RESEARCH-ARTICLE

Recipe: Hardware-Accelerated Replication Protocols: Rethinking Crash Fault Tolerance Protocols for Untrusted Cloud Environments

DIMITRA GIANTSIDI, Microsoft Research, Redmond, WA, United States

EMMANOUIL GIORTAMIS, Technical University of Munich, Munich, Bayern, Germany

JULIAN PRITZI, Technical University of Munich, Munich, Bayern, Germany

MAURICE BAILLEU, Huawei Technologies Co., Ltd, United Kingdom, Reading, Berkshire, U.K.

MANOS KAPRITSOS, University of Michigan, Ann Arbor, Ann Arbor, MI, United States

PRAMOD BHATOTIA, Technical University of Munich, Munich, Bayern, Germany

Open Access Support provided by:

Microsoft Research

Technical University of Munich

Huawei Technologies Co., Ltd, United Kingdom

University of Michigan, Ann Arbor

Published: 15 December 2025

Citation in BibTeX format

Middleware '25: 26th International Middleware Conference
December 15 - 19, 2025
TN, Nashville, USA

https://doi.org/10.1145/3721462.3770763
ISBN: 9798400715549

RECIPE: Hardware-Accelerated Replication Protocols

Rethinking Crash Fault Tolerance Protocols for Untrusted Cloud Environments

Dimitra Giantsidi
Azure Research*
Cambridge, UK

Emmanouil Giortamis
TU Munich
Munich, Germany

Julian Pritzi
TU Munich
Munich, Germany

Maurice Bailleu
Huawei Research*
Edinburgh, UK

Manos Kapritsos
University of Michigan
Michigan, USA

Pramod Bhatotia
TU Munich
Munich, Germany

Abstract

Today's modern hardware, with manycore servers, RDMA-capable networks and trusted execution environments, challenges the conventional wisdom about CFT protocols' design. We explore the synergy between modern hardware and the security and performance of strongly consistent replication protocols. Specifically, *can we leverage (and how) modern cloud hardware to harden the security properties of a CFT protocol for Byzantine settings while achieving high performance?*

We propose RECIPE, a generic approach to transform *existing* CFT protocols to tolerate Byzantine failures in untrusted cloud environments. RECIPE leverages advances in confidential computing and direct network I/O to guarantee non-equivocation and transferable authentication in the presence of Byzantine actors, while offering performance and resource overheads on par with CFT protocols. Our evaluation based on the transformation of four CFT protocols against the state-of-the-art BFT protocols shows that RECIPE can increase throughput up to 5.9×—24× and reduce the number of participating nodes by f.

CCS Concepts

• **Security and privacy → Distributed systems security**.

ACM Reference Format:

Dimitra Giantsidi, Emmanouil Giortamis, Julian Pritzi, Maurice Bailleu, Manos Kapritsos, and Pramod Bhatotia. 2025. RECIPE: Hardware-Accelerated Replication Protocols Rethinking Crash Fault Tolerance Protocols for Untrusted Cloud Environments. In *26th ACM Middleware Conference (Middleware '25), December 15–19, 2025, Nashville, TN, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3721462.3770763

1 Introduction

Replication protocols play the foundational role in designing diverse distributed systems [2, 4, 5, 9–12, 14, 20, 25, 135, 137, 148]. For performance and fault tolerance, distributed systems employ Crash Fault Tolerant (CFT) *replication protocols* [29, 93, 101, 107, 121, 131, 132, 144] guaranteeing a consistent view of the datasets as well as reliability and availability when failures occur [12, 55, 66, 70, 86, 99].

CFT protocols assume a *fail-stop model*, i.e., replicas are honest and only fail by crashing [58]. As such, they are *inadequate* for modern untrusted cloud environments, where the cloud infrastructure can be compromised by adversaries, e.g., co-located tenants or a misbehaving cloud operator that eavesdrop or actively influence the replicas' behavior. In such environments, the surface of faults and attacks expands beyond the CFT fail-stop model [76, 78, 138]. CFT protocols are fundamentally incapable of providing consistent replication under non-benign (*Byzantine*) faults in the untrusted cloud.

Byzantine Fault Tolerant (BFT) protocols [102] offer core foundations for building distributed systems with stronger guarantees under *Byzantine failures*, i.e., nodes fail in arbitrary ways. However, they are *not adopted in practice* due to their high-performance and replication resource overheads, and implementation complexity [128].

The "CFT vs. BFT" conundrum creates a fundamental design trade-off between the *efficiency of CFT protocols* for practical deployments and the *robustness of BFT protocols* for Byzantine settings of modern cloud environments. However, traditional BFT protocols design and evaluation has not taken into account *modern cloud hardware*.

Our work seeks to resolve this trade-off transforming existing CFT protocols for the Byzantine (untrusted) cloud environments with modern cloud hardware. Our approach underpins the robustness and efficiency axes. For *robustness*, we leverage trusted hardware available as part of confidential cloud computing to harden the security properties of CFT protocols [17, 30, 87, 103]. We use trusted execution environments (TEEs) to provide two key properties [49] (§ 4.2) for successfully transforming a CFT protocol for Byzantine settings: (a) transferable authentication, i.e., the ability to establish trust among distributed nodes by designing a remote attestation protocol, and (b) non-equivocation, i.e., once a node is trusted through remote attestation, the node follows the CFT replication protocol faithfully, thus, it cannot send conflicting statements to other nodes. For *efficiency*, we leverage modern networking hardware, such as RDMA/DPDK for kernel bypass, to design a highly optimized communication protocol that replicates state across nodes in distributed settings [16, 90, 110], overcoming the I/O bottlenecks in trusted computing [145].

Our proposal: RECIPE. RECIPE leverages TEEs along with direct I/O, two prominent cloud technologies, to resolve the tension between security and performance by building an efficient and practical transformation of unmodified CFT replication protocols for Byzantine settings. RECIPE achieve this by implementing a distributed trusted computing base (TCB) that shields the replication protocol execution and *extends* the security properties offered by a single TEE (whose security properties are only effective in a single-node setup) to a distributed setting of TEEs. Our design is comprised of a transferable authentication phase (§ 5.3) for distributed trust establishment, a high-performant network stack for secure communication over the untrusted network (§ 6) and a memory-efficient KV store (§ 6). We materialise RECIPE approach as a generic library, RECIPE-lib (§ 6).

Our evaluation assesses RECIPE's generality and efficiency. To show the generality, we apply and evaluate RECIPE on real hardware with four well-known CFT protocols (we use an 'R-' prefix for

*This work was done while the authors were at the University of Edinburgh, UK.

MIDDLEWARE '25, December 15–19, 2025, Nashville, TN, USA

ACM ISBN 979-8-4007-1554-9/2025/12.
https://doi.org/10.1145/3721462.3770763

	Leader-based	Leader-less
Total order	**Raft** [121], ZAB [131], Multi-Paxos [147]	**AllConcur** [126], Derecho [88]
Per-key order	**CR** [132], CRAQ [144], PB [120], CHT [44]	**ABD** [107], CP [101], Hermes [93]

Table 1: CFT protocols taxonomy. Using Recipe, we transform one protocol (shown in bold) of each category.

our transformed protocols); a decentralized (leaderless) linearizable multi-writer multi-reader protocol (ABD) [107] (R-ABD), two leader-based protocols with linearizable reads, Raft [121] (R-Raft) and Chain Replication (CR) [132] (R-CR), and AllConcur [126] (R-AllConcur), a decentralized consensus protocol with consistent local reads. To evaluate performance, we compare Recipe protocols with two competitive BFT replication protocols, BFT-smart [140] (PBFT [40]), whose specific implementation has been adopted in industry [13] and Damysus [56] a state-of-the-art BFT replication protocol. Our evaluation shows that Recipe achieves up to 24× and 5.9× better throughput w.r.t. PBFT and Damysus, respectively, while improving scalability—Recipe requires $2f+1$ replicas, f fewer compared to PBFT ($3f+1$). We further show that Recipe can offer confidentiality—a security property not provided by traditional BFT protocols—while achieving a speedup of 7×—13× w.r.t. PBFT and up to 4.9× w.r.t. Damysus.

To summarize, we make the following contributions:

- **Hardware-assisted transformation of CFT protocols:** We present Recipe, a generic approach for transforming CFT protocols to tolerate Byzantine failures without any modifications to the core of the protocols.
- **Formal analysis and verification:** We formally verify safety and security properties of Recipe using Tamarin [109]. By modeling Recipe while assuming a Dolev-Yao attacker [59], we verify key properties like safety, integrity, and freshness through automated deduction and equational reasoning. Therefore, we provide a correctness analysis for the safety and liveness properties of our transformation of CFT protocols operating in Byzantine settings.
- **Generic Recipe APIs:** We propose generic Recipe APIs to transform existing CFT protocols for Byzantine settings. Using Recipe APIs, we successfully transform a range of leader-/leaderless-based CFT protocols enforcing different ordering semantics.
- **Recipe in action:** We present an extensive evaluation of Recipe by applying it to four CFT protocols: Chain Replication, Raft, ABD, and AllConcur. We evaluate these four protocols against the state-of-the-art BFT protocol implementations and show that Recipe achieves up to 24× and 5.9× better throughput.

