provably fitting ARSS2 in our generic framework (and provably building secure causal BFT from ARSS in general) as our contributions.

ARSS2 is designed specifically for Shamir's SS. ARSS2 with $(\mathsf{Share}, \mathsf{Rec})$ algorithms can be built from a $(t, n)$ Shamir's SS with $(\mathsf{Share}', \mathsf{Rec}')$ algorithms. $\mathsf{Share}$ is identical to $\mathsf{Share}'$. The reconstructor waits for $f + 2$ shares to see if they are consistent shares (*i.e.,* they are points on the same polynomial). If not, the reconstructor will know that at least one out of the $f + 2$ shares is faulty. In the worst case, the reconstructor has to wait for $2f + 2$ shares in total, and would try all $\binom{2f+2}{f+2}$ combinations.

In ARSS2, to check whether $g$ ($g \geq t$) shares are consistent, one only needs to check whether the interpolation of $m$ points yields a polynomial with degree $t - 1$. This consistency check requires only one Lagrange interpolation operation, and recovering the secret will take another interpolation operation.

In comparison, ARSS1 needs one Lagrange interpolation operation and one hash evaluation operation. However, for both the failure-free and failure scenarios, ARSS2 needs a larger number of shares to recover the secret.

## V. The Protocols

We begin by reviewing CP0 and then present three new secure causal BFT protocols—CP1, CP2, and CP3. We describe them in the client-server computing model.

### A. CP0

**CP0 definition [21].** In addition to the conventional safety and liveness notions, CKPS defined three notions that are directly related to causality preservation—message integrity, message consistency, and message secrecy. The first two ensure that correct replicas receive the same underlying plaintext, while the last one is related to confidentiality.

CKPS divided the protocol into *schedule* and *reveal* processes. The schedule process corresponds to the conventional atomic broadcast protocol, and the reveal process is an additional process run after each replica completes the atomic broadcast. CKPS requires that for correct replicas two consecutive schedule or reveal processes are not allowed. The idea is that only after a replica *schedules* a ciphertext, it can *reveal* and broadcast its decryption share.

Let's first recall the message secrecy definition. The adversary interacts with the correct replicas in an arbitrary way. It then chooses two messages $m_0$, $m_1$, and a tag $\mathsf{ID}$ and provides them to an encryption oracle. The oracle randomly chooses a bit $b \in \{0, 1\}$, computes an encryption $c$ of $m_b$ with tag $\mathsf{ID}$, and gives the ciphertext to the adversary. The adversary then continues to play with the correct replicas subject to the condition that no correct replica schedules $c$ with $\mathsf{ID}$. Finally, the adversary outputs a bit $b'$ as its guess.

Regarding message integrity, the adversary chooses some message $m$ and $\mathsf{ID}$ and gives them to the encryption oracle. The adversary wins the game if some correct replica schedules a ciphertext but the associated plaintext $m'$ is different from $m$. Message consistency requires that if two correct replicas schedule the same ciphertext $c$ with tag $\mathsf{ID}$, then the associated plaintexts are the same.

**CP0 construction.** CKPS uses a CCA secure $(f + 1, n)$ robust labeled threshold encryption scheme. A trusted dealer is responsible for initializing the system keys: it generates a system public key (so that everyone can encrypt messages using the public key) and distributes the private key shares to the corresponding replicas (so that any group of $f + 1$ replicas can collectively decrypt the ciphertext). Alternatively, an expensive and interactive key setup protocol can be used for the setup.

In the schedule process, a client generates a labeled threshold ciphertext, which is then sent to the replicas. Then replicas run the underlying atomic broadcast protocol to schedule this ciphertext. After scheduling the ciphertext, each replica will reveal its decryption share to the rest of the replicas such that correct replicas will be able to reconstruct the plaintext.

It is important to use "labeled" threshold encryption where the label should contain a unique identifier $\mathsf{ID}$ (including the client identity and the message identifier). Moreover, the communication between all parties (clients and replicas) should use authenticated channels. Each replica should verify that the label in the ciphertext indeed contains the identity of the sender.

### B. An Extended Definition Framework

CKPS's definition is coupled with threshold encryption. We extend their framework to support general primitives.

In our framework, a client may send a message $m$ via encryption, commitment scheme, or secret sharing. The delivery of a secure causal protocol consists of two steps: schedule and reveal. In the schedule process, a message of the form $(\mathsf{ID}, \mathsf{schedule}, D)$ will be committed. $D$ may be a ciphertext (as in CP0), a NM-CAD (as in CP1), a conventional commitment (as in CP2), or an empty message (as in CP3). In the reveal process, a message of the form $(\mathsf{ID}, \mathsf{reveal}, m)$ will be delivered. We require that there must not be two consecutive schedule and reveal processes.

