Practical State Machine Replication with Confidentiality

Sisi Duan
Oak Ridge National Laboratory
duans@ornl.gov

Haibin Zhang
University of North Carolina at Chapel Hill
haibin@cs.unc.edu

***Abstract*—We address the problem of how to store and process data privately in cloud environments that employ state machine replication. We show that the only known solution to the problem (Yin *et al.*, SOSP '03) is potentially susceptible to attacks. We then present a new protocol that is secure in the stronger model we formalize. Our protocol uses only efficient symmetric cryptography, while Yin *et al.*'s uses costly threshold signatures.**

We implemented and evaluated our protocol. We show that our protocol is two to three orders of magnitude faster than Yin *et al.*'s, which is less secure than ours.

I. Introduction

State machine replication [34, 48] is fundamental and proven software technique to enable fault-tolerant and highly available services. It has been deployed in an increasing number of online services and cloud computing platforms, including Google's Chubby [13] and Spanner [23], Microsoft Azure [19], and Amazon Web Services [3].

In the hostile and untrusted cloud environments, where some nodes in the cloud may be compromised or exhibit arbitrary behavior, Byzantine fault-tolerant (BFT) state machine replication (*e.g.*, [20]), one dealing with arbitrary failures and malicious attacks, is particularly relevant.

However, it is notoriously difficult to achieve another essential goal for secure cloud computing—confidentiality (*i.e.,* data privacy) in replicated state machines. Here confidentiality means that a service only reveals the information users authorized to see. More broadly, in the client-server model, confidentiality includes data protection for client requests, server replies, and data stored and processed at the servers. Despite significant progress in secure outsourcing and computation, it has largely focused on the case of a single server (*e.g.,* fully homomorphic encryption [30]), replicated servers with limited operations (*e.g.*, storage systems), or generic and inefficient multi-party computation (see [25]). None of these efficiently achieve reliability and confidentiality for arbitrary operations.

The reason why maintaining both availability and confidentiality simultaneously is challenging is straightforward: while increasing the number of replicas reduces the possibility that many of them can fail at the same time, it also increases the chance that an adversary might gain control of the weakest replica and read its state.

For this reason, confidentiality issue has become a fundamental obstacle to cloud storage and cloud computing where replication plays a central role [41]. Also, as pointed out by [53], the performance overhead and the difficulty in protecting confidentiality are two major barriers that inhibit the genuine utility of BFT protocols. After many years' efforts, a variety of BFT protocols and implementations [1, 5, 10, 20, 24, 27, 32] have improved their efficiency to the point that they can be practical and useful in many situations (see, *e.g.,* [10]). However, the existing approaches to confidentiality are far from satisfactory.

The only known solution to the problem that can support arbitrary (non-deterministic) operations, confidential BFT state machine replication, is due to Yin, Martin, Venkataramani, Alvisi, and Dahlin (YMVAD) [53], which insightfully takes advantage of separating BFT agreement from operation execution. In their approach, an agreement cluster (AC) is responsible for running a conventional BFT protocol to assign unique sequence numbers to client requests. The execution cluster (EC) then executes requests in sequence number order. Between AC and EC is a distributed, reliable *privacy firewall* (PF), consisting of a cluster of replicated filter nodes. It is used to *filter* any malicious or inconsistent information sent from execution replicas and therefore helps achieve confidentiality.

Their construction is not yet practical because it uses expensive threshold signatures. We also find that in a broader context their protocol neither *strictly* achieves safety in the sense of reliable distributed systems nor does it consider how to achieve request and reply privacy in the sense of reduction-based provably secure cryptography.

Our contributions. In our paper we make the following contributions:

- We refine the notion of YMVAD confidential BFT (CBFT) and formalize it as probabilistic interactive Turing machines.
- We provide another look at the YMVAD protocol (hereinafter C-BFT$_0$), showing that any of its trivial instantiations may be susceptible to attacks.
- We provide a new and practical CBFT protocol, C-BFT, which uses only efficient symmetric cryptography, as opposed to C-BFT$_0$ that uses costly threshold signatures. We prove the correctness of C-BFT in our extended security model.
- We show that our CBFT protocol (with refined and enhanced security notions) can be used to build a much more efficient causality-preserving replicated scheme, a

primitive which was proposed by Reiter and Berman [42] over two decades ago and has only found a public-key cryptographic realization so far [16, 42].
- We implemented and evaluated C-BFT. We show that C-BFT is two to three orders of magnitude faster than C-BFT$_0$.

Fitting in the cloud infrastructure. While C-BFT$_0$ is an important first step towards a practical solution, it has not received much attention since its invention. This is due to its large overhead and the large number of (relatively expensive) filter nodes in PF. Our CBFT protocol significantly improves C-BFT$_0$ by two to three orders of magnitude; in particular, our PF is cryptography-free (assuming authenticated channels) and can be implemented using computationally constrained tools and hardwares. The improvement allows us to revisit the YMVAD architecture. Indeed, our new architecture fits well with modern cloud computing: AC and EC match control layer and computing nodes in cloud computing respectively, while PF can be realized using some existing tools already deployed in popular cloud platforms, such as internal authorization services, node auditors, load balancers, and intrusion detection systems, or using other tools, say, trusted components [36, 47].

II. Related Work

Efficient BFT protocols. Byzantine fault tolerance, or more broadly, Byzantine agreement [35], handling failures beyond crashes, has been a topic which attracted people from various areas (e.g., theory, cryptography, security, systems, distributed systems). Beginning with PBFT [20], a large number of practical, asynchronous BFT protocols were proposed [1, 5, 20, 22, 24, 27, 32]. Our protocol C-BFT is general and can be built on any of these protocols. Additionally, if the underlying BFT protocol is secure against Byzantine clients (*e.g.,* [22]), so is C-BFT.

Confidentiality and Byzantine fault tolerance. Confidentiality is a central goal in secure distributed programs in practice. However, it is non-trivial to extend existing distributed protocols to support this attribute. In particular, trivial methods using TLS or SSH to protect confidentiality and integrity of transmitted messages do not work, as they defend against "outside" adversaries only.

There are a number of works aiming to achieve confidentiality in distributed file systems or storage systems [2, 9, 18, 29, 31, 33, 40, 46]. These systems only support *store* and *retrieve* operations (except that [40] additionally supports *add* operations). However, YMVAD and ours strive to support arbitrary state machine operations. Furthermore, ours is more general than YMVAD as ours can handle both deterministic and randomized operations, while YMVAD only supports deterministic ones.

Separating agreement/control plane from execution/storage plane. YMVAD argued that the separation of agreement that orders requests and execution that processes requests for state machine replication enabled two significant advantages over previous approaches—reducing execution replication costs and enabling confidentiality. For replicated storage, a similar idea of separating functions that control metadata from those that store the payload data, introduced in [2], was recently investigated to build systems with stronger properties in both the replication-based setting [15] and erasure-coded setting [4]. Our work bolsters this line of research, demonstrating that with a more rigorous and formal treatment, the separation of agreement and execution for state machine replication can lead to a CBFT with stronger security guarantee and a causality-preserving replicated protocol [16, 42] without relying on public-key cryptography.