2 Background

2.1 The CFT Vs. BFT Conundrum

CFT protocols. CFT protocols assume a trusted infrastructure, tolerating only benign faults; replicas can fail by stopping or by omitting some steps [58]. As such, while having low overheads, they are not suitable for modern applications deployed in the third-party untrusted cloud infrastructure [31]. In this paper, we evaluate protocols that enforce either sequential consistency [100] or linearizability [79], also referred to as *strongly-consistent* replication protocols.

We can broadly split strongly-consistent CFT protocols into two categories (see Table 1 for the taxonomy): *(i)* leader-based protocols (e.g., Raft [121], Chain Replication (CR) [132]), where a node, designated as a leader, drives the protocol execution and *(ii)* decentralized protocols (e.g., ABD [107], AllConcur [126]), where there is no leader and all nodes can propose and execute requests.

We further divide them based on their ordering semantics. First, protocols with total ordering, where the protocols create a total order of all writes across all keys and apply them in that order. Second, protocols with per-key ordering semantics where the protocol enforces the total order of writes on a per-key basis. The evaluation of Recipe (§ 8) relies on this taxonomy to systematically study its protocols' performance, as these two dimensions significantly impact the performance of the CFT protocols [67].

BFT protocols. In contrast to CFT protocols, BFT protocols assume very little about the nodes and the network; faulty nodes may behave arbitrarily while the network is unreliable. To tolerate f arbitrarily faulty processes that may *equivocate* (i.e., make conflicting statements for the same request to different replicas), BFT protocols add f extra replicas to their system model requiring at least $3f+1$ replicas for safety. As such, BFT protocols exhibit worse scalability compared to CFT protocols (which only require at most $2f+1$ replicas).

BFT protocols are limited in performance, too. They incur high message complexity ($O(f^2)$) [41, 92, 149], multiple protocol rounds [27, 41, 92, 106, 153] and complex recovery ($O(f^2)$ in view-change) [41, 92, 106, 149]. As an example of this, PBFT [41], a well-known BFT protocol, requires at least $3f+1$ nodes, executes three broadcast rounds, and incurs $O(n^2)$ message complexity.

Thirdly, BFT protocols are complex, introducing burdens to developers. Guerraoui et al. [33] found that most protocol implementations consist of thousands of lines of (non-trivial) code, e.g., PBFT [41] and Zyzzyva [95]. Even if system designers wish to use a state-of-the-art BFT protocol, optimizing it for the specific application settings (e.g., network bandwidth, number of clients and replicas, cryptographic libraries, etc.) is a rather complicated task.

2.2 Modern Hardware in the Context of BFT

Confidential Computing. In the long line of confidential computing, trusted execution environments (TEEs) [3, 17, 30, 52, 133] offer the state-of-the-art confidential computing environment that guarantees the integrity and confidentiality of its code and data, remaining resistant against all software attacks even in the presence of a privileged attacker (hypervisor or OS). Based on this promise, TEEs are now being streamlined by all major CPU manufacturers, i.e., Intel TDX/SGX [17, 87], RISC-V Keystone [133], AMD-SEV [30], Arm Realms [3]. etc. and are adopted in cloud [6, 7, 50, 73, 113].

In our work, we build Recipe leveraging these recent advances in hardware in the context of BFT. We realize the potential of TEEs in hardening the properties of CFT replication protocols in the presence of Byzantine actors (e.g., network adversaries, compromised OS/hypervisor, corrupted host memory, etc.) in the untrusted cloud.

Direct network I/O. High-performance distributed systems [34, 61, 83, 89, 91, 115, 116] abandon the traditional kernel-based networking (sockets) to avoid syscalls' overheads [139]. Instead, they adopt direct network I/O (RDMA [110], DPDK [16]) to map the device's address space into userspace, bypassing the kernel stack.

We also adopt direct network I/O as it is even more well-suited to TEEs where syscall execution is extremely expensive [35, 69]. We leverage eRPC [90], a general-purpose and asynchronous remote

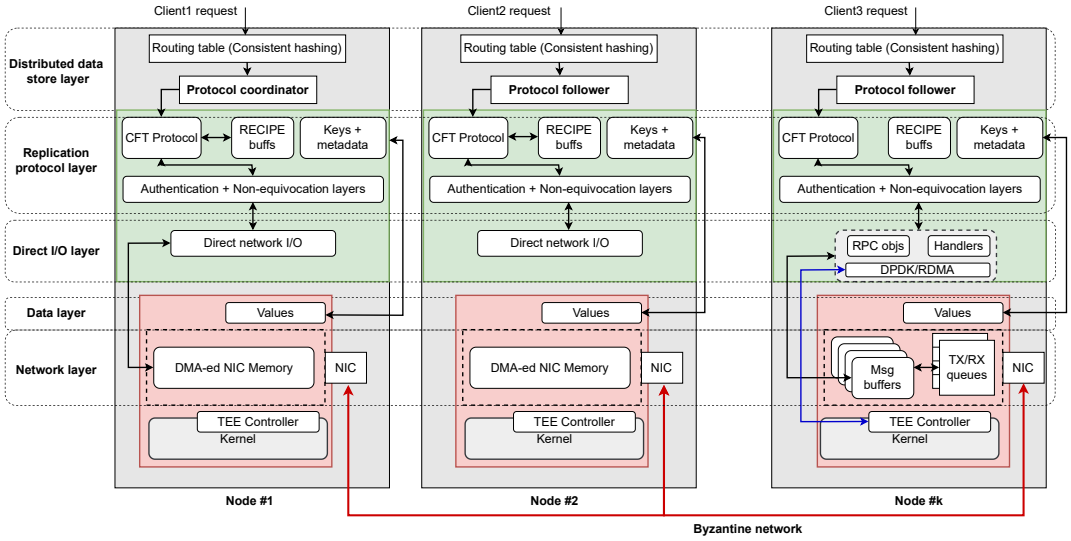

Figure 1: RECIPE's system architecture.

procedure call (RPC) library for high-speed networking for lossy Ethernet or lossless fabrics. eRPC provides us with a UDP stack and RDMA, DPDK and RoCE transport layers. We extended eRPC to *(1)* optimize the TEE memory usage and *(2)* shield the network messages against the untrusted network infrastructure.

3 System Model

Model sketch. We model the distributed system as a set of N TEEs in N nodes (or replicas) that host either *follower* or *coordinator* processes which execute a CFT protocol and communicate by exchanging messages. We assume that RECIPE's nodes run in a third-party untrusted cloud infrastructure. A coordinator serves client requests by initiating the implemented CFT replication protocol. Upon completion, it replies back to clients. In leaderless protocols, coordinators are selected randomly (any node can act as a follower and/or a coordinator). In leader-based protocols, only the active leader can act as a coordinator, the rest of the nodes are followers.

Communication model. Nodes communicate via a fully-connected, bidirectional, point-to-point and unreliable message-passing network, where messages can be arbitrarily delayed, re-ordered or dropped. In line with previous BFT protocols, we adopt the partial synchrony model [63], where there is a known bound Δ and an unknown Global Stabilization Time (GST), such that after GST, all communications arrive within time Δ.

Fault and threat model. We say that a node is *faulty* if it exhibits Byzantine behavior [102]. The unprotected (*out-of-the-TEE*) infrastructure (e.g., host memory, OS, NIC, network infrastructure/adversaries) can exhibit Byzantine behavior while we assume that the TEEs can only crash-fail. We say that a node is faulty if one of the following holds true: *(i)* the TEE fails by crashing or *(ii)* the unprotected infrastructure is Byzantine. For safety (and liveness), we assume that for $N \geq (2f+1)$ nodes up to f can be *faulty*.

4 RECIPE Overview

4.1 Architecture Overview

Distributed systems architecture. Figure 1 shows the overview of a distributed data store system that builds on top of the RECIPE system. Distributed data stores implement a tiered architecture consisting of a *distributed data store layer*, *replication layer*, and *data layer*. In our case, the replication and data layers are provided by RECIPE. The distributed data store layer maintains a routing table that matches the keyspace with the owners' nodes. This layer is responsible for forwarding client requests to the appropriate coordinator nodes (e.g., leader of the replication protocol) for execution. The RECIPE replication layer is responsible for consistently replicating the data by executing the implemented protocol. After the protocol execution, RECIPE nodes store the data in their Key-Value stores (KVs), the data layer, and they reply to the client [66, 104, 130, 134].

RECIPE architecture. RECIPE design is based on a distributed setting of TEEs that implement a (distributed) trusted computing base (TCB) and shield the execution of unmodified CFT protocols against Byzantine failures. RECIPE's TCB contains the CFT protocol's code along with some metadata specific to the protocol. The code and TEEs of all replicas are attested before instantiating the protocol to ensure that the TEE hardware and the residing code are genuine. All authenticated replicas receive secrets (e.g., signing or encryption keys) and configuration data securely at initialization.