As in CKPS, we define message secrecy, message integrity, and message consistency. We follow CKPS to define message consistency: if two correct replicas committed the same schedule message, then the plaintext content recovered by each replica for that message will be the same, with all but negligible probability. However, message secrecy and message integrity will be modeled depending on concrete constructions.

### C. CP1

**Defining message secrecy and integrity.** The definition for CP1 is related to a commitment scheme. We model message secrecy using message hiding and message non-malleability. Message hiding requires the adversary not to learn any information on the committed value before the committer decides to reveal it. We now describe message non-malleability. The adversary is given the commitment key $ck$ and interacts with the correct replicas in an arbitrary way. It then chooses a non-trivial message space $\mathcal{M}$ and a header space $\mathcal{H}$. An commitment oracle randomly chooses a message and a header and computes $(c, d) \xleftarrow{\$} \mathsf{Commit}^h_{ck}(m)$. The adversary is given $(h, c)$ and now has to generate a new commitment $(h', c')$ (before $(h, c)$ gets scheduled). After the adversary is given the message $m$ and the opening $d$, it needs to output its message $m'$ and the opening $d'$ for the new commitment such that $\mathsf{R}(m, m') = 1$ for some PPT function $\mathsf{R}$. (We do not need to worry about unopened commitments, as they will be removed according to our protocol.) The adversary wins the game if either $(m \neq m', h \neq h')$ or $(m = m', h \neq h')$. Note that the adversary will not win the game for the case of $h = h'$, as in this case a derived commitment will be trivially rejected by correct replicas, because in our protocol, replicas will need to verify if the header of a message matches the message sender.

Message integrity can be defined as follows. We consider an adversary that interacts with the correct replicas and is given a commitment $(h, c)$ for some message $m$. We require the following probability to be negligible: some correct replicas scheduled $(h, c)$ in the schedule process, but the associated committed value is not equal to $m$.

**Construction.** CP1 is built on a NM–CAD $\Pi = (\mathsf{Cgen}, \mathsf{Commit}, \mathsf{Open})$ and a fair BFT protocol that can tolerate up to $f$ Byzantine failures. CP1 has at least two benefits compared to CP0: first, CP1 does not rely on a trusted setup or

an expensive, interactive setup; second, CP1 can be efficiently realized using only symmetric cryptography.

The basic idea of CP1 is as follows: in the schedule process, a commitment to some chosen value $m$ with an identifier $\mathsf{ID}$ is delivered via the underlying BFT protocol; in the reveal process, $m$ and the associated opening $d$ also go through the same BFT protocol using the *same* identifier $\mathsf{ID}$.

Let's describe CP1 in detail. The system fixes a commitment key $ck$ using $\mathsf{Cgen}$. Given a client message $m$, the client picks a unique identifier (denoted as $\mathsf{ID}$) as the commitment header $h$. It then computes $(c, d) \xleftarrow{\$} \mathsf{Commit}_{ck}^{\mathsf{ID}}(m)$. The client then sends $(\mathsf{ID}, \mathsf{schedule}, c)$ to replicas. Replicas need to verify the authenticity of the message and also verify if the header matches the identity of the client. Then replicas run the BFT protocol to schedule the commitment and then notify the client that they have delivered the message. As in a conventional BFT protocol, once receiving $f + 1$ matching reply messages, the client will initiate the reveal process by sending $(\mathsf{ID}, \mathsf{reveal}, (m, d))$ to the replicas. The replicas verify the correctness of the message and the opening and then run the BFT protocol again to deliver the message. Then replicas can process the message and send the reply messages to the client.

We implement the following optimization: While the replicas are waiting for an opening to be delivered, they still batch and schedule client requests (but do not run the reveal process).

This completes the description of failure-free scenarios. But what could go wrong here? First, clients may fail to send messages and openings "in time." Second, replicas could delay or drop messages and openings too. Both scenarios can block the service.

To defend against these circumstances, we first implement an amplification step for a common type of client crash failures: in the reveal process, once a correct replica verifies the message and the opening, it simply *forwards* them to the rest of the replicas. The message and the opening serve as a transferable witness, and they do *not* need to be authenticated. This is rather different from the protocols using similar techniques.

To (completely) resolve these, we require the replicas to *periodically* get rid of tentative requests, *i.e.,* committed but unopened client requests. In sequencer-based protocols, this procedure can be initiated by the primary. This procedure only makes sense for a fair BFT protocol, because otherwise replicas might unfairly delay some correct clients' reveal messages such that the corresponding requests are cleaned by the cleanup procedure.