Threshold cryptography and majority voting. YMVAD extensively uses expensive threshold signatures. More efficient threshold symmetric encryption and MACs [12, 37, 38] cannot be used to replace threshold signatures in YMVAD. The reasons are twofold. First, these primitives are not Byzantine adversary resilient—one single distributed party can easily attack the liveness and safety of the system. Second, each distributed party has to communicate with one another, which imposes high computational and bandwidth overhead.

We use a technique fundamentally different from threshold signatures and threshold symmetric cryptography, *i.e.,* we leverage the idea of "majority voting," one dating to von Neumann [51]. One reason why we can use simple majority voting is that non-faulty execution replicas always provide randomized yet identical ciphertexts using a consistent random coin (which is output by an underlying randomized BFT protocol [43]—detail coming shortly). As a result, our privacy firewall construction is *unkeyed* and *cryptography-free* (assuming authenticated channels); it can be instantiated using much less expensive devices.

III. Preliminary and System Model

Notation. If n is an integer then $[1..n]$ denotes the set $\{1, \cdots, n\}$. If x is a string then $|x|$ denotes its length. 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. We write $z \leftarrow \mathcal{A}^{\mathcal{O}_1, \mathcal{O}_2, \cdots}(x, y, \cdots)$ to indicate the operation that runs $\mathcal{A}$ having access to oracles $\mathcal{O}_1, \mathcal{O}_2, \cdots$ on inputs $x, y, \cdots$ and outputs z. A function $\epsilon(l)\colon \mathbb{N} \to \mathbb{R}$ is *negligible* if, for any positive number d, there exists some constant $l_0 \in \mathbb{N}$ such that $\epsilon(l) < (1/l)^d$ for any $l > l_0$.

A. Cryptographic Primitives

We provide formal definitions of security for the cryptographic primitives used in the paper. All these primitives fall into the category of symmetric cryptography.

PRF. A pseudorandom function (PRF) is an efficient keyed deterministic map $\mathsf{F}\colon \mathcal{K} \times \mathcal{D} \to \mathcal{R}$ where $\mathcal{K}$, $\mathcal{D}$, and $\mathcal{R}$ are key space, domain, and range respectively. Let $\mathcal{K} = \{0,1\}^l$ for simplicity. We define the PRF advantage of an adversary $\mathcal{A}$,

$\mathbf{Adv}^{\text{prf}}_{\mathsf{F},\mathcal{A}}(1^l)$, by

$$\Pr[k \xleftarrow{\$} \mathcal{K} \colon \mathcal{A}^{\mathsf{F}(k,\cdot)}(1^l) = 1] - \Pr[\mathcal{A}^{\$(\cdot)}(1^l) = 1]$$

The oracle $\mathsf{F}(k,\cdot)$, on input $m \in \mathcal{D}$, returns $\mathsf{F}(k,m)$. The oracle $\$(\cdot)$, on input $m \in \mathcal{D}$, returns $|\mathsf{F}(k,m)|$ random bits. If input $m \notin \mathcal{D}$, both oracles return $\perp$.

In practice, AES is assumed to be a good PRF (and also a good PRP, or pseudorandom permutation) with the same input and output length (typically 128, or 256). With appropriate assumptions, HMAC [7] is a provably secure PRF with arbitrary input length. It is easy to extend any PRF with some fixed output length to a PRF with arbitrary output length.

Symmetric encryption schemes. A symmetric encryption (SE) scheme is a triple of algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$. The key generation algorithm Gen takes as input a security parameter l and outputs a key $k \in \mathcal{K}$ (*key space*). The encryption algorithm Enc takes as input a key k and a message $m \in \mathcal{M}$ (*message space*), and outputs a ciphertext c. Enc can be probabilistic or stateful. The deterministic decryption algorithm Dec takes as input a key k and a ciphertext c, and outputs a message m. The security notion we consider for an SE is IND\$ [45]: ciphertexts should be indistinguishable from random bits. It is easily shown to imply all conventional chosen-plaintext attack formulations of indistinguishability and semantic security. We define $\mathbf{Adv}^{\text{ind\$}}_{\text{SE},\mathcal{A}}(1^l)$ as

$$\Pr[k \xleftarrow{\$} \mathsf{Gen}(1^l) \colon \mathcal{A}^{\mathsf{Enc}(k,\cdot)}(1^l) = 1] - \Pr[\mathcal{A}^{\$(\cdot)}(1^l) = 1]$$

The oracle $\mathsf{Enc}(k,\cdot)$, on input $m \in \mathcal{M}$, returns $\mathsf{Enc}(k,m)$. The oracle $\$(\cdot)$, on input $m \in \mathcal{M}$, returns $|\mathsf{Enc}(k,m)|$ random bits. If input $m \notin \mathcal{M}$, both oracles return $\perp$.

An authenticated encryption (AE) scheme has the same syntax as a conventional encryption scheme except that the deterministic decryption algorithm returns either a message $m \in \mathcal{M}$ or else the distinguished symbol $\perp$ ("invalid"). AE provides both the privacy (we consider IND\$ security) and authenticity (*i.e.,* no adversary can forge a new and valid ciphertext).

When a message needs to be privacy-protected and authenticated, there may exist additional information (termed *associated data*) such as a packet header, which must be authenticated. An authenticated encryption scheme with associated-data (AEAD) [44], compared to an AE scheme, additionally takes as input a header H in its encryption and decryption algorithms. The privacy notion requires that ciphertexts be indistinguishable from random bits except that the header is in the clear. The authenticity notion requires no adversary be able to forge either a new and valid ciphertext or an old ciphertext with a new header.

Authenticated channels. Unless otherwise stated, we assume that messages are authenticated, which can be easily realized using message authentication code (MAC). This assumption has two implications: first, it eliminates the forgery of messages on the links between two correct processes; second, with authenticated channels, the IND\$ secure encryption scheme proposed in this paper is also secure in the sense of AE or AEAD.

B. BFT State Machine Replication

We consider BFT replication problem where faulty replicas can behave arbitrarily and a computationally bounded adversary can coordinate faulty replicas to compromise the system.