Further, RECIPE builds a *direct I/O layer* comprised of a networking library for low-latency communication between nodes (§ 6). The library bypasses the kernel stack for performance and shields the communication to guarantee non-equivocation and transferable authentication against Byzantine actors in the network (§ 4.3 and § 6). RECIPE guarantees both properties by layering the non-equivocation and authentication layers on top of the direct I/O layer. In addition, to strengthen RECIPE's security properties and eliminate syscalls, we map the network library software stack to the TEE's address space.

Lastly, RECIPE builds the *data layer* on top of local KV store instances. Our design of the KV store increases the trust to individual nodes, allowing for local reads (§ 6). Our KV store achieves two goals; first, we guarantee trust to individual replicas to serve reads locally, and secondly, we limit the TCB size, optimizing the TEE memory (enclave) usage. As shown in Figure 1, RECIPE keeps bulk data (values) in the host memory and stores only minimal data (keys + metadata)

in the TEE area. The metadata, e.g., hash of the value, timestamps, etc., are kept along with keys in the TEE for integrity verification.

RECIPE's adoption. Our decision to use TEEs with direct I/O for networking is grounded in practice. Confidential computing and kernel-bypass networking are widely adopted by cloud providers for both security and performance. Our research aligns with this trend; we trust the TEE and its vendor, and we believe that growing confidence in TEEs [98, 114] will further validate RECIPE's design.

4.2 Transformation Requirements

The basic requirements for transforming a CFT protocol for Byzantine environments are established in a theoretical result published by Clement et al. in PODC 2012 [49]. Due to this seminal paper, we know that non-equivocation and transferable authentication are necessary to go from $3f+1$ to $2f+1$ replicas for a reliable broadcast in Byzantine settings. Our work shows that not only can this lower bound be achieved in practice, but we can do so while providing high performance by leveraging modern hardware in a cloud environment. Next we discuss how RECIPE satisfies these two fundamental requirements, while § 4.4 elaborates on how to design *practical and* efficient protocols that meet these requirements.

Transferable authentication: The transferable authentication property refers to the authenticity of a received message, requiring that a replica must be able to verify that the supposed sender indeed had sent the message. The authentication is transferable if the original sender can be verified, even when the authentication proof is forwarded. Formally, a message m with authentication proof σ_i from P_i, which is verified by a correct process P_j, is also verified by any other correct process P_k. From which follows for correct processes:

$$\forall P_j,P_k:\ \text{Verify}_{P_j}(m,\sigma_i,P_i) \Rightarrow \text{Accept}_{P_k}(m,P_i)$$

Non-equivocation: The non-equivocation property guarantees that there are no conflicts between the accepted messages of correct processes. That implies that RECIPE must detect attacks in which adversaries try to send different, potentially stale messages to some processes. Formally, if k is a unique identifier, like a message counter or request id, then a Byzantine node P_i cannot get two messages $m \neq m'$ with the same k accepted by different correct processes P_j and P_k:

$$\forall P_j,P_k:\ \text{Accept}_{P_j}(k,m,P_i) \wedge \text{Accept}_{P_k}(k,m',P_i) \Longrightarrow m = m'$$

4.3 RECIPE Primitives In a Nutshell

RECIPE embodies the TEE-assisted primitives, i.e., the RECIPE authentication primitive and the RECIPE attestation primitive, shown in Algorithms 1 and 2, to ensure non-equivocation and transferable authentication. These primitives are integrated into the network library (§ 5.1) and complement the attestation protocol (§ 5.3) to guarantee in practice that only trusted replicas access the cryptographic keys and can generate and verify authenticated messages. We present an analysis and formal verification of RECIPE in § 7.

Authentication primitive. For the authentication, we use cryptographic primitives (e.g., MAC and encryption functions when RECIPE aims for confidentiality) to verify the integrity and the authenticity of the messages. Each message m sent from a node n_i to a node n_j over a communication channel cq is accompanied by metadata for preventing (or detecting) equivocation. This metadata includes a trusted counter value cnt_{cq}, the current view number $view \in \mathbb{N}$, sender and receiver nodes' ids, and the calculated message authentication code (MAC) $h_{\sigma_{cq}}$. The MAC is calculated over the payload and the metadata, then follows the message m. Formally, $m \leftarrow (\langle req,(view,cq,cnt_{cq}),h_{\sigma_{cq}}\rangle,(req||view||cq||cnt_{cq}))$. The sender node calls into the shield_request(req, cq) and generates such a trusted message for the request req. On the receiver side,

$$Accept(\langle req,\ (view,cq,cnt_{cq})\rangle,\ ID_j) \Longleftrightarrow \texttt{verify\_request}(h_{\sigma_{cq}},\ req,\ (view,cq,cnt_{cq})).$$

Algorithm 1: RECIPE's authentication primitives.

```
1  ▷ cnt_cq: the latest sent message id from cq
2  ▷ rcnt_cq: the last committed message id from cq
3  function shield_request(req, cq) {
4      cnt_cq ← cnt_cq+1; t← (view, cq, cnt_cq);
5      [h_σcq, (req,t)] ← singed_hash(req, t);
6      return [h_σcq, (req,t)];
7  }
8  function verify_request(h_σcq, req, (view, cq, cnt_cq)) {
9      if verify_signature(h_σcq, req, (view, cq, cnt_cq)) == True then
10         if view == current_view then
11             if cnt_cq <= rcnt_cq then
12                 return [False, req, (view, cq, cnt_cq)];
13             if cnt_cq == rcnt_cq+1 then rcnt_cq
                 ← rcnt_cq+1; buffer_locally(req, (view, cq, cnt_cq));
14             return [True, req, (view, cq, cnt_cq)];
15     return [False, req, (view, cq, cnt_cq)];
16 }
```

Non-equivocation. RECIPE prevents equivocation and replay attacks in the network using sequence numbers for all the messages. Replicas maintain local sequence tuples $t = (view,cq,cnt_{cq})$ where cnt_{cq} is the current trusted counter value in that view for the latest request sent over cq. The sender assigns a unique tuple to messages and then increments the trusted counter to guarantee monotonicity. Formally, if $m_i \prec m_j$ is the relation defining the message order, then $\forall m_i,m_j:\ m_i \prec m_j \Longleftrightarrow cnt_{cq}(m_i) < cnt_{cq}(m_j)$. Replicas can verify the freshness of a message by examining its cnt_{cq} (verify_request() primitive). The primitive verifies that the message's id (as part of the metadata) is consistent with the receiver's local counter rcnt_{cq} (rcnt_{cq} is the last seen valid message counter for received messages in cq).

Attestation primitive. Remote attestation is the building block to verify the authenticity of a TEE, i.e., the code and the TEE state are the expected [125]. As such, RECIPE provides attest(), generate_quote() and remote_attestation() primitives (Algorithm 2) that allow replicas to prove their trustworthiness to other replicas or clients. The attestation takes place before the control passes to the protocol's code. Only successfully attested nodes get access to secrets (e.g., signing or encryption keys, etc.) and configurations.

4.4 System Design Challenges

Our work shows how to leverage modern hardware to build efficient, robust, and easily adaptable distributed protocols by meeting the aforementioned transformation requirements. Specifically, we address the following research questions.

Q1: How to use TEEs efficiently? TEEs are not a panacea: due to their architectural limitations (limited trusted memory and slow syscalls' API) [69, 94], their naive adoption to build practical systems does not necessarily translate into performance improvements. For example, communication in the state-of-the-art BFT protocols [56, 112, 149], which is at the core of any distributed protocol, primarily

Algorithm 2: RECIPE's attestation primitive.

```
1  function remote_attestation() {
2      nonce ← generate_nonce();
3      send(nonce, k_pub); DHKE(); quote_{σ_{k_pub}} ← recv();
4      if verify_signature(quote_{σ_{k_pub}}) == True then
5          μ_TEE ← decrypt(quote_{σ_{k_pub}}, k_priv);
6          if (verify_quote(μ_TEE) == True) send_secrets();
7  }
8  function attest() {
9      μ ← gen_enclave_report(); return μ;
10 }
11 function generate_quote(μ, k_pub) {
12     key_hw ← EGETKEY();
13     quote ← sign(μ, key_hw); quote_{σ_{k_pub}} ← sign(quote, k_pub);
14     return quote_{σ_{k_pub}};
15 }
```

builds on standardised kernel interfaces (e.g., sockets) suffering from big latencies and not exploiting the full potential of the system [152].

In RECIPE, we carefully address these limitations without introducing an additional burden to developers. We build a secure Remote Procedure Call (RPC) framework on top of a direct I/O network stack for TEEs that: (1) boosts performance by avoiding expensive syscalls within TEEs. (2) extends the transferable authentication and non-equivocation primitives across the untrusted network infrastructure realising the transformation in practice. (3) follows an established RPC-programming paradigm that has proven to be the most effective one for building distributed protocols [61, 90, 91, 93].

Q2: How to use TEEs to transform and build *practical* systems? While Clement et al. show that a translation of a CFT protocol to a BFT protocol *exists*, this mechanism adopts an impractical strategy when it comes to building real systems. The entire (transformed) system relies on an expensive mechanism to ensure the correct execution of the underlying CFT protocol. In each round, each replica needs to receive the history of all previous messages, verify their authentication and replay the execution of the protocol's entire history. This way, it is ensured that non-Byzantine replicas rebuild their state correctly and also that the currently executed message is legitimate (i.e., derives from a valid execution scenario of the protocol).