An underlying assumption is that the delay between a correct client and at least $f+1$ correct replicas (which watch if primary enforces fairness and trigger view change otherwise) cannot be significantly larger than the delay between another correct client and the $f + 1$ correct replicas. We define the channel delay to be the maximum number of steps (measured by tentative requests) that a specific request can be later than another correct request issued at the same time. We also define the fairness delay to be the maximum number of steps (also measured by tentative requests) that a specific request can be delayed. Clearly, to ensure correctness, we just require that the cleanup cycle be larger than the sum of the channel delay and fairness delay. If the primary does not initiate the cleanup protocol according to this rule, a view change will be triggered.

It is easy to see that CP1 allows batching and if the underlying BFT protocol tolerates faulty clients so does CP1.

THEOREM 1. *Assuming a secure NM-CAD (satisfying hiding, binding, and NM-OAD) and a fair BFT protocol, CP1 is a secure causal BFT protocol.*

**Remark.** There are a few seemingly feasible approaches that actually do not work. One such approach is to first deliver the commitment using the BFT protocol and then use a best-effort broadcast protocol to broadcast the opening. This does not work because, for instance, the client may simply crash. A second approach is to first deliver the commitment and then use a reliable broadcast protocol to broadcast the opening. Liveness might be impeded because BFT and reliable broadcast have different means to ensure liveness and tentative requests from correct clients may be falsely cleaned.

One may also use NMC instead of NM-CAD. In order for this to work, each client needs to append $\mathsf{ID}$ to the committed message, and in the reveal process replicas will verify if $\mathsf{ID}$ matches the identity of the sender. This needs extra bookkeeping for the protocol and extra communication for amplification. More importantly, NM-CAD is a more natural fit for the scenario where associated data need not be privacy-protected. Depending on constructions, the method using NMC may also incur additional computational overhead.

### D. CP2 and CP3

We consider the scenario where clients may be only subject to crash failures. This scenario models a large set of realistic circumstances: 1) Clients are honest-but-curious, a commonly used assumption in distributed systems and multi-party computation. 2) Clients may be some upper-level applications already made intrusion-tolerant. In practical distributed systems and cloud computing platforms, both clients and servers may be replicated. 3) Clients may not be interested in attacking the safety property of the BFT service, but just aim at compromising causality. This is because attacking causality can allow clients to gain immediate benefits, while attacking the consistency of the system may not.

**Definition of security for CP2 and CP3.** We define for CP2 and CP3 message secrecy, message integrity, and message consistency in a way that resembles ARSS. As in ARSS, there is no need to consider message non-malleability because in the SS setting, the notion is implied by message secrecy.

**A generic construction from ARSS.** A secure causal BFT protocol can be built from any BFT protocol and any $(f+1, n)$ ARSS scheme with $(\mathsf{Share}, \mathsf{Rec})$ algorithms as follows.

A client runs the ARSS $\mathsf{Share}$ algorithm on its request $m$ to generate a $n$-vector $S = S[1..n]$. The client then sends each replica $i$ a request $(\mathsf{ID}, \mathsf{schedule}, S[i])$. Each replica needs to maintain a mapping between $S[i]$ and $\mathsf{ID}$. In the

schedule process, the replicas run the underlying BFT protocol to order the message identifier ID. When each replica $i$ delivers the identifier, it starts the reveal process by broadcasting $(\mathsf{ID}, \mathsf{reveal}, S[i])$ to the rest of the replicas. When a replica collects enough shares, it runs Rec to decide a secret or a distinguished symbol.

The communication carrying secret shares (*i.e.,* the communication in the secret distribute and reconstruct stages) needs authenticated and private channels. For rest of the communication, we just need authenticated channels.

THEOREM 2. *The above generic construction built from a BFT protocol and an ARSS scheme is a secure causal BFT protocol.*

**Instantiating CP2 and CP3.** CP2 and CP3 can be obtained by instantiating the above generic construction using ARSS1 and ARSS2 respectively.

For CP2, we use the specific ARSS1 algorithm for the case where reconstructors are also share holders. More specifically, in the schedule process, the replicas will agree upon not only ID but also the commitment $c$. In the reveal process, it can reject any faulty shares with a commitment different from $c$. This leads to a more efficient construction.

For both CP2 and CP3, if clients are faulty (either Byzantine or not), input causality is never violated, but both safety and liveness can be compromised, just as those BFT protocols that do not tolerate Byzantine faulty clients.

### E. Summary

We summarize the three frameworks and the four instantiations for secure causal BFT protocols in TABLE I. The framework using threshold encryption can be realized using only specific number-theoretic assumptions. It requires either a trusted dealer or an expensive interactive protocol to distribute the system keys. This is also the only framework using expensive threshold cryptography. All the rest of the frameworks have efficient instantiations. The framework using fair BFT and NMC can be based on rather generic cryptographic primitives such as one-way functions. The framework using ARSS deals with the case of benign clients and can be divided into two categories—using ARSS1 and using ARSS2. The first one can be based on any commitment scheme (thus one-way function) and any secret sharing scheme. The second one is based Shamir's SS and is information-theoretically secure.