A BFT protocol must satisfy safety (*i.e.,* integrity) and liveness (*i.e.*, availability). Safety requires that a service correctly process clients' requests. Liveness means that a service operates without interruption. We consider practical BFT protocols whose safety holds in any asynchronous environment, where messages may be delayed, dropped, altered, or delivered out of order, and whose liveness is ensured assuming *partial synchrony* [28]: synchrony holds only after some unknown global stabilization time, and the bounds on communication and processing delays are themselves unknown.

```
10  algorithm R-BFT_{p_i,Π}(k, ⟨REQUEST⟩)                {at replica p_i ∈ Π}
11   on receiving ⟨REQUEST, id, t, op, oi⟩                {oi : op input}
12    (s,Assign(⟨REQUEST⟩,n)) ←$ p_i^Π(s,⟨REQUEST⟩)        {s : p_i's state}
13    if flag(op) = 0 then                                {if op is deterministic}
14      (s, ⟨REPLY⟩) ← op(s, oi)
15    else
16      r ← F(k,(id, t))  {preparing coins if op is non-deterministic}
17      (s, ⟨REPLY⟩) ← op(s, r, oi)
18    send ⟨REPLY⟩
```

Fig. 1. R-BFT server-side algorithm for $p_i \in \Pi$. **All the replicas share the same PRF key—k.** Line 12 represents the conventional interactive agreement procedure where an agreed sequence number n is assigned to $\langle\text{REQUEST}\rangle$ at replica p_i. In line 14, each time an operation is executed replica p_i's (internal) state s will be updated.

Randomized BFT (RBFT). In (BFT) state machine replication, to keep states consistent, it is necessary to require that operations for state machines be deterministic. But it is easy to enhance conventional BFT protocols to handle some randomized operations, as first described in BASE [43] (see also [17]).

In particular, we need to feed "good" random coins to state machines. We identify two requirements for random coins: *uniformity* and *matching*. By uniformity, we mean that the coins that correct replicas have for any operations are distributed uniformly at random over some prescribed domain. Note that the property does not involve cryptography or adversaries. By matching, we mean that every correct process outputs the same coin value for the same operation.

We describe a RBFT protocol, R-BFT, satisfying the uniformity and matching properties. The basic idea is to run a predetermined PRF $\mathsf{F} \colon \mathcal{K} \times \mathcal{D} \to \mathcal{R}$ taking as input the *request-specific information* to obtain the *request-specific random coin*, which is fed to state machines. Concretely, all the state machines initially need to maintain the same PRF F and share the same PRF key $k \in \mathcal{K}$. By *request-specific information*, we mean certain information contained in the client request message m (the timestamp t and the client identifier id) which is used to bind the request. It serves as a request *unique*

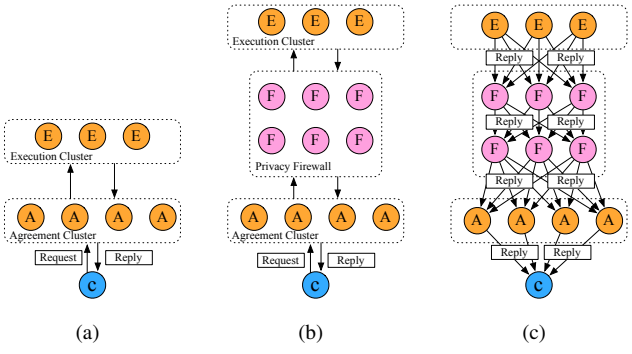


Fig. 2. **YMVAD SBFT and CBFT Architectures**. Fig. 2(a) presents the SBFT architecture that separates agreement from execution. Fig. 2(b) illustrates the CBFT architecture in a black-box manner; it consists of an agreement cluster (AC), a privacy firewall (PF), and an execution cluster (EC). Fig. 2(c) presents the concrete instantiation of YMVAD—C-BFT$_0$, where AC is reused as a row of PF. It tolerates one execution replica failure and two filter node failures.

identifier. The coins are matching for all the replicas as the PRF function is deterministic and its inputs are fixed and the same. The coins are uniformly distributed at random as the underlying PRF is indistinguishable from a random function and input for each request is uniquely identified. We then feed the PRF value as the needed coin each time it executes a client request operation op on the operation input oi. Note that the above procedure is *independent* of the underlying consensus procedure, so it can be combined with *any* existing BFT protocols. R-BFT algorithm is depicted in Fig. 1, running by a set of N replicas $\Pi = \{p_0, \cdots, p_{N-1}\}$.

IV. C-BFT$_0$ Attack Analysis

In this section we review YMVAD CBFT protocol—C-BFT$_0$ [53] and provide an attack analysis.

Review of C-BFT$_0$. In a traditional BFT architecture, replicas first agree on a linearizable order of client requests and then execute these requests. YMVAD [53] showed how to separate agreement from execution in BFT protocols (hereinafter SBFT), as depicted in Fig. 2(a). Agreement cluster (AC) runs a conventional BFT protocol and assigns unique sequence numbers to client requests. AC also generates cryptographically-verifiable *agreement certificates*. AC uses $3f + 1$ agreement replicas to tolerate at most f failures. Then execution cluster (EC) verifies the agreement certificates and executes the requests in sequence number order. These execution replicas do not communicate with one another and they execute the requests independently. EC uses $2g + 1$ execution replicas to tolerate at most g failures. While SBFT slightly increases the latency of BFT services, it also brings two immediate benefits: first, it reduces the number of execution replicas which might be expensive application-specific devices such as supercomputers and network routers; second, the SBFT architecture makes CBFT achievable.

It is easily seen that if some execution replicas and agreement replicas collude, EC state information will be leaked. To achieve confidentiality, they introduced *privacy firewall* (PF) which lies in between AC and EC to filter *incorrect* and *inconsistent* information (see Fig. 2(b)). Any client request message and server reply message must go through PF. As depicted in Fig. 2(c), YMVAD uses $h+1$ rows of $h+1$ filter nodes per row to tolerate at most h faults, while providing safety, liveness, and confidentiality. YMVAD heavily relies on threshold signature to ensure these properties. At the beginning, each replica in EC executes a client request and returns a server reply. For a deterministic operation, non-faulty replicas will share the same reply. Each replica in EC also generates a signature share for a $(g + 1, 2g + 1)$ threshold signature on the reply such that $g+1$ or more valid signature shares can be used to reconstruct a single threshold signature. Each replica in EC then broadcasts the reply and its signature share to the first row of PF. If no more than g execution replicas fail, each non-faulty filter node at the first row of PF will be able to reconstruct a threshold signature on the reply. The reply and the reconstructed signature are then subsequently sent to all filter nodes at the next row which need to verify the correctness of the signature. Then they are sent to next row and so on. For YMYAD, there are $h+1$ rows of $h+1$ filter nodes per row. It is not hard to see that (1) there exists at least one correct *path* between AC and EC which only consists of correct filters and (2) there exists one row which only consists of correct filter nodes and the rows below do not have any information from any replicas in EC (whether benign or malicious). The first observation ensures liveness even with faulty filter nodes and the second observation prevents faulty filter nodes from leaking information.

It is important to note that if $h \leq 3f$, agreement replicas can be re-used as a PF row.

Confidentiality analysis. YMVAD did mention "[R]equest and reply bodies ... are encrypted so that the client and execution nodes can read them but agreement nodes and firewall nodes cannot." However, it did not specify how to instantiate the encryption schemes.