Secondly, the transformed protocol may amplify the native semantics of the original CFT protocol such as linearizable local reads. As in classical BFT protocols, clients cannot trust individuals, instead, they build collective trust by receiving $f+1$ identical replies from different replicas to ensure that at least one correct replica has responded.

We design RECIPE to work out-of-the-box to build real systems. RECIPE leverages the properties offered by TEEs to shield the correct protocol execution while our network stack extends the security properties to the network. As a result, our approach does not impose any dependency on the history execution of the protocol, and the original protocol's message complexity is not affected. We also offer an authenticated, per-node, in-memory KV store to allow replicas to detect integrity and authenticity violations and to support local reads individually. Our approach improves performance, but enables easy adoption as well; developers do not have to worry about maintaining protocols' semantics in Byzantine settings.

Q3: How to realize initialisation, recovery, and failure detection? While the transformation remains agnostic to the transformed CFT protocol in normal operation, the system designers still need to design recovery mechanisms when failures occur. Clement et al. do not address how the system *initialises* its state, *detects* failures, and *recovers* from them. CFT protocols vary in their mechanisms for recovery and failure detection. Some protocols continue to operate when failures occur [107, 120] while others rely on accurate timeouts to detect non-responsive leaders and nodes [93, 121, 132]. Unfortunately, TEEs come with neither a trusted initialisation mechanism for distributed systems [15] nor a trusted timer source [22, 24].

RECIPE builds on a secure substrate that overcomes these limitations. We build on a mechanism for collective attestation and a trusted lease mechanism [146] which is a foundational primitive for *trusted timeouts*, failure detectors [84], leader election [65], etc.

Q4: Is BFT enough? The case for confidential BFT protocols. Applications that manage sensitive data (e.g., health-care applications [96]) adopt blockchain solutions for privacy. To this end, cloud-hosted blockchain solutions have been launched [2, 4, 14, 20, 25]. However, these cloud-hosted blockchain systems that fundamentally build on agreement protocols for serialising the ledger [21], jeopardise the blockchain principles of decentralised trust [112].

While BFT protocols offer an important foundation to build trustworthy systems, we argue that modern applications [1, 37, 45, 68, 119, 123, 127] seek confidentiality beyond the scope of the BFT model. RECIPE satisfies this need. Built on top of TEEs, RECIPE transparently offers confidential execution without sacrificing performance.

5 RECIPE Protocol

Clients in RECIPE execute requests through a `PUT`/`GET` API. As discussed in § 4.1, the request is forwarded to the protocol's coordinator node. Figure 2 shows as an example a RECIPE implementation of Raft (R-Raft) including all three execution phases of a typical RECIPE protocol: the transferable authentication phase (blue box), the initialization phase (green box) and the normal execution phase where the transformed CFT protocol executes clients' requests (red box). Prior to the protocol execution, nodes pass through a transferable authentication phase (§ 5.3) to prove that the TEEs and loaded code are genuine, followed by initialization and normal operation.

5.1 Normal Operation

We first explain the initialization and the normal execution phases, assuming all participant nodes executed the transferable authentication phase successfully. The nodes execute the initialization phase, initializing their own local KVs (B.1) and their network connections (e.g., configures NIC-memory, network ports, etc.) and establish connections with other peers (B.2) based on the configuration it securely received at the attestation process (A.7). The leader then runs the underlying CFT protocol (in our case, Raft (C.1)—(C.9)) to execute the client request (R.1). Upon completion, it replies back to the client (R.2). Next, we discuss the RECIPE abstraction under the normal operation.

We use the notation $[h_{c_{\sigma_c}}, \text{payload}]$ to denote an *attested* or *shielded* message that is comprised of the signed hash ($h_{c_{\sigma_c}}$) of payload (*certificate*) along with the raw payload data. We use the symbol σ_c to denote that a piece of data is signed with a key c. Figure 2 uses the notation (α, kv) for an attested message referring to a key-value pair kv with certificate α.

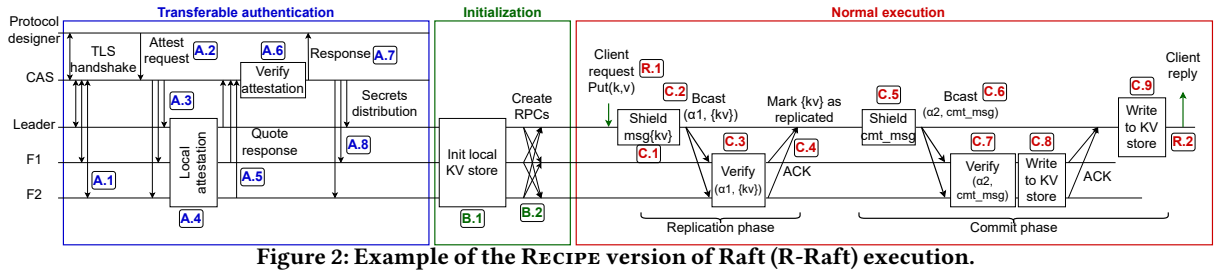

Figure 2: Example of the RECIPE version of Raft (R-Raft) execution.

#1: Clients send to the coordinator their request of the form [$h_{c_{\sigma_c}}$, (metadata, req_data)] (R.1). The req_data is the request's associated data and the metadata might include among others the client's and the request's id, the leader's and term's ids (known to the client).

#2: Nodes receive and process a request after successfully verifying their integrity and authenticity. RECIPE's protocols inherit the constrains of the original CFT protocol. For instance, our R-Raft leader will drop requests with wrong view of the term or leader.

#3: Upon the reception of a client's request:

#3.1 The coordinator (leader) verifies the integrity and authenticity of the message using RECIPE's authentication layer. It also verifies the metadata, e.g., the message is invalid if the term and the leader (if any) known to the client are incorrect. The leader updates the client table with the latest processed request for each client.

#3.2 Next, the leader initializes the protocol for that request. In our example, the Raft leader shields the message (C.1), generating a trusted message format (α1, kv) where α1 is the certificate of kv and broadcasts the request to the followers (C.2) (replication phase).

#3.3 The messages exchanged between replicas are of the form [$h_{r_{\sigma_{cq}}}$, (metadata, req_data)]. The metadata includes a per-request unique tuple (view, cq, cnt_{cq}) that contains: (1) the view, an identifier that is optionally set by the implementation for every new leader (if any) (2) the communication channel id (cq) and (3) a sequencer id or a message counter (cnt_{cq}) that is assigned to the messages sent over this channel and is increased monotonically for every new message.

#4: When a replica receives a message:

#4.1 If the replica is in normal state operation, it verifies the validity of the message. Else, it refuses to process the request.

#4.2 The replica verifies the received sequencer id (recv_{cnt}) to see if it is consistent with its local counter (cnt_{cq}). If the recv_{cnt} equals (cnt_{cq}+1), the replica executes the request immediately, increases its local counter, acknowledges to the sender and updates the client table. If the recv_{cnt} refers to a "future" message (recv_{cnt}>cnt_{cq}+1), the replica queues the request in the protected area. Periodically, it applies the queued requests eligible for execution and notifies coordinators accordingly.

#5: In our example, the followers verify the request (C.3), enqueue the un-committed request in a TEE buffer, and send ACKs back to the leader. The leader, upon receiving the majority of ACKs marks the request as replicated (C.4) and proceeds to the second round of the protocol instructing the followers that replied to commit the update ((C.5)—(C.7)). At this point, each follower instructed to commit applies the request to its local KV store(C.8) and ACKs the commit to the leader. Similarly to the replication phase, the leader finally commits (C.9) when it receives ACKs from the majority.

#6: After the protocol's execution, the coordinator marks the request as committed and notifies the client (R.2).

5.2 View Change

While decentralized protocols remain available as long as most nodes are part of the membership, the leader-based protocols do not progress if the leader goes down. To remain available after the leader crashes, the followers need to closely monitor the leader (e.g., heartbeat messages in inactive periods) and, in case it is unreachable, to *elect* a new one, i.e., perform a *view change*.

In line with the CFT protocols, RECIPE protocols must assign a leader with a term and a node identifier. The term id can be seen as an *epoch*, a monotonically increasing counter that uniquely identifies the current view of the system. To continue serving requests after a leader election, the majority needs to confirm the new leader along with the new term. Since a leader needs to be acknowledged by the majority of the nodes to operate, the latest term will survive in at least one node, ensuring the term's monotonic increments.

Correctness. The correctness condition for leader elections is that every commit must survive into the new leader election in the order selected for it at the time it was executed. RECIPE does not make further assumptions, protocols can rely on their election algorithms as we guarantee that the replicas are trusted.

Failure detection. CFT protocols [121, 132] often require trusted timers to detect failures. RECIPE builds on top of Intel SGX, which does not secure timers [22, 24] whereas OS-timers and software clocks cannot be trusted. To overcome this, RECIPE implements a trusted lease mechanism [146]. Our mechanism supports all the properties of classical leases [74] that are the building block for trusted timeouts, failure detectors [84], leader election [65], etc.