## VI. IMPLEMENTATION AND EVALUATION

### A. Implementation

Our test setting comprises a cluster of 15 machines (2.13GHz Xeon processor, 4GB RAM), 5 of which serve as client nodes. Experiments are carried out on DeterLab [17].

We utilize PBFT [26] as our baseline and as the underlying BFT protocol to instantiate CP0, CP1, CP2, and CP3. Our PBFT implementation is based on that of [4] and we implement our fairness mechanism according to [27]. While our protocols also apply to asynchronous consensus-based BFT protocols (*e.g.,* the one in CKPS [21] implemented in SINTRA [23]), the performance difference is less visible compared to efficient BFT protocols such as PBFT. The reason is that in addition to threshold encryption operations, there are other expensive operations for those asynchronous protocols.

We use HMAC [6] as the MAC algorithm to build authenticated channels. We use a composed authenticated encryption with associated-data scheme [58] to build authenticated and private channels. Specifically, we apply CTR mode encryption (using AES-256) and then compute its HMAC. We use SHA-256 to instantiate the NM-CAD (for CP1) and the conventional commitment scheme (for CP2).

For CP0, we extend Baek and Zheng's threshold cryptosystem [5] to securely support labels. We then modify an implementation of Miller *et al.* [52] to enable this change. The implementation uses hybrid encryption to encrypt long messages. When evaluating this threshold cryptosystem, we choose a very conservative (insecure) security parameter (less than 80 bits of security) [15]. Still, CP0 is several orders of magnitude slower than our three protocols, if the network latency is small.

### B. Evaluation

**Overview.** We use the Castro and Liskov micro-benchmarks [26] to assess throughput, latency, scalability, and performance during failures of all the five protocols—PBFT, CP0, CP1, CP2, and CP3. In the $x/y$ micro-benchmarks, clients send $x$ kB requests and receive $y$ kB replies. Clients invoke requests in a *closed-loop*, where a client does not start a new request before receiving a reply for a previous one. All the protocols implement batching of concurrent requests to reduce cryptographic and communication overheads.

We benchmark the protocols in two settings: a LAN setting with 100 MB bandwidth and 0.1 ms latency, and a WAN setting with 1 MB bandwidth and 120 ms latency. We show that for both settings, and for both the gracious and uncivil executions, CP1, CP2, and CP3 add reasonably small overhead to the underlying PBFT protocol, and they all significantly outperform CP0.

**Latency.** We first report the latency evaluation in the LAN setting. We examine and compare the average latency under no contention in the 4/0 benchmark, as depicted in TABLE II. The results for $0/0$, $0/4$, and $4/4$ benchmarks are similar, as the hybrid encryption for CP0 and symmetric cryptography for the other four protocols scale well as the length of the message increases.

We find that CP1 is 79%∼84% slower than PBFT. This is expected because CP1 essentially runs two rounds of PBFT to deliver a message. Both CP2 and CP3 have lower latency than CP1 and higher latency than PBFT. This is also expected, because CP2 and CP3 only add one more broadcast among replicas compared to PBFT. We also find that the latency difference between CP2 (and CP3) and PBFT becomes larger as the maximum number of faulty replicas $f$ increases.

In contrast, CP0 is several orders of magnitude slower than the rest of the four protocols. This is because the penalty due to the expensive threshold cryptography is particularly visible in the LAN setting. We also report in Fig. 3 the latency for

TABLE I

A COMPARISON. THE COLUMN LABELED "TY" SPECIFIES IF THE SCHEME USES PUBLIC-KEY CRYPTOGRAPHY (PK), SYMMETRIC CRYPTOGRAPHY (SK), OR IS INFORMATION-THEORETICALLY SECURE (ITS). THE COLUMNS LABELED "BYZANTINE CLIENTS," "SETUP," AND"BATCH" SPECIFY IF THE SCHEME TOLERATES BYZANTINE FAULTY CLIENTS, RELIES ON TRUSTED OR EXPENSIVE, INTERACTIVE SETUP, AND ALLOWS BATCHING, RESPECTIVELY.

| frameworks | instantiations | ty | Byzantine clients | setup | batch | generality |
|---|---|---|---|---|---|---|
| BFT+ThreshEnc | CP0 | pk | √ | √ | √ | no known constructions from generic primitives |
| Fair BFT+NMC | CP1 | sk | √ | – | √ | any (adaptive) one-way function |
| BFT+ARSS1 | CP2 | sk | – | – | √ | any commitment scheme and any SS |
| BFT+ARSS2 | CP3 | its | – | – | √ | only for Shamir's SS |