In the reduction-based sense of modern cryptography, any provably secure encryption scheme must be probabilistic (or stateful). Otherwise, an adversary can at least tell if two ciphertexts can be decryptable to the same message. It would cause at least two problems if a probabilistic encryption scheme were trivially adopted in C-BFT$_0$ to encrypt server replies. First, while randomness is indispensable to a secure encryption scheme, C-BFT$_0$ currently does not enable randomization. Second, (even) if the execution replicas were able to run randomized algorithms in the conventional sense, it would invalidate the signing algorithm of threshold signature. This is because the encrypted replies for a request from the execution replicas should *all* be different after applying the probabilistic encryption, but in order to reconstruct the threshold signature, more than a half of the signature shares are shares signed for the same encrypted reply.

The counters or nonces based encryption might well work, but replicas need to agree with them. Also, one needs to handle associated-data.

Alternative deterministic solutions, such as deterministic en-

cryption [6] and message-locked encryption [8], only provide adequate security for messages with high min-entropy.

Safety analysis. We find that C-BFT_0 cannot handle faulty clients. A single malicious client may easily attack the safety of the replicated system, even if the underlying BFT in AC is secure against Byzantine clients.

In YMVAD's architecture, when each execution replica receives the encrypted request message, it would decrypt it independently. With a trivial encryption scheme, the ciphertexts sent to execution replicas may not correspond to the same underlying message. Therefore, if following the protocol specification and executing the requests, execution replicas would be in an inconsistent state.

YMVAD neither addressed the issue nor did it discuss any instantiations of the encryption scheme used to encrypt client requests. This is no slight overlook, and to give an example among many, we consider the most trivial instantiation where we assume that a client shares a pairwise key with each execution replica, an assumption used to establish authenticated channels. In this setting, using a secure encryption scheme to encrypt the requests with pairwise keys fails to work, since decryption is performed independently and each execution replica cannot tell if others receive the same request.

Since there were no formal definitions of security with respect to CBFT, our discussion regarding C-BFT_0 has so far been informal. We therefore provide formal definitions.

V. Reformulating CBFT

We follow YMVAD's architecture (Fig. 2(b)) where there is an AC, a PF, and an EC. In the following we assume that messages transmitted respect the message flow of the architecture, *i.e.*, messages transmitted between AC and EC must go through PF. We emphasize that this is a strong assumption and the architecture needs to be carefully deployed. The contents of this section is a strengthening and extension of YMVAD, providing a rigorous syntax, refining prior security requirements, and adding new ones.

Syntactically, a CBFT scheme consists of the following algorithms (Init, CEnc, ACA, ECE, PFF, CDec). The CBFT initiation algorithm Init takes as input a security parameter l, initializes the system and establishes all the necessary cryptographic keys (*i.e.*, the keys for encrypting client requests and server replies and the keys for achieving Byzantine fault tolerance in AC). A client encrypts a request via CEnc and sends it to AC. The AC agreement algorithm ACA runs an interactive BFT protocol to assign a sequence number n to the client request and generates an agreement certificate. The agreed, encrypted request and agreement certificate will then be passed through PF to EC. The EC execution algorithm ECE takes as input the information from AC, verifies the correctness of the certificate, decrypts the client request, executes the underlying operation *op* with the underlying operation input oi, and sends an encrypted reply message to PF. The PF filtering algorithm PFF is used to filter inconsistent and malicious information sent from faulty replicas from EC. Client uses CDec to decrypt the encrypted reply output from PF.

Property I. *Request and reply privacy:* Adversary can learn no information about client requests and server replies while they are transmitted.
Property II. *Soundness:* Adversary cannot generate encrypted client requests that can be decrypted into different client requests.
Property III. *Filter liveness:* Consistent messages from a majority of execution replicas can go through the filter.
Property IV. *Filter safety:* Inconsistent (and malicious) messages from a minority of execution replicas are filtered.

Fig. 3. Confidentiality requirements.

As described in Fig. 3, we will decompose the confidentiality-related notions into the following security requirements: request and reply privacy, soundness, filter liveness, and filter safety.

All the requirements described in Fig. 3 can be rigorously formalized such that the computations made by the honest parties and the adversary are modeled as probabilistic interactive Turing machines. We assume that adversary can corrupt replicas non-adaptively, as in all the other practical BFT protocols.

When discussing these notions, agreement replicas take the same responsibility as the filter nodes. We therefore do not distinguish filter nodes from agreement replicas, and only consider filter nodes for discussion. We assume that the adversary can corrupt no more than g execution replicas and h filter nodes, and learn their internal states.

We take a modular approach to define the security of CBFT; namely, we separate PF definitions of security from others. When defining other definitions of security, we can treat PF as a black box that respects filter liveness and filter safety; it is convenient to regard PF as a single correct node. We begin by PF related notions.

PF Syntax and Definitions. Our privacy firewall notions, filter liveness and filter safety, are rather specific, being tied to our system architecture. We can be more general, defining a privacy firewall in a way that depends on *policies*, and then applying it to our case.

A privacy firewall $\mathcal{PF}$ is a pair of algorithms (Filter, Extract) with respect to policy $\mathcal{P}$, where $\mathcal{P}$ is a set of predicates. The *filter algorithm* Filter takes as input a predicate $P \in \mathcal{P}$ and b values $X = (x_1, \cdots, x_b)$ to be filtered, and outputs a value σ. Filter may be *probabilistic*, in which case it additionally takes as input a random coin. Filter may also be *keyed*, in which case there is an additional key setup mechanism and Filter needs to take the key as input too. (YMVAD PF is keyed as it relies on threshold signatures, but our PF that will be described in §VI is unkeyed if assuming authenticated channels.) The *extract algorithm* Extract is deterministic and unkeyed. It takes as input σ and P and outputs the (desired) result.

In YMYAD's architecture (and ours below), the policy set contains only a single predicate, and the Filter functionality is to output the majority of results with the same logical timestamp, and output $\perp$ if there is no majority. Extract is also a majority function run by clients. We can rigorously define

PF security notions following this syntax. Filter liveness, in this context, simply means that on input $X = (x_1, \cdots, x_b)$ chosen by adversary, the extracted value from σ must be a majority result of X (or $\perp$ if there is no majority). Filter safety ensures that on input X chosen by adversary, σ cannot leak any information other than the majority value of X.

Defining soundness in multicast encryption. Among all these confidentiality-related notions, it is less intuitive how to formalize the soundness notion, which is used to thwart the safety attack. We consider the scenario where a (possibly malicious) client sends a ciphertext to a set of recipients (corresponding to execution replicas in the setting of CBFT). The essence of the *soundness* security notion is to ensure that each execution replica will have the same underlying plaintext after decrypting the encrypted client requests. The notion is a strengthening of the conventional *correctness* notion of an encryption scheme.