5.3 Transferable Authentication

Before initialization, all participant nodes run the transferable authentication phase (are *attested*). The phase ensures that only authenticated replicas receive configurations and secrets and participate to the protocol, guaranteeing the transferable authentication property and protecting against Sybil attacks [60]. This step is essential to prevent impersonation attacks, even with correct and protected code, as basic authentication cannot stop Byzantine behaviors like forking.

RECIPE materializes this phase using a remote attestation protocol. Right after the attestation protocol we rely on node identifiers and message sequencers (§ 5.1, § 6).

The attestation protocol is initialized by the protocol designer (PD) (*challenger*) who establishes TLS connection with the Configuration and Attestation service (CAS)(A.1). CAS is responsible for proving the authenticity of a TEE. For now, we focus solely on the attestation protocol; the CAS is discussed in § 6. The CAS also establishes secure communication channels with the participant nodes.

The PD sends an *attest request* to the CAS(A.2), which is then forwarded to the replicas (A.3). The replicas perform *local attestation* [125],

i.e., they calculate a measurement of their code and generate a quote that is uniquely bound to that particular TEE (A.4). The quotes are sent over the TLS channel to the CAS for verification. Upon a successful attestation of a TEE, the CAS notifies the PD to forward configurations to the replicas (A.7)—(A.8).

CAS runs on attested TEEs, hence, it does not pose vulnerability risks. As a single point of trust, it may face availability issues—e.g., an attacker shutting it down prohibits RECIPE nodes to be attested and join the membership (§ 5.5).

5.4 Recovery

As nodes fail, new or recovered nodes need to be added to continue operating at peak performance. To add a new node, the membership needs to be reliably updated, following that all other live replicas are notified of the new node's intention to join the replica group. For non-equivocation, recovered nodes always start as fresh nodes and as such are assigned unique node ids by the CAS through the attestation phase. Overall, a new joining node follows the next steps:
#1: A recovering node needs first to be attested before any secrets and membership information are shared. Before the control passes to the CFT protocol, the node sends a join request to a designated node, notifying it about its willingness to join the cluster.
#2: The challenger-node that receives the request initializes a remote attestation to verify the new node's trustworthiness (§ 5.3).
#3: After a successful attestation, as a response to the join-request, the challenger-node shares the network signing or encryption keys and the configurations. The challenger-node also broadcasts a message to the other replicas about the successful attestation of the new node. Once the new joiner acknowledges the response from the challenger-node, it establishes connections with the other replicas.
#4: The new node joins as a shadow replica fetching the state of the system as in [62, 122]. If the CFT protocol allows, this node can participate in writes while recovering. Once it is synchronized with the system's state, it transitions to normal operation of the protocol.

5.5 RECIPE liveness and safety properties

We do not consider DoS-based liveness or censorship attacks, since the unprotected cloud infrastructure can be controlled by an adversary to compromise the availability of RECIPE. While not implemented, client-centric techniques as in [151] can also be adapted.

RECIPE is safe. Especially, in the context of rollback attacks, where other systems, e.g., MinBFT [149], recover from persistent storage, RECIPE attestation phase carefully addresses this violation relying on unique fresh node identifiers and message sequence numbers using lightweight symmetric-key hashing, with no measurable performance overhead to generate distinguishable protocol messages.

6 RECIPE Library

This section describes four core components of RECIPE. Table 2 summarizes the RECIPE's API for each component.

RECIPE networking. RECIPE adopts the Remote Procedure Call (RPC) paradigm [34] over a generic network library with various transportation layers (Infiniband, RoCE, and DPDK), which is also favorable in the context of TEEs where traditional kernel-based networking is impractical [97].

We offer asynchronous network operations following the RPC paradigm. For each RPCobj, RECIPE keeps a transmission (TX) and reception (RX) queue, organized as ring buffers. Developers enqueue requests and responses to requests via RECIPE's specific functions which place the message in the RPCobj's TX queue. Later, they can call a polling function that flushes the messages in the TX and drains the RX queues of an RPCobj. The function will trigger the sending of all queued messages and process all received requests and responses. Reception of a request triggers the execution of the request handler for that specific type. Reception of a response to a request triggers a cleanup function that releases all resources allocated for the request, e.g., message buffers and rate limiters (for congestion).

Attestation API	
`attest(measurement)`	Attests the node based on a measurement.
Initialization API	
`create_rpc(app_ctx)`	Initializes an RPCobj.
`init_store()`	Initializes the KV store.
`reg_hdlr(&func)`	Registers request handlers.
Network API	
`send(&msg_buf)`	Prepares a req for transmission.
`respond(&msg_buf)`	Prepares a resp for transmission.
`poll()`	Polls for incoming messages.
Security API	
`verify_msg(&msg_buf)`	Verifies the authenticity/integrity and cnt of a msg.
`shield_msg(&msg_buf)`	Generates a shielded msg.
KV Store API	
`write(key, value)`	Writes a KV to the store.
`get(key, &v`$_{TEE}$`)`	Reads the value into v$_{TEE}$ and verifies integrity.

Table 2: RECIPE library APIs.

While RECIPE's kernel-bypassing architecture targets performance, its network stack embeds non-equivocation and transferable authentication primitives (see § 4.3), enabling BFT transformation of CFT protocols with just $2f+1$ nodes.

Non-equivocation and authentication layers: RECIPE's `send()` function calls into the shield_request() primitive (§ 4.3) ensuring that sender always assigns to unique and monotonic, thus, distinguishable, sequence numbers to the transmitted messages. Replicas poll for messages (`poll()`) executing the implemented CFT protocol when they receive verified valid messages (verify_request() primitive). In addition, RECIPE's replicas are willing to accept "future" valid messages as these might come out of order, i.e., messages whose cnt$_{cq}$ is > (rcnt$_{cq}$+1). These messages are processed and committed according to the CFT protocol. The shield_request() and verify_request() are also verifying the authenticity and the integrity of the messages.

Secure runtime. We build our codebase in C++ using SCONE to access the TEE hardware. SCONE exposes a modified libc library and combines user-level threading and asynchronous syscalls [139] to reduce the cost of syscall execution. While we limit the number of syscalls, leveraging SCONE's exit-less approach allows us to optimize the initialization phase that vastly allocates host memory for the network stack and the KV store. To enable NIC's DMA operations and memory mappings to the hugepages (for message buffers and TX/RX queues) (§ 6), we overwrite the `mmap()` syscall of SCONE to bypass its shield layer and allow the allocation of (untrusted) host memory. RECIPE runtime supports a trusted lease mechanism [146] that preserves the correctness guarantees of classical leases [75], even when timers may be unreliable.

RECIPE key-value store. RECIPE provides a lock-free, high-performant KV store based on a skip-list. We partition the keys from the values' space by placing the keys along with metadata (and a pointer to the

value in host memory) inside the TEE's memory area, the *enclave*, and storing the values in the host memory. Recipe's KV store design resolves Byzantine errors since the metadata (and the code that accesses them) reside in the enclave. That said, Recipe allows for local reads as nodes can verify the integrity of the stored values.

Attestation and secrets distribution. The attestation process is initialized by the *challenger*, a remote process that can verify the authenticity of a specific TEE. The challenger executes the remote_attestation() primitive (§ 4.3) to send an attestation request to the application—usually in the form a nonce (a random number). The challenger and the application, then, pass through a Diffie-Hellman key exchange process [111]. The application generates an ephemeral public key used by the challenger later to provision any secrets.

When the TEE receives the nonce, it calls the attest() and generates a *measurement* (μ) of its state and loaded code. Following this, the TEE calls into the generate_quote(μ,k_{pub}) to sign μ (quote) with the key_{hw} which is fetched from the TEE's h/w. The TEE signs and encrypts the quote $\text{quote}_{\sigma_{k_{pub}}}$ over the challenger's public key k_{pub} which is, then, sent back to the challenger. Upon successful verifications of the $\text{quote}_{\sigma_{k_{pub}}}$, the challenger shares secrets and configurations.

7 Recipe Analysis and Formal Verification

7.1 Requirements Analysis

We show how Recipe satisfies the required properties (§ 4.2).

Non-equivocation. Recipe prevents equivocation attacks through trusted monotonically increasing message counters cnt_{cq} that assign sequence numbers to the network messages. The sender assigns a monotonically increasing sequence number to every message of a given round, $cnt_{cq} \leftarrow cnt_{cq} + 1$, guaranteeing a total ordering of all network messages between any two communication endpoints. Formally, for any two messages m_1,m_2 over a communication channel cq: $\forall m_1, m_2 : \; m_1 \prec m_2 \iff cnt_{cq}(m_1) < cnt_{cq}(m_2)$ thus $cnt_{cq}(m_1) = cnt_{cq}(m_2) \iff m_1 = m_2$. On the receiving side, it suffices for replicas to verify that the message's counter is *consistent* with their local known sequencer for this communication channel, $cnt_{cq} = rcnt_{cq} + 1$ or for accepting "future" valid messages as these might come out of order, i.e., $cnt_{cq} > (rcnt_{cq} + 1)$. Recipe's sequencer prevents the replays (stale but authenticated messages), which is indistinguishable from equivocation, $cnt_{cq} < rnt_{cq}$. In addition, a Byzantine node may "appear" to not send messages to some (weak non-equivocation) or all (strong non-equivocation) other nodes during a given operation [108]. Recipe is responsible for neither—we rely on the CFT as both weak and strong non-equivocation are indistinguishable from crash failures [108].