TABLE II

LATENCY IN MS (LAN).

| Protocol | $f = 1$ | $f = 2$ | $f = 3$ |
|---|---|---|---|
| PBFT | 0.23 | 0.24 | 0.25 |
| CP0 | 769.13 | 881.00 | 1090.59 |
| CP1 | 0.42 | 0.43 | 0.46 |
| CP2 | 0.25 | 0.28 | 0.39 |
| CP3 | 0.24 | 0.29 | 0.43 |

TABLE III

LATENCY IN MS (WAN).

| Protocol | $f = 1$ | $f = 2$ | $f = 3$ |
|---|---|---|---|
| PBFT | 311.35 | 382.94 | 433.41 |
| CP0 | 1300.03 | 1429.30 | 1506.73 |
| CP1 | 471.39 | 512.18 | 591.83 |
| CP2 | 372.44 | 400.03 | 525.29 |
| CP3 | 479.10 | 502.14 | 585.35 |

each operation in the threshold encryption implementation as the number of the replicas varies, where "verify ciphertext" stands for the public verification operation of the ciphertext.

In the WAN setting, the latency difference (see TABLE III) is comparatively smaller. Yet, CP0 is still roughly three times as slow as the rest.

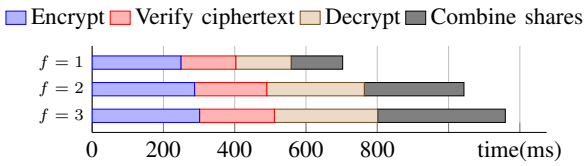


Fig. 3. Latency for each threshold encryption operation for $f = 1, 2, 3$.

**Throughput.** We discuss the throughput of all the five protocols with different workloads *under contention*, where multiple clients issue requests concurrently. In Fig. 4, we report the throughput for the case of $f = 1$ as the number of clients increases in the LAN setting. We find that all the rest of the protocols significantly outperform CP0. As the number of clients increases, CP0's throughput does not improve as much. Its peak performance is nearly 15 times lower than that of PBFT.

With fewer than 40 clients, CP2, CP3, and PBFT have similar throughput. As the number of clients further increases, PBFT achieves higher throughput. While CP2 and CP3 have comparable performance, CP2 has slightly higher throughput. The reason is that CP3 will need more shares than CP2

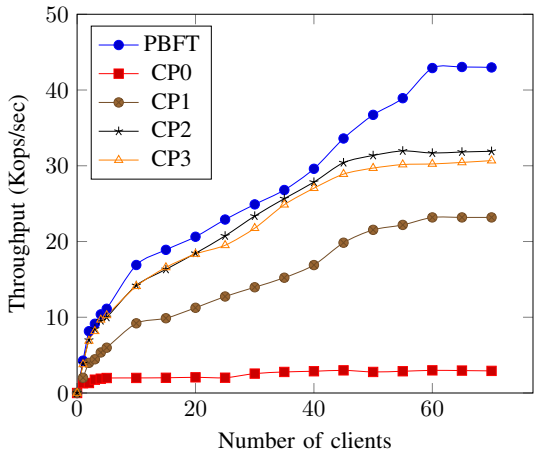


Fig. 4. Throughput for $f = 1$ in the LAN setting.

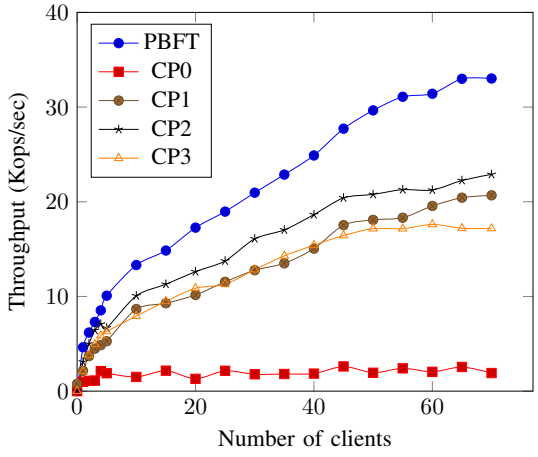


Fig. 5. Throughput for $f = 1$ in the WAN setting.

to recover the message. The peak throughput observed for CP2 is around 26% lower than that of PBFT. CP2 and CP3 have consistently better performance than CP1, and their peak throughput are 38% and 32% higher than that of CP1, respectively.