We introduce a primitive—d-multicast encryption and then formalize its soundness notion. We will use it to achieve CBFT soundness. The definition makes sense in both the symmetric and asymmetric cryptographic settings, but for simplicity, we consider only the former. Let $\mathcal{E} = (\mathsf{MGen}, \mathsf{MEnc}, \mathsf{MDec})$ be a *d-multicast encryption* scheme (to distinguish from the conventional broadcast encryption) that encrypts a message $m \in \mathcal{M}$ (message space) to d recipients. The key generation algorithm MGen takes as input a security parameter l and outputs d symmetric keys k_i for $i \in [1..d]$ shared between a sender and each of the recipients. The keys might be distributed via a trusted authority, generated independently of one another, or obtained by running an interactive key exchange/generation algorithm. The sender encryption algorithm MEnc takes as input a message m and d keys, and outputs a ciphertext c. It can be either probabilistic or stateful. The deterministic recipient decryption algorithm MEnc takes a ciphertext c and a key k_i ($i \in [1..d]$) and outputs a plaintext m.

We impose the basic *invertibility* requirement on a d-multicast encryption scheme: MEnc is injective on the message space $\mathcal{M}$. We define a strong form of *soundness* notion: any malicious adversary cannot "cheat" by creating a ciphertext which can be decryptable to different plaintexts by different recipients. In particular, it cannot be the case where one replica decrypts the ciphertext to a message $m \in \mathcal{M}$, while the other decrypts it to $\perp$ (*i.e.,* an invalid ciphertext).

Formally, we consider in Fig. 4 the experiment that is associated to an adversary $\mathcal{A}$ and a symmetric-key d-multicast encryption scheme $\mathcal{E} = (\mathsf{MGen}, \mathsf{MEnc}, \mathsf{MDec})$.

We define the *soundness-advantage* of $\mathcal{A}$ in the above experiment as $\mathbf{Adv}^{\text{sound}}_{\mathcal{E},\mathcal{A}}(1^l) = \Pr[\mathbf{Exp}^{\text{sound}}_{\mathcal{E},\mathcal{A}}(1^l) = 1]$. We say that $\mathcal{E}$ satisfies the *soundness* property if for any adversary $\mathcal{A}$, we have that $\mathbf{Adv}^{\text{sound}}_{\mathcal{E},\mathcal{A}}(1^l)$ is negligible in the security parameter l, where the probability is taken over the choice of $(k_1, \cdots, k_d) \xleftarrow{\$} \mathsf{MGen}(1^l)$ and over the internal coins, if any, of $\mathcal{A}$. Notice that we do not need to provide the adversary with the decryption oracle since all the secret keys are already given. We can let the adversary be either computationally bounded or unbounded. If the advantage is always equal to 0, we say that $\mathcal{E}$ has the *perfect soundness* property.

Experiment $\mathbf{Exp}^{\text{sound}}_{\mathcal{E},\mathcal{A}}(1^l)$
$(k_1, \cdots, k_d) \xleftarrow{\$} \mathsf{MGen}(1^l)$
$c \xleftarrow{\$} \mathcal{A}(1^l, k_1, \cdots, k_d)$
for $i, j = 1$ **to** d **do**
  if $i \neq j$ **and** $\mathsf{MDec}(k_i, c) \neq \mathsf{MDec}(k_j, c)$ **then**
    return 1
return 0

Fig. 4. Soundness experiment with respect to an adversary $\mathcal{A}$ and a d-multicast encryption scheme $\mathcal{E} = (\mathsf{MGen}, \mathsf{MEnc}, \mathsf{MDec})$.

While there exist encryption schemes for multiple recipients such as broadcast encryption, they fail to define any soundness notion. A malicious adversary may yield ambiguous ciphertexts that can be decryptable to different plaintexts by different recipients. To our knowledge, the issue has not been pointed out in broadcast encryption literature, but we find that many (but not all) existing chosen ciphertext attack (CCA) secure broadcast encryption constructions can defend against the attack. We do not use them as they are too costly for our purpose and we can achieve the same goal more efficiently. We also note that Chow, Franklin, and Zhang [21] defined a similar notion in the case of two recipients for a different purpose.

Strictly speaking, the soundness property is implied by the safety property. However, one must satisfy soundness to achieve safety, and we choose to single out the soundness security notion to highlight its importance.

It is not difficult to provide a secure multicast encryption even when the keys for different recipients are generated independently of one another. But as we will show, key independence is not a requirement to achieve a secure CBFT protocol.

Request privacy. The *request privacy* notion simply guarantees privacy for request transmission. We consider the notion of "indistinguishability from random bits under an adaptive chosen-plaintext attack" for our d-multicast encryption scheme $\mathcal{E} = (\mathsf{MGen}, \mathsf{MEnc}, \mathsf{MDec})$ with message space $\mathcal{M}$. We define $\mathbf{Adv}^{\text{req-priv\$}}_{\mathcal{E},\mathcal{A}}(1^l)$ as

$$\Pr[(k_1, \cdots, k_d) \xleftarrow{\$} \mathsf{MGen}(1^l) \colon \mathcal{A}^{\mathsf{MEnc}(k_1, \cdots, k_d, \cdot)}(1^l) = 1] - \Pr[\mathcal{A}^{\$(\cdot)}(1^l) = 1]$$

The oracle $\mathsf{MEnc}(k_1, \cdots, k_d, \cdot)$, on input $m \in \mathcal{M}$, returns $\mathsf{MEnc}(k_1, \cdots, k_d, m)$. The oracle $\$(\cdot)$, on input $m \in \mathcal{M}$, returns $|\mathsf{MEnc}(k_1, \cdots, k_d, m)|$ random bits. If input $m \notin \mathcal{M}$, both oracles return $\perp$.

As we assume authenticated channels, we do not need to consider authenticity notions or chosen-ciphertext attacks.

Reply privacy. The *reply privacy* notion aims to to ensure privacy for reply transmission. Again, we consider the notion of "indistinguishability from random bits" but simply for a conventional symmetric encryption scheme with two parties sharing a pairwise key (see §III). The reason why modeling

for a conventional symmetric encryption is sufficient is that due to use of PF, any adversary attacking reply privacy can only see a single ciphertext output by the PF. However, it can still mount chosen-plaintext attacks.

VI. CBFT Construction

A. Overview of C-BFT

C-BFT made three significant improvements to C-BFT$_0$. First, C-BFT is provably secure under the definitions we formally described. Second, C-BFT eliminates all the public-key cryptographic operations. Third, C-BFT supports arbitrary randomized processing.

We obtain these results by extending the idea of R-BFT, leveraging an AEAD scheme (to achieve soundness), designing a tailored, length-preserving encryption scheme for server replies, and inventing a novel, cryptography-free PF.

Our PF takes $h+1$ rows of $2h+1$ filter nodes per row to tolerate h Byzantine node failures. When $h=1$ (the most applicable setting), it takes only one more filter node than that of C-BFT$_0$—note that the agreement replicas can be re-used in both cases. Assuming authenticated channels, filter nodes do not *even* need to perform any cryptographic operations. Thus, our PF can be instantiated using much cheaper devices.