Transferable authentication. Recipe ensures the core two properties from its TEE-assisted primitives: (1) Recipe distributes the configuration, keys etc. in a secure manner to trusted nodes, and (2) Recipe preserves the authenticity and integrity of the network messages.

Transferable authentication is provided implicitly by properties (1) and (2). The first ensures that signing keys are shared for every communicating pair of processes after their successful attestation. Recall that configuration data, signing keys, and other secrets are securely provisioned only to successfully attested TEEs, i.e., trusted nodes that have successfully completed remote attestation. Thus, only trusted (correct) processes can sign (and generate) valid (and verifiable) messages, since every message m is signed by the sender's TEE using a private key sk. It also follows that Byzantine adversaries, even by controlling the unprotected infrastructure (see §3), cannot alter or forge messages without the signing key, including "future" messages; instead, they are only limited to replaying old messages. Formally, $Pr[Verify(\sigma_c,m,k_{pub}) = 1] \leq negl(\lambda)$, where σ is the signature, λ is the security parameter (e.g., key size), and $negl(\lambda)$ is a negligible function, meaning it decreases faster than any polynomial function. Lastly, authenticity is transferable and can be verified in the exact same way that any two directly communicating nodes do.

7.2 Correctness Analysis

CFT protocols need to provide the following safety properties regarding the messages delivered by the network [80, 81]. We show how these are provided by Recipe's non-equivocation and (transferable) authentication layers.

Safety. If a correct process p_i *receives and accepts* a message m from a process p_j, then the sender p_j is correct and has executed the `send` operation with m.

Integrity. If a correct process p_i *receives and accepts* a message m, then m is a valid and correct message generated according to the protocol specifications.

Freshness. If a correct process p_i *receives and accepts* a valid message m_{j_x} sent from a correct process p_j, then it will not accept any future message m_{j_y} with the same identifier, $y = x$, $\forall x,y \in \mathbb{N}^+$.

Next, we explain how Recipe satisfies these properties. Safety and integrity are directly satisfied by our transferable authentication mechanisms. Firstly, every message m is signed by the sender's TEE using a private key k_{priv}, and receivers verify m using the sender's public key k_{pub}. Thus, only trusted and correct processes can generate valid messages (i.e., valid signatures) that can be successfully verified: a message m accepted by some correct process p_i must have been generated and sent by a correct process p_j. Moreover, correct processes cannot deviate from the protocol's specification to generate messages that do not adhere to it. Byzantine adversaries can neither forge nor alter messages without k_{priv} (§ 7.1).

To guarantee freshness, our system uses monotonically increasing counters to establish a total order for messages between two endpoints. This allows a correct process p_i to discard duplicate messages, preventing replay attacks.

7.3 Formal Verification of the Recipe Protocol

We formally verify the safety and additional security properties of Recipe using Tamarin [109]. Tamarin operates in the symbolic (Dolev-Yao) model [59] and thus requires us to make the following assumptions: (1) we do not consider individual bits, but instead atomic terms, like a counter, cryptographic key, etc., which are composed to derive messages (2) all cryptographic functions are pure, i.e., they have no side effects, and perfect (e.g., no hash collision) (3) a potential attacker can read and delete all messages sent on the network and modify them using the functions built into Tamarin or explicitly provided in the model (e.g., no side channels). Based on these assumptions, we can model the system state as a multiset, and the possible state transitions as multiset rewriting rules, resulting in a labeled transition system. In the case of Recipe, this involved mapping of the transferable authentication, initialization, and execution phase to *facts* stored in the multiset and *rules* for each operation that modifies the system state. This allows us to consider an unbounded number of processes, messages, and protocol executions. Utilizing the results of

previous TLS verification work [23, 53, 54], we abstract the TLS handshake and subsequent connection as a secure channel in our model.

In order to verify the properties of this system, we annotate rules with parameterizable *action facts* and use them to express temporal first-order properties on all possible *traces*, i.e., transition sequences. Tamarin can then use deduction and equational reasoning to derive either a proof of correctness or a counterexample, which violates our property [109]. To express the temporal relation of action facts we will use $a@t_i$, to express that action fact a occurred in the trace at time point t_i. The relation $t_a \prec t_b$ specifies that t_a occurred strictly before t_b in the trace, and $t_a \equiv t_b$ expresses that both occur at the same time, which implies that they map to the same rule execution. Using this framework, we can express the following safety properties:

We use the action facts *Acc*(ept) and *Send*, which map to the according process operations, as well as the action fact *Tr*(usted), which marks the rule where the processes are finally attested and from which point onward they are trusted. The first property we verify, which corresponds to our safety and integrity properties, is that sent and accepted (by some process) messages always originate from a trusted (correct) process or formally in Tamarin:

$$\begin{aligned} &\forall p_i, m_{j_x}, t_{t_i}, t_a : \quad Tr(p_i)@t_{t_i} \wedge Acc(p_i, m_{j_x})@t_a \wedge t_{t_i} \prec t_a \\ \Rightarrow \quad &\exists p_j, t_{t_j}, t_s : \quad Tr(p_j)@t_{t_j} \wedge Send(p_j, m_{j_x})@t_s \wedge t_{t_j} \prec t_s \prec t_a \end{aligned}$$

We define property (2) to verify that messages are always accepted in the order they are sent. We express this in Tamarin as:

$$\begin{aligned} &\forall p_i, m_{j_x}, m_{j_y}, t_{t_i}, t_{a_x}, t_{a_y} : \\ &Tr(p_i)@t_{t_i} \wedge Acc(p_i, m_{j_x})@t_{a_x} \wedge Acc(p_i, m_{j_y})@t_{a_y} \wedge t_{t_i} \prec t_{a_x} \prec t_{a_y} \\ \Longrightarrow &\exists p_j, t_{s_x}, t_{s_y} : Send(p_j, m_{j_x})@t_{s_x} \wedge Send(p_j, m_{j_y})@t_{s_y} \wedge t_{s_x} \prec t_{s_y} \end{aligned}$$

Finally, we verify that messages are never accepted twice, which corresponds to our freshness property, expressed in Tamarin as:

$$\begin{aligned} \forall p_i, m_{j_x}, t_{t_i}, t_{a_x}, t_{a_y} : \quad &Tr(p_i)@t_{t_i} \wedge Acc(p_i, m_{j_x})@t_{a_x} \\ &\wedge Acc(p_i, m_{j_x})@t_{a_y} \wedge t_{t_i} \prec t_{a_x}, t_{a_y} \Rightarrow t_{a_x} \equiv t_{a_y} \end{aligned}$$

All of these properties are successfully verified by Tamarin for any number of TEEs, processes, and protocol executions, even in the presence of an attacker, as outlined above. A link to the full Tamarin model and proof artifact, which consists of more than seven thousand generated lines, will be provided after the double-blind review.

8 Evaluation

8.1 How to Apply the Recipe Library?

Recipe draws inspiration from systems like Fabric [31], which supports both CFT [21] and BFT [86] protocols for blocks ordering. Since replication protocols are central to fault-tolerant cloud services, we envision Recipe serving blockchain layers and generalizing to distributed KVs and replicated systems requiring BFT-like guarantees [35]. Below, we outline how developers can use the Recipe-lib API and transform widely adopted CFT protocols like Raft [8, 21, 51].

Developers can use the Recipe-lib API to transform their preferred CFT protocol for Byzantine settings without further modifying the core states of the protocol. Listing 1 illustrates Raft's transformation using the same example of R-Raft from Figure 2. In Listing 1, the blue sections show the native Raft code, whereas the orange sections show the modifications introduced by Recipe.

```
// ----------- Requests handlers definition -----------
void replication_hdlr(Raft_ctx* ctx, Msg* recv_msg) {
   // verifies recv_msg integrity and counter
   [msg, cnt] = verify_msg(recv_msg);
   ... // appends the req to the on-going reqs buffer
   conn.respond(shield_msg(ACK_repl)); // transmits ACK
}
void cmt_hdlr(Raft_ctx* ctx, Msg* recv_msg) {
   [msg, cnt] = verify_msg(recv_msg);
   auto [key, val] = decode(req);
   // stores val in host mem and its certificate in TEE
   ctx->kv.write(key, val);
   conn.respond(shield_msg(ACK_cmt)); // transmits ACK
}
// ----------- Init phase  -----------
auto ctx = new Raft_ctx(metadata, node_type);
// init local KV with host allocated memory and a cipher
ctx->kv = init_store(HostMemSize, cipher);
RPC_obj conn = create_rpc(enc_key); // create RPC handle
conn.reg_hdlr(&cmt_hdlr); // registers handlers
conn.reg_hdlr(&replication_hdlr);
// ----------- Raft leader -----------
for (auto& node : followers_list) {
   conn.wait_until_connected(node); // connects with followers
}
for (;;) {
   ... // gets client request and marks it as on-going
   for (auto& follower : connections)
   // generates a shielded message and bcast to followers
     follower.send(shield_msg(rep_req), TypeRepl);
   conn->poll(); // polls to flush TX/RX queues
   for (auto& follower : connections) ... // bcast commit
     follower.send(shield_msg(cmt_req), TypeCmt);
   ... // commit phase, apply changes to local kv
   ctx->kv.write(key, val);
}
```

Listing 1: Raft transformation using Recipe: blue sections (native Raft) and orange sections (Recipe additions).