We also test the throughput of the five protocols as the maximum number of faulty replicas $f$ varies, as summarized in Fig. 6. We observe that the throughput difference between CP2 and CP3 becomes more visible when $f$ increases. This is because CP3 needs increasingly more secret shares to recover the secret, as $f$ increases.

Fig. 5 reports the throughput in the WAN setting. We find that the throughput of CP1, CP2, and CP3 remains signifi-

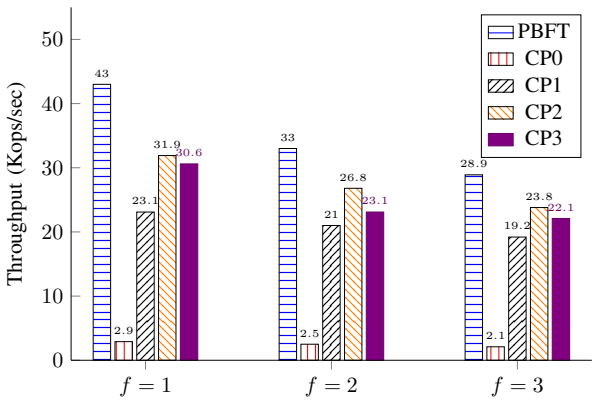


Fig. 6. Peak throughput for $f = 1, 2, 3$.

cantly higher than that of CP0. An interesting observation is that CP1 outperforms CP3 as the number of clients increases.

**Failure scenarios (in the LAN setting).** We first evaluate the case for CP1 where clients only send the witnesses (*i.e.,* the message and the opening) to a fraction of the replicas but not all, and the replicas are correct. In this case, correct replicas will run the amplification step to forward the witness. The forwarded message does not need to be authenticated, because the witness is transferrable. We observe no obvious performance difference between this failure scenario and the failure-free scenario in terms of both latency and throughput, which confirms our expectations.

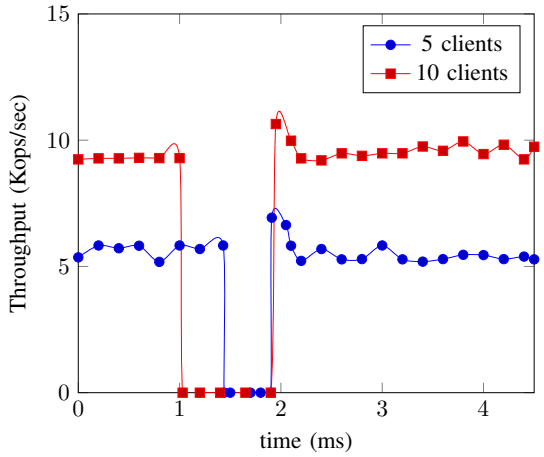


Fig. 7. Throughput for CP1 when clients are faulty.

We then evaluate another scenario for CP1 where clients are faulty and fail to send the witnesses to any correct replica. We set the cleanup cycle to be a conservative value—10 times average latency. (Normally, we need not set up a fixed cleanup cycle, and replicas just count how many requests have been scheduled after the first tentative request appears. Yet, in our setting, clients issue requests continuously and it makes sense to assume a fixed cleanup cycle.) In Fig. 7, we demonstrate the throughput when there are 5 and 10 clients respectively. In the experiment, they experience a failure at 1 ms and 1.5 ms, respectively. In both cases, during the cleanup cycle, the throughput becomes 0. The cleanup cycle with 5 clients is smaller than that with 10 clients, because the former has lower average latency. During the cleanup cycle, the replicas continue to batch and schedule clients requests. This explains the large throughput improvements after the cleanup procedure. The performance of the systems resume after the cleanup procedure. We comment that the performance fluctuation shares similarities with several state-of-the-art BFT protocols under failures such as Aliph [4] and BChain [33]. However, CP1 additionally preserves causal order.

TABLE IV
LATENCY FOR CP0, CP2, AND CP3 WITH FAULTY REPLICAS (IN MS).

| Protocol | $f = 1$ | $f = 2$ | $f = 3$ |
|---|---|---|---|
| CP0 | 1280.30 | 1513.40 | 1671.60 |
| CP2 | 0.26 | 0.39 | 0.50 |
| CP3 | 0.28 | 0.43 | 0.71 |

Last, we test the performance for CP0, CP2 and CP3 under Byzantine replica failures in TABLE IV. In this experiment, we *randomly* corrupt replicas and ask them to contribute faulty decryption shares. We find that both CP2 and CP3 have reasonable performance degradation. However, the difference between CP2 and CP3 becomes even more visible compared to that in the failure-free scenario. The reason is that CP3 requires even more communication and computation to recover the message in the *average* case. In comparison, CP0 becomes much slower, as each replica has to run a lot more *expensive* decryption share verification operations to combine the message.