B. More Intuition

A crucial observation is that to achieve CBFT, each client does not necessarily need to maintain different keys with different execution replicas. In fact, sharing one common group key does not damage the *request and reply privacy* notion due to the soundness property and the use of PF. For our construction, each client shares with all the execution replicas the same symmetric keys k_1 (for encrypting client requests) and k_2 (for encrypting server replies). The keys can be generated as part of the CBFT Init algorithm, either pre-distributed via a trusted authority, or generated via an authenticated group key exchange protocol [11].

To encrypt a client request message, a client uses an AEAD scheme to form a *single* ciphertext with the key k_1, such that information like the client identifier and the unique timestamp—which replicas in AC will use for agreement—is authenticated but in the clear. The soundness property easily follows in this setting: for any single AEAD ciphertext, all the recipients (*i.e.*, execution replicas) sharing the same key will either all reject the ciphertext or all decrypt it into the same plaintext. The scheme actually satisfies perfect soundness as defined: the soundness-advantage of any adversary is 0. We comment that many other tempting encryption methods (still) simply fail (besides the one we mentioned in §IV). One such incorrect method is that the client and all the execution replicas share the same key for encryption, while the client and each execution replica share a pairwise key for MAC authentication. It is insecure as it is possible that some of execution replicas having valid MACs can decrypt its own share, while others having invalid ones simply reject the ciphertext.

Our approach to encrypting server reply messages at the execution replicas is to generate random coins via the same method as in RBFT and then feed them to a symmetric encryption mode of operation. Specifically, the execution replicas can *directly* derive random coins, by applying a PRF on request-specific information as defined in RBFT (the timestamp t and the client identifier id) and on *agreement information* (the view number v and the agreed sequence number n from AC). This time, agreement information must be included, because agreement and execution procedures are separated. In addition, instead of deriving random coins and feeding them into an existing symmetric mode of operation, we adopt a "tailored," *length-preserving* encryption scheme. Moreover, we must yield random coins to support randomized operations too. This is no problem given a PRF with the output-length the same as the length of all the needed coins.

C. C-BFT Algorithms

The C-BFT algorithms are described in Fig. 5. We do not choose to include authentication in the algorithms, since we assume authenticated channels. As C-BFT$_0$, C-BFT consists of an AC, a PF, and an EC. AC uses $3f+1$ replicas (denoted o_i, $i=[1..3f+1]$) to tolerate f failures, PF uses $h+1$ rows of $2h+1$ filter nodes per row (denoted $p_{i,j}$, $j=[1..h+1]$, $i=[1..2h+1]$) to tolerate h failures, and EC uses $2g+1$ replicas (denoted q_i, $i=[1..2g+1]$) to tolerate g failures.

We assume without loss of generality that the underlying BFT protocol run by AC is sequencer-based [26], where there is a designated primary that is responsible for ordering client requests. Let id be the identifier of a client. Let v be the current view number for the BFT protocol run by AC. Let $m=(id,t,op,oi)$ be a request message from client id, where t is the timestamp, op is the operation, and oi is the operation input. Let $\mathcal{IA}$ be an interactive agreement protocol run by replicas Π in AC that takes as input an incoming encrypted client request $\langle\text{REQUEST}\rangle$ and their local state sa, and outputs an agreed sequence number n assigned to $\langle\text{REQUEST}\rangle$, a corresponding certificate cert_n, and an updated state. An execution replica runs Vrf to verify the correctness of cert_n and executes the operation op. The function op takes as input a state se, a random coin r_1, and an operation input oi, and outputs a result u and an updated state. In the ECE algorithm, a PRF F will be used to generate a request-specific randomness r_1 to enable randomized operations, and a randomness r_2 to mask the execution result to achieve confidentiality.

Let $\mathcal{E}=(\mathsf{Gen},\mathsf{Enc},\mathsf{Dec})$ be an AEAD scheme. We build our d-multicast encryption scheme $\mathcal{E}'=(\mathsf{MGen},\mathsf{MEnc},\mathsf{MDec})$ from $\mathcal{E}$ such that all the corresponding algorithms are *identical*. Let $\mathsf{F}\colon \mathcal{K}\times\mathcal{D}\to\mathcal{R}$ be a PRF such that $\mathcal{K}=\{0,1\}^l$, $\mathcal{D}=\{0,1\}^{|id|+|t|+|v|+|n|}$, and $\mathcal{R}=\{0,1\}^{|r_1|+|r_2|}$. Initially, we generate and pre-distribute two group symmetric keys shared among a client id and all the execution replicas. Let $k_1 \xleftarrow{\$} \mathsf{Gen}$ be the key for a multicast encryption scheme and $k_2 \xleftarrow{\$} \{0,1\}^l$ be the PRF key. $H=(t,id)$ is the header, and $e=u\oplus r_2$ is the ciphertext. (H,e) will be sent to PF by each replica in EC. Using authenticated channels built from PRF-based MACs, the encryption is secure in the sense of an AEAD [44].

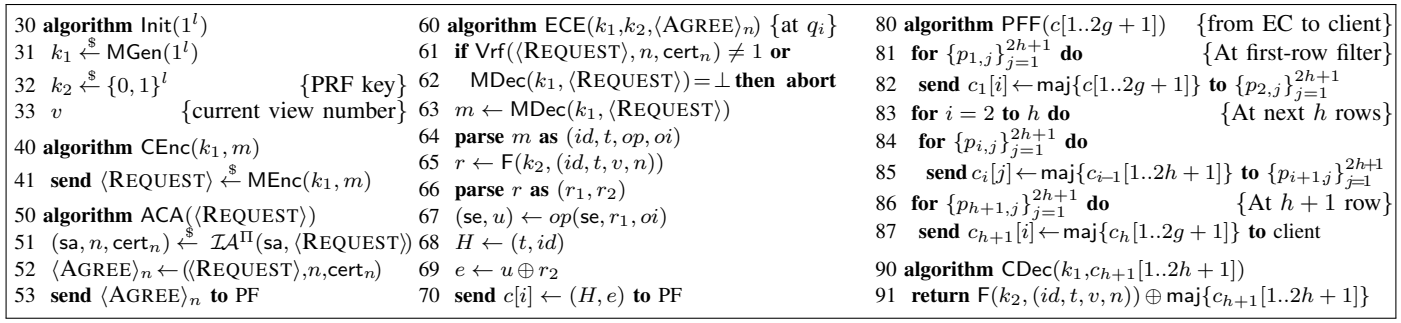


Fig. 5. C-BFT = (Init, CEnc, ACA, ECE, PFF, CDec). We only include the PFF algorithm from EC to AC (lines 80–87), as the one for the other direction (from AC to EC) is symmetric.

Our PF does not rely on a distributed threshold signature scheme; instead, it uses an efficient *majority vote* procedure to forward the correct and consistent information from a majority of execution replicas, and filter the malicious and inconsistent information that is not from a majority of execution replicas. Assuming authenticated channels, filter nodes do not need to perform any cryptographic operations.