Developers need to port the codebase within the TEE and use the Recipe's network API to replace the conventional unsecure RPC-API [67, 90, 110] with Recipe-lib's networking functions extending the TEEs' trust accross the network. Some of the Recipe's API remains equivalent to the native API; typical examples are the poll() and reg_hdlr() functions. On the other hand, we introduced some slight modifications in send() operation and Initialization APIs.

8.2 Recipe in Action for CFT Protocols

Experimental setup. We run our experiments in a cluster of three SGX machines (NixOS, 5.15.43) with CPU: Intel(R) Core(TM) i9-9900K each with 8 cores (16 HT), NIC: Intel Corporation Ethernet Controller XL710 for 40GbE QSFP+ (rev 02) and a 40GbE QSFP+ network switch. For the evaluation, we use the YCSB benchmark [26] (configured with approx. 10K distinct keys with Zipfian distribution) with various R/W ratios and value sizes. Competitive baselines (e.g., Damysus [56] and PBFT [140]) run on the same machines. PBFT used an additional machine with identical specs.

To show the benefits of our approach, we implement four widely adopted CFT protocols (one of each category in § 2, Table 1), with the Recipe-lib API. We build R-ABD, R-Raft, R-AllConcur and R-CR which are the Recipe versions of ABD, Raft, AllConcur and CR respectively. We compare these protocols with BFT-smart [140], an optimized version of PBFT [41] with system-level optimizations and real-world integration (e.g., Hyperledger Fabric) and Damysus [56], the state-of-the-art version of HotStuff [27] on top of SGX (with $2f+1$) including optimizations such as pipelining and concurrent processing.

Our evaluation targets the following: not only to show how Recipe's approach with TEEs can improve throughput using only

R/W ratio	R-ABD	R-CR	R-Raft	R-AllConcur
50%	6.5×	13.7×	5.3×	6.8×
75%	13.3×	14.8×	10.05×	9.4×
90%	13×	24×	16.5×	9×
95%	12.8×	21×	10.7×	9.5×
99%	12.3×	23×	9.8×	10.5×

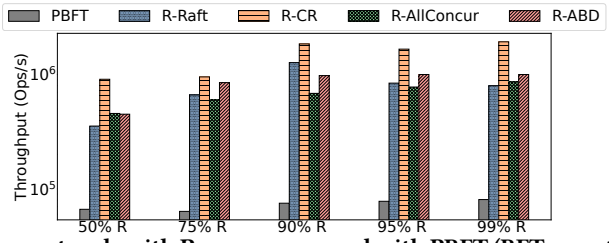

Figure 3: Speedup (left Table) and throughput (right Figure) of four protocols with Recipe compared with PBFT (BFT-smart).

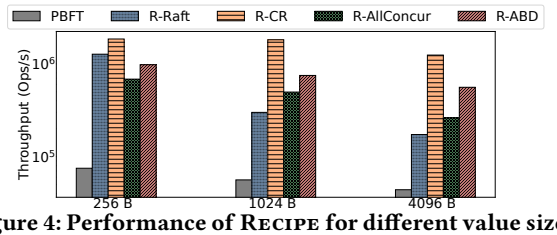

Figure 4: Performance of Recipe for different value sizes.

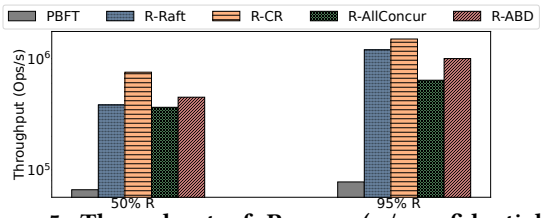

Figure 5: Throughput of Recipe (w/ confidentiality) compared with PBFT (BFT-Smart).

$2f + 1$ nodes but also, to explore the impact of kernel-bypass networking for BFT.

Our findings based on the evaluations showed that Recipe improves performance by preserving the original CFT protocol workflow. For example, it allows local reads where permitted, due to trusted leases and the authenticated per-node KV store. Additionally, kernel-bypass networking eliminates critical path bottlenecks improving throughput. Next, we discuss the characteristics of protocols categories, our chosen protocol and our evaluation results in detail.

A: Leaderless w/ per-key order. Protocols in this family agree on a per-key order of writes in a distributed manner. All nodes can coordinate a write that is completed in at least two rounds. A typical example is Classic Paxos (CP) that achieves consensus in three broadcast rounds. Several works [64, 82, 93, 107, 117] simplify the complexity of CP to boost performance. Protocols such as [64, 82, 117] can offer consensus in two rounds but fall back to CP if conflicts occur. Others [93, 107] execute writes in two rounds enforcing all messages to be received by all nodes or relaxing the Read-Modify-Write semantics. These protocols offer linearizable reads by executing quorum reads to consult (at least) the majority. Protocols like [93] where writes need to reach all nodes allow for local reads (at the cost of availability—if a node fails, writes block).

Choice: ABD [107]. We implemented ABD, a multi-writer, multi-reader protocol with Recipe (R-ABD). R-ABD offers linearizable (quorum) reads using Lamport timestamps (TS) [102] for each key-value (KV) pair. R-ABD broadcasts requests to all replicas and waits for acks from the quorum.

Writes are executed in two rounds of broadcasts. First, the coordinator asks from all replicas to hand over the key's TS, securely stored inside the TEE (KVs' metadata). Upon receiving a majority of the timestamps, the coordinator creates a higher TS for that key by increasing the highest received TS. Finally, it broadcasts the new KV pair along with its new TS to all replicas which, in turn, insert the KV pair into their KV store. Upon gathering a majority of acks it replies to the client.

R-ABD (usually) executes reads in one round by collecting all values (and their TS) from the majority. If the majority agrees on the latest seen TS, the coordinator replies to the client. Otherwise, the coordinator chooses the highest TS and invokes the second round of the write-path (for availability).

B: Leader-based w/ total ordering. The protocols [121, 131, 147] serialize writes at the leader, offering total order. The writes usually require two broadcast rounds; the leader proposes writes to (passive) followers, which they ack the proposal. Once the leader collects the acks from the majority, it runs the commit round where the nodes apply the proposed writes in their total order. Since writes are propagated to the majority where the leader is always part of it, the leader can always know the latest committed write for all keys. As such, leaders can always read locally while followers must forward reads to the leader. Some protocols [131] allow followers to read locally. This is achieved in two ways: they might forego linearizability and downgrade to sequential consistency [32] (with the possibility of reading stale values [131]), or ensure that all writes reach all followers at the cost of availability.

Choice: Raft [121]. As a representative protocol of this family we implement Raft with Recipe (R-Raft). We target linearizability; all reads are forwarded to the leader, which also serializes writes. The leader proposes writing to replicas and commits the request when the majority of them acknowledge the proposal.

The leader stores writes in an uncommitted_queue inside the TEE. We spawn a dedicated (worker) thread to manage this queue and serialize all writes. The worker thread broadcasts the request (or a batch of consecutive requests) to all followers. The followers verify the messages. As an optimization, followers accept future messages, storing them in a separate queue. The followers commit requests respecting the leader's total order and send acks for one or more consecutive requests. The leader only commits a request and responds to the client when it receives a response from the majority.

C: Leader-based w/ per-key order. In these protocols all writes are steered to the leader node, which ensures that writes to the same key are applied in the same order by all replicas. These protocols can

(a) Performance overheads of TEEs. **(b) Performance evaluation of Recipe-lib (net).**

Figure 6: Performance overheads of transformation and TEEs and performance analysis of Recipe networking.

offer linearizability (it is a compositional property) similarly to the leader-based protocols with total order. While writes are propagated to a majority of nodes, reads are propagated to the leader. As the protocols do not respect a total ordering, local reads to followers lead to weak guarantees such as eventual consistency [150]. As before, we can allow for local reads to all nodes when writes are guaranteed to propagate to all followers.

Choice: Chain Replication [132]. As a representative protocol, we build Chain Replication (R-CR) with Recipe. In R-CR, the nodes are organized in a chain, through which writes are propagated from the head of the chain to its tail. Similarly to [67], we consider CR among the leader-based protocols as the head node is the leader to serialize all writes. A write traverses the chain until it reaches the tail where it is considered committed, which guarantees that all writes reach all nodes. We offer linearizability by reading locally from the tail.

D: Leaderless w/ total ordering. These protocols rely on a predetermined static allocation of write-ids to nodes. For example, all nodes know that the writes 0 to N-1 will be proposed and coordinated by node-0, the next N writes will be proposed by node-1 and so on. Therefore, in each round each node can calculate the place of each write in the total order based on its own node-id, without synchronizing with any other node. Then, the node broadcasts its writes along with their place in the total order. Typically a commit message is broadcast after gathering acks from a majority of the nodes. Crucially, all nodes must apply the writes in the prescribed total order.