## VII. CONCLUSION

Over the past three decades, all known secure causal atomic broadcast protocols have been based on expensive threshold cryptosystems which can be built from only a handful of number-theoretic assumptions. We revisited the problem by providing both generic frameworks and efficient instantiations. In particular, we showed that secure causal atomic broadcast protocols can be constructed from a variety of cryptographic primitives; we also showed that our protocols significantly outperform those based on threshold cryptosystems. At the core of our constructions are two new cryptographic primitives which may be of independent interests.

## ACKNOWLEDGMENT

The authors are indebted to the DSN reviewers for their insightful comments. Sisi was sponsored in part by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the Department of Energy. Mike was supported in part by NSF grant CNS 1330599. Haibin was supported in part by NSF grant CNS 1330599 and CNS 1413996.

## REFERENCES

[1] M. Abd-El-Malek, G. Ganger, G. Goodson, M. K. Reiter, and J. Wylie. Fault-scalable Byzantine fault-tolerant services. *SOSP 2005.*
[2] A. Adya *et al.* FARSITE: Federated available and reliable storage for incompletely trusted environments. *OSDI'02.*

[3] Amazon Web Services (AWS). https://aws.amazon.com/

[4] P-L. Aublin, R. Guerraoui, N. Knezevic, V. Quema, and M. Vukolic. The next 700 BFT protocols. *TOCS*, vol. 32, issue 4, January 2015.

[5] J. Baek and Y. Zheng. Simple and efficient threshold cryptosystem from the gap diffie-hellman group. *GLOBECOM '03*, pp. 1491–1495, 2003.

[6] M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. *CRYPTO 1996.*

[7] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among notions of security for public-key encryption schemes. *CRYPTO '98.*

[8] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. *CCS 93*, 1993.

[9] M. Bellare and P. Rogaway. Robust computational secret sharing and a unified account of classical secret-sharing goals. *CCS 07*, 2007.

[10] M. Bellare and A. Sahai. Non-malleable encryption: Equivalence between two notions, and indistinguishability-based characterization. *CRYPTO '99.*

[11] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. *STOC 1988.*

[12] A. Bessani, E. Alchieri, M. Correia, and J. Fraga. DepSpace: A Byzantine fault-tolerant coordination service. *EuroSys '08.*

[13] A. Bessani, J. Sousa, and E. Alchieri. State machine replication for the masses with BFT-SMART. *DSN '14.*

[14] K. Birman and T. Joseph. Reliable communication in the presence of failures. *ACM Transactions on Computer Systems 1(5)*, 47–76, 1987.

[15] BlueKrypt: Cryptographic key length recommendation. https://www.keylength.com/

[16] D. Boneh, X. Boyen, and S. Halevi. Chosen ciphertext secure public key threshold encryption without random oracles. *CT-RSA*, 2006.

[17] T. Benzel.The science of cyber security experimentation: the deter project. *ACSAC*, pp. 137–148, 2011.

[18] M. Burrows. The Chubby lock service for loosely-coupled distributed systems. *OSDI*, 2006.

[19] C. Cachin. State machine replication with Byzantine faults. *Replication 2010*, pp. 169–184, 2010.

[20] C. Cachin, K. Kursawe, A. Lysyanskaya, and R. Strobl. Asynchronous verifiable secret sharing and proactive cryptosystems. *ACM CCS*, 2002.

[21] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup. Secure and efficient asynchronous broadcast protocols (extended abstract). *CRYPTO 2001.*

[22] C. Cachin, K. Kursawe, and V. Shoup. Random oracles in Constantinople: Practical asynchronous Byzantine agreement using cryptography. *Journal of Cryptology* 18(3), 219–246.

[23] C. Cachin and J. Poritz. Secure Intrusion-tolerant Replication on the Internet. *DSN 2002*, pp. 167–176, 2002.

[24] C. Cachin and A. Samar. Secure distributed DNS. *DSN 2004.*

[25] C. Cachin and S. Tessaro. Asynchronous verifiable information dispersal. *SRDS 2005.*

[26] M. Castro and B. Liskov. Practical Byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst*, 20(4): 398–461, 2002.

[27] A. Clement, E. Wong, L. Alvisi, M. Dahlin, and M. Marchetti. Making Byzantine fault tolerant systems tolerate Byzantine faults. *NSDI* 2009.

[28] J. Corbett *et al*. Spanner: Google's globally-distributed database. *OSDI*, 2012.

[29] J. Cowling *et al.* HQ replication: A hybrid quorum protocol for Byzantine fault tolerance. *OSDI* 2006.

[30] X. Défago, A. Schiper, and P. Urbán. Total order broadcast and multicast algorithms: taxonomy and survey. *ACM Comp. Surv.*, 36(4), 2004.