Let's describe our PF in more detail. First, each filter node at the first row waits for $g+1$ consistent ciphertexts with the same timestamp from a majority of execution replicas to decide a correct ciphertext. Then each filter node will broadcast the determined ciphertext to the filters at the next row. At each of the next h levels in PF, we take the same majority vote procedure to determine the correct ciphertext. On the one hand, since there are $2h+1$ filter nodes at each level and there are at most h malicious filter nodes, there must exist a majority of nodes which are correct. This ensures *filter liveness*, *i.e.,* the consistent messages (from a majority of execution replicas) can always be forwarded. On the other hand, since there are $h+1$ levels of filter nodes and there are at most h malicious filter nodes, there must exist a level i of nodes which are all correct. All the inconsistent and malicious information can be filtered by this level of nodes, and all the malicious filter nodes at the levels greater than i do not have any information other than the consistent and determined ciphertext (from a majority of non-faulty execution replicas). Thus, *filter safety* is satisfied.

Upon receiving $c_{h+1}[1..2h+1]$ from the last row of PF, the client sets the ciphertext c as $\mathsf{maj}\{c_h[1..2h+1]\}$, where $\mathsf{maj}\{\cdot\}$ is a majority function. (This corresponds to the Extract function of PF syntax in §V.) To decrypt, the client computes $u = \mathsf{F}(k_2, (t, id, v, n)) \oplus \mathsf{maj}\{c_{h+1}[1..2h+1]\}$.

Theorem 1. *If no more than f agreement replicas, h filter nodes, and g execution replicas are faulty, then* C-BFT *is a secure CBFT protocol.*

All proofs can be found in our full paper.

D. Application to Causality-Preserving Replicated Service

We show how our CBFT protocol (with refined and enhanced security notions we defined) can be used to build an efficient and secure replicated service that respects request causality in the sense of Reiter and Birman [42] and Cachin, Kursawe, Petzold, and Shoup [16].

Causality-preserving replicated service. We begin by describing a request causality-preserving replicated service. When a client issues a request to the replicated service, some faulty replicas may create a new request which will be executed before the request of the client. This will violate the causal order of client requests. To illustrate the problem, Reiter and Birman [42] 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 a corrupt client to issue a new request for the same stock. If the correct replicas deliver the new request before that 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. Consider another interesting service that registers names on a "first come, first served" manner [14]. A faulty replica may observe an interesting name being registered, and it may then register the name for another client.

Prior constructions. Reiter and Beirman [42] provided a construction from an atomic broadcast protocol and a threshold cryptosystem. The client can encrypt the client request with a $(f+1, 3f+1)$ threshold public-key encryption system, where f is the maximum number of faults the system can tolerate. After reaching an agreement on a ciphertext, each replica first uses its corresponding decryption key to compute its decryption share, and then sends the decryption share to the rest of the replicas. Each replica must wait for $f+1$ decryption shares before it can recover the client request and then deliver the request. Cachin, Kursawe, Petzold, and Shoup [16] revisited the problem by rigorously formalizing the problem and built a construction secure against CCA adversaries from a labeled CCA threshold cryptosystem [50]. Clearly, both constructions heavily rely on public-key threshold cryptosystem. This makes the protocols less efficient, and moreover makes it difficult to integrate the construction into an existing, practical BFT protocols such as PBFT [20] and Zyzzyva [32] which use only symmetric cryptography.

We claim that our CBFT protocol immediately leads to a causality-preserving Byzantine replicated service.

Theorem 2. *Any CBFT protocol that satisfies security notions in §V is also a causality-preserving BFT protocol.*

Note that C-BFT$_0$ does not preserve causality, as no encryption has been considered in their protocol. It is easy to see that

a faulty agreement replica can see the underlying request during their transmission and then forge a derived request. We have shown that our CBFT protocol—C-BFT can be achieved using only symmetric cryptography. Hence, we obtain the first causality-preserving BFT protocol that uses only symmetric cryptography and is much more efficient than previous constructions [16, 42]. Yet one should be also aware that our causality-preserving protocol, despite being more efficient, relies on a specific system architecture.

VII. Implementation and Evaluation

This section studies the performance of R-BFT, and compares them with unreplicated services, the underlying BFT protocol, R-BFT, S-BFT [53] (an implementation that separates agreement from execution), and C-BFT_0, on microbenchmarks.

A. Experimental Setup and Implementation

Our test setting comprises a replica cluster of up to 15 machines (2.13GHz Xeon processor, 4GB RAM) that are connected through a 100Mbps switched LAN.

We utilize P-BFT [20], the first practical and most mature BFT protocol, as our baseline and as the underlying BFT protocol to instantiate more complex BFT protocols—R-BFT, S-BFT [53], C-BFT_0 [53], and C-BFT. All these protocols are evaluated in the configurations that can tolerate a minimum number of faulty replicas. Both P-BFT and R-BFT can tolerate one faulty replica. S-BFT can tolerate one faulty execution replica and one faulty agreement replica. Both C-BFT and C-BFT_0 can tolerate one faulty execution replica, one faulty filter node, and one faulty agreement replica.

We use HMAC as the underlying PRF and as the MAC algorithm to build authenticated channels. It is important to use distinct keys to ensure provable security though. We use a composed AEAD scheme to instantiate the multicast encryption. Specifically, we apply CTR mode of operation and then compute its HMAC. When instantiating C-BFT_0, we adopt an efficient, deterministic but insecure mode of operation ECB as the encryption algorithm in EC. As explained, C-BFT_0 is difficult to instantiate securely and we choose ECB just for a relatively fair comparison.

We use ThreshSig [52] which implements Shoup's RSA threshold signature scheme [49]. While there are other threshold signatures, they are either interactive or use more expensive pairing-based cryptography. Since we adopt AES-128, we use an equivalently secure RSA-3072 (following NIST recommendation [39]). For an efficiency comparison, we also test RSA with two smaller keys, where RSA-1024 has been disallowed and RSA-2048 is still acceptable for most applications. Let C-BFT_0-3072/2048/1024 denote C-BFT_0 with RSA-3072/2048/1024, respectively.

Concrete analysis is vital to evaluating modern cryptographic protocols. YMVAD did not report the security parameter. Later on, Belisarius [40] pointed out the problem and re-evaluated C-BFT_0. The authors found that their implementation on C-BFT_0, if following NIST recommendation, has a much larger overhead than the one presented in YMVAD, and therefore they reported RSA-128 in order to be in line with YMVAD. While we do understand the experimental choice of Belisarius, we comment that RSA-128 can be easily factored within a second using a commodity laptop.

B. Microbenchmark Evaluation

We assessed the latency and throughput using the x/y microbenchmarks by Castro and Liskov [20], where clients send x kB requests and receive y kB replies.