[31] G. Di Crescenzo, Y. Ishai, and R. Ostrovsky. Non-interactive and non-malleable commitment. *STOC '98*.

[32] D. Dolev, C. Dwork, and M. Naor. Nonmalleable cryptography. *SIAM J. Comp.* 30 (2) 391–437, 2000.

[33] S. Duan *et al*. BChain: Byzantine replication with high throughput and embedded reconfiguration. *OPODIS 2014.*

[34] S. Duan, Karl Levitt, H. Meling, S. Peisert, and H. Zhang. ByzID: Byzantine fault tolerance from intrusion detection. *SRDS*, 2014.

[35] S. Duan and H. Zhang. Practical state machine replication with confidentiality. *SRDS*, 2016.

[36] C. Fidge. Timestamps in message-passing systems that preserve the partial ordering. *11th Australian Computer Science Conference*, 1988.

[37] M. Fischlin and R. Fischlin. Efficient non-malleable commitment schemes. *Journal of Cryptology*, vol. 22, no. 4, 2009.

[38] J. Garay, R. Gennaro, C. Jutla, and T. Rabin. Secure distributed storage and retrieval. *TCS*, 243 (1-2): 363–389, 2000.

[39] C. Gentry. Fully homomorphic encryption using ideal lattices. *STOC '99.*

[40] O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious RAMs. *Journal of the ACM*, 1996.

[41] L. Harn and C. Lin. Detection and identification of cheaters in $(t, n)$ secret sharing scheme. *DCC*, vol. 52, no. 1, pp. 15–24, Jan. 09.

[42] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed. ZooKeeper: Wait-free coordination for Internet-scale systems. *USENIX ATC 2010.*

[43] A. Iyengar, R. Cahn, C. Jutla, and J. Garay. Design and implementation of a secure distributed data repository. *14th IFIP ISC*, 1998.

[44] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong. Zyzzyva: Speculative Byzantine fault tolerance. *SOSP 2007.*

[45] J. Kubiatowicz *et al.* OceanStore: An architecture for global-scale persistent storage. *9th ASPLOS*, 2000.

[46] R. Ladin, B. Liskov, and L. Shrira. Lazy replication: Exploiting the semantics of distributed services. *PODC*, pp. 43–57, 1990.

[47] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Comm. ACM 21*, 7 (July), 558–565, 1978.

[48] L. Lamport. Using time instead of timeout for fault-tolerant distributed systems. *Trans. Prog. Lang. and Systems* 6(2):254–280, 1984.

[49] L. Lamport. On interprocess communication. part I: Basic formalism. *Distrib. Comput. 1, 2*, 77–85, 1986.

[50] L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. *ACM Trans. on Programming Languages and Systems* 4(3): 382–401.

[51] H. Lin and R. Pass. Concurrent non-malleable commitments from any one-way function. *TCC 2008*, 2008.

[52] A. Mille, Y. Xia, K. Croman, E. Shi, and D. Song. The honey badger of BFT protocols. *ACM CCS 16*, 2016.

[53] R. Padilha and F. Pedone. Belisarius: BFT Storage with confidentiality. *NCA 2011.*

[54] R. Padilha and F. Pedone. Confidentiality in the Cloud. *IEEE Security & Privacy*, 2015

[55] O. Pandey, R. Pass, and V. Vaikuntanathan. Adaptive one-way functions and applications. *CRYPTO 2008*, pp. 57–74, 2007.

[56] R. Pass and A. Rosen. Concurrent nonmalleable commitments. *SIAM J. Comput.* 37(6): 1891-1925 (2008).

[57] M. K. Reiter and K. Birman. How to securely replicate services. *ACM TOPLAS*, vol. 16 issue 3, pp. 986–1009, ACM, 1994.

[58] P. Rogaway. Authenticated-encryption with associated-data. *CCS'02.*

[59] P. Rogaway, M. Bellare, and J. Black. OCB: A block-cipher mode of operation for efficient authenticated encryption. *ACM TISSEC*, 2003.

[60] A. Rowstron and P. Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. *SOSP 2001.*

[61] F. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Comput. Surveys* 22(4): 299–319, 1990.

[62] R. Schwarz and F. Mattern. Detecting causal relationships in distributed computations: In search of the holy grail. *Distributed Computing 7(3)*, 149–174, 1994.

[63] A. Shamir. How to share a secret. *Comm. ACM*, 22(11), 612–613, 1979.

[64] V. Shoup and R. Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. *EUROCRYPT '98.*

[65] J. Yin, J. Martin, A. Venkataramani, L. Alvisi, and M. Dahlin. Separating agreement from execution for Byzantine fault tolerant services. *SOSP 2003.*