Table I shows the average latency of the protocols. We find that P-BFT and R-BFT have almost identical latency and outperform the others. Note that when testing the performance of R-BFT, we only need to additionally perform a PRF operation compared to P-BFT. Since S-BFT separates agreement from execution, it adds an additional overhead. While there is a performance impact, C-BFT's latency remains acceptable. For all the four micro-benchmarks, C-BFT is more than two to three orders of magnitude faster than C-BFT_0. The reason remains that C-BFT uses only efficient symmetric cryptography but C-BFT_0 relies on rather expensive RSA public-key threshold signatures.

Our latency experiment practically rules out the possibility of using C-BFT_0 in latency-critical applications. We go on to evaluate the throughput of the rest of protocols using 0/0 microbenchmarks, as depicted in Fig. 6(a). (The results for other microbenchmarks are similar and thus omitted.) R-BFT achieves almost the same throughput as P-BFT. C-BFT has around 50% of the throughput of P-BFT. When there are around 80 to 90 concurrent clients, the network is saturated and the throughput for all the four protocols becomes steady. We also reported the latency vs. throughput experiment in Fig. 6(b).

VIII. Conclusion

We studied a fundamental problem on how to generally store and process data reliably and privately in replicated state machines. We followed the architecture of YMVAD that separates agreement from execution. But we went one step further to provide formal definitions. Then we provided an efficient construction C-BFT satisfying them. C-BFT uses only symmetric cryptography, and fits well with the cloud computing infrastructure. As an application, we show that C-BFT can be used to build more efficient causality-preserving BFT protocols.

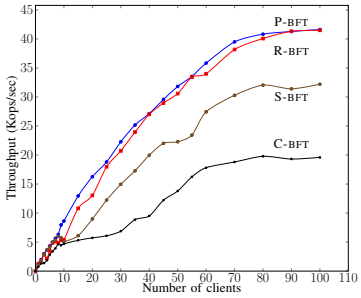
We implemented and evaluated C-BFT on microbenchmarks. We show that C-BFT is at least two to three orders of magnitude faster than C-BFT_0, which is less secure than ours and cannot generally deal with randomized algorithms.

Acknowledgement

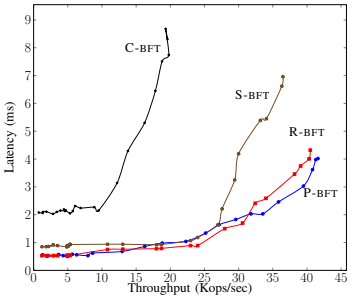
The authors are indebted to Andrew Chi, Hein Meling, Sean Peisert, Michael Reiter, our shepherd Cristian Cachin, and the reviewers for their insightful comments. Sisi was sponsored in part by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the Department of Energy. Haibin partly acknowledges NSF grant CNS 1330599 and CNS 1413996,

TABLE I
LATENCY (MS) FOR MICROBENCHMARKS.

Protocol	0/0	0/4	4/0	4/4
P-BFT	0.53	1.31	2.98	3.74
R-BFT	0.54	1.42	3.03	3.97
S-BFT	0.95	2.80	4.05	5.29
C-BFT	2.64	3.05	4.38	4.68
C-BFT$_0$-1024	37.55	38.28	39.19	40.38
C-BFT$_0$-2048	248.37	249.10	251.42	252.13
C-BFT$_0$-3072	767.43	768.20	769.20	773.19



(a) Throughput for 0/0 microbenchmark.



(b) Latency vs. throughput.

Fig. 6. Evaluation.

as well as the Office of Naval Research grant N00014-13-1-0048.

REFERENCES

[1] M. Abd-El-Malek, G. Ganger, G. Goodson, M. 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] E. Androulaki, C. Cachin, D. Dobre, and M. Vukolic. Erasure-coded Byzantine storage with separate metadata. *OPODIS 2014.*

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

[6] M. Bellare, A. Boldyreva, and A. O'Neill. Deterministic and efficiently searchable encryption. *CRYPTO '07.*

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

[8] M. Bellare, S. Keelveedhi, and T. Ristenpart. Message-locked encryption and secure deduplication. *EUROCRYPT '13.*

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

[10] A. Bessani, J. Sousa, and E. Alchieri. State Machine Replication for the Masses with BFT-SMART. *DSN '14.*

[11] E. Bresson, O. Chevassut, and D. Pointcheval. Provably-secure authenticated group Diffie-Hellman key exchange. *ACM TISSEC '07.*

[12] E. Brickell, G. Di Crescenzo, and Y. Frankel. Sharing block ciphers. *ACISP 2000.*

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

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

[15] C. Cachin, D. Dobre, and M. Vukolic. Separating data and control: Asynchronous BFT storage with $2t + 1$ data replicas. *SSS 2014.*

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

[17] C. Cachin, S. Schubert, M. Vukolic. Non-determinism in Byzantine Fault-Tolerant Replication. CoRR abs/1603.07351 (2016)

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

[19] B. Calder *et al.* Windows Azure Storage: A highly available cloud storage service with strong consistency. *ACM SOSP*, 2011.

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

[21] S. Chow, M. Franklin, and H. Zhang. Practical dual-receiver encryption: soundness, complete non-malleability, and applications. *CT-RSA 2014.*

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

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

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

[25] R. Cramer, I. Damgård, and J. Nielsen. Secure multiparty computation and secret sharing. *Cambridge University Press.*

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

[27] S. Duan, H. Meling, S. Peisert, and H. Zhang. BChain: Byzantine replication with high throughput and embedded reconfiguration. *OPODIS 2014.*

[28] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM* 35(2): 288–323, 1988.

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

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

[31] A. Iyengar, R. Cahn, C. Jutla, and J. Garay. Design and implementation of a secure distributed data repository. *14th IFIP Internat. Information Security Conf..*

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

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

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

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

[36] D. Levin, J. R. Douceur, J. R. Lorch, and T. Moscibroda.. TrInc: Small trusted hardware for large distributed systems. *NSDI*, 2009.

[37] K. Martin, R. Safavi-Naini, H. Wang, and P. Wild. Distributing the encryption and decryption of a block cipher. *DCC*, vol. 36, issue 3, pp. 263–287, September 2005.

[38] S. Micali and R. Sidney. A simple method for generating and sharing pseudo-random functions, with applications to Clipper-like escrow systems. *CRYPTO'95.*

[39] NIST publications. http://csrc.nist.gov/

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

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

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

[43] R. Rodrigues, M. Castro, and B. Liskov. BASE: using abstraction to improve fault tolerance. *ACM Trans. Comput. Syst*, 21(3): 236–269, 2003.

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

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

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

[47] N. Santos, K. Gummadi, and R. Rodrigues. Towards trusted cloud computing. *HotCloud 09.*

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

[49] V. Shoup. Practical threshold signatures. *EUROCRYPT 2000.*

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

[51] J. von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. *Automata Studies*, 1956.

[52] S. Weis and L. Kissner. Threshsig: Java threshold signatures. https://code.google.com/archive/p/threshsig/

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