Choice: AllConcur [126]. To study this category, we implemented AllConcur with Recipe (R-AllConcur), a decentralized replication protocol with total order that relies on an atomic broadcast primitive. Nodes are organized in a digraph (G) [126] where the fault tolerance of the system is given by G's connectivity. For example, to tolerate 1 node failure on a 3-node system, we calculated the vertex-connectivity to be equal to 2, namely, each node is connected to the other 2 nodes. For the writes, all nodes track all messages for each round and commit them in a predefined order without synchronization. We can treat reads as writes (for linearizability) or, we allow for local reads to replicas offering sequential consistency [85].

8.3 Evaluation Analysis

Recipe vs PBFT. Figure 3 shows the throughput (Ops/s) and the speedup of the four case studies we implemented with Recipe compared to BFT-smart [140] (PBFT) for different read/write workloads (and constant value size/payload, 256 B). Our evaluation shows that all four protocols with Recipe outperform the classical BFT 5× to 24×. We observe that the local linearizable reads offered by R-CR greatly improve performance. We see less speedup in read-heavy workloads for the protocols with local reads (e.g., R-Raft and R-AllConcur). We found out that in these protocols, the total ordering was the bottleneck. In R-Raft, the writer thread that serialized all writes was slower than the other worker threads (which executed reads or enqueued writes to the writer thread's queue). Additionally, for R-AllConcur we saw that collecting all messages of each round decreased throughput. The speedup in R-ABD, R-Raft and R-AllConcur is moderate for write-heavy workloads where writes require two rounds of messages. R-ABD has lighter read-path; reads require the majority to agree on a value which is typically resolved in one round. R-CR outperforms R-ABD as reads are done locally. Lastly, we see that as the workload becomes more read heavy, the throughput is improved slightly due to (1) request rate limiter and (2) single-node bottlenecks.

Recipe vs Damysus. We compare Recipe against Damysus [56] with SGX in simulation mode due to SGX driver incompatibility with our system. Damysus achieves throughput of 320 kOp/s, 230 kOp/s and 152 kOp/s for payload sizes 0 B, 64 B and 256 B respectively. Our Recipe (with 256 B payload) outperforms 1.1×—2.8× and 2.3×—5.9× Damysus with 0 and 256 B payloads. Since the SGX simulation avoids syscall and enclave memory overheads [36, 69, 129, 141], it is typically 2–4x faster than real SGX hardware. As such, Recipe is shown to consistently outperform Damysus' idealized upper bound.

Recipe with confidentiality. Figure 5 shows Recipe throughput when we also strive for confidentiality; an extra property that is not offered by classical BFT protocols. We guarantee confidentiality by encrypting all data that leave the TEE (network messages, values residing in the host memory). Briefly, the cost for this extra property is a throughput decrement by a factor of 2. Surprisingly, R-ABD shows minimal degradation compared to R-ABD without confidentiality. The reason is that R-ABD quickly saturated all memory resources in our system so the throughput was limited mainly by the requests' rate limiter. Recipe, even with stronger properties, i.e., confidentiality, achieves higher throughput than PBFT: on average we calculate 7× and 13× speedup for 50% and 95% workloads respectively.

Recipe with confidentiality boosts throughput up to 4.9× w.r.t. Damysus that does not offer confidentiality.

Value size. Figure 4 shows the throughput for different value sizes (under a 90% R workload) for each of the four protocols. The performance drops as the value size is increased due to the enclave's limited size. While Recipe places the values and network buffers in the untrusted (unlimited) memory, the bigger the allocations are the more we stress test the (limited) enclave memory. R-Raft and R-AllConcur show the greatest slowdown (2× to 7× for 4096 B). We interestingly found out that the batching technique in these protocols with value size of 4096 B deteriorates the performance and, even,

	Mean / s	Speedup
Recipe CAS	0.169	18.2×
IAS	2.913	

Table 3: The end-to-end latency comparison between the attestation mechanisms using Recipe CAS and IAS.

crashes the system by consuming all Scone's memory. For these two protocols with value size 4096 B we depict the numbers with little (< 4) or no batching factor. The other two protocols, R-ABD and R-CR, also show similar behavior. In these protocols we did not use batching as an extra optimization.

Transformation and TEEs overheads. Figure 6a shows the overheads introduced by Recipe where we compare a native CFT implementations of the protocols (without TEEs). Both use the same network stack, but native CFT omits authentication, allowing us to isolate TEEs overheads. Overall, an R-CFT protocol experiences 2×—15× slowdown compared to its native execution. The overheads mainly derive from the TEEs. To prove that, we also ran these protocols in `simulation` mode in Scone where the trusted memory is unlimited: we found the throughput to be almost equivalent to the native runs' results. Our observation is also explained from the fact that the higher overheads are for AllConcur and Raft. To optimize these protocols we found extremely helpful the batching. However, batching requires allocations/de-allocations of bigger continuous (virtual) memory buffers which stress test Scone memory subsystem.

Recipe-lib network performance. Figure 6b shows the network throughput (Gbps) of five competitive network stacks: *(i)* a native and a TEE-based network stack on top of kernel sockets [18], *(ii)* a native and a TEE-based direct I/O for networking (RDMA/DPDK) and *(iii)* our TEE-based Recipe-lib network library. This is to isolate the performance gains of the RDMA-based stack in Recipe.

We deduct two core conclusions. First, TEEs (Scone) can degrade network throughput 4×—8× for both kernel-net and direct I/O networking compared to their unprotected (native) runs. Consequently, a naive adoption of TEEs for BFT does not necessarily translate to performance gains. Secondly, Recipe-lib network performs up to 1.66× faster than the kernel-based networking (kernel-net (TEEs)). As a takeaway the performance speedup (24× w.r.t. PBFT and 5.9× w.r.t. Damysus) for all our four use-cases with Recipe are primarily due to the transformation (Recipe) rather than the use of direct I/O.

Attestation. Table 3 shows the latencies of Intel's Attestation Service (IAS) [15] and Recipe CAS. We found that the (mean) average of our CAS is 0.17 s, i.e., 18× faster than the IAS (2.9 s). Note that since Recipe recovers by launching fresh replicas that immediately join the write path while synchronizing in-memory state from peers, recovery time is primarily influenced by CAS attestation latency.

9 Related Work

Classical BFT protocols. PBFT [41] and its variations [136] run a three-phase protocol. Replicas broadcast messages and transit to the next phases after receiving *quorum certificates* [40] from at least $2f+1$ distinct replicas leading to $O(n^2)$ message complexity. Zyzzyva [95] offloads to the clients the responsibility to correct replicas' state in case of a Byzantine primary. However, prior work [28] found safety concerns in the protocol.

Streamlined protocols [27, 39, 42, 43] avoid heavy state transfers at the view-change by rotating the leader on each command at the cost of additional rounds. HotStuff [27] adds two extra phases at the commit. Basil [143] targets *operability* when Byzantine nodes sabotage the execution requiring $5f+1$ replicas. Others focus on algorithmic optimizations [47, 72]. In contrast to our Recipe that operates with $2f+1$ nodes, these systems require (at least) $3f+1$ nodes.

Hybrid BFT protocols. *Hybrid* BFT protocols use trusted hardware to enhance performance, often at the cost of generality and ease of adoption. Examples include MinBFT [149]—unfortunately, incompatible with our SGX setup [19]— Damysus [56], OneShot [57] and Hybster [38], which use TEEs to reduce replication factors, communication rounds, or message complexity. CheapBFT [92] and FastBFT [106] operate with trusted modules and fewer active replicas but fall back to classical BFT under Byzantine faults. Others [46, 48, 77, 105, 153] incorporate TEEs, trusted counters and logs. HotStuff-TPM [27] uses TPM [124] with an additional phase. CCF [51] uses TEEs offering a REST API, requiring two requests per commit. Recipe commits in one, improving latency and throughput.

Achilles [118] and Engraft [151] improve rollback resilience using TEEs to offload rollback prevention from the critical path or stateful storage. Recipe addresses this through attestation and cooperative recovery, where nodes assist each other in state restoration.

Programmable hardware for BFT. Programmable hardware such as FPGAs, SmartCards, and switches has been used to support BFT primitives [71, 92, 105, 142]. NeoBFT [142] implements authenticated ordered multicast in switches but introduces a single point of failure. In contrast, Recipe distributes trust across TEEs for better availability and compatibility with unmodified CFT protocols. Trinc [105] and CheapBFT [92] rely on peripherals for attestations but suffer from high latency ((50us—105ms). In contrast to our Recipe, these approaches still require a working understanding of BFT; a task as challenging as it is error-prone [28].

10 Conclusion

We present Recipe, a generic approach for transforming CFT protocols to tolerate Byzantine failures without any modifications to the core of the protocols, e.g., states, message rounds, and complexity. We realize our approach by implementing Recipe-lib with trusted hardware to guarantee transferable authentication and non-equivocation for thwarting Byzantine errors. Further, we combine trusted hardware with direct network I/O [16, 110] for performance. We present an extensive evaluation of Recipe by applying it to four CFT protocols: Chain Replication, Raft, ABD, and AllConcur. All four protocols show that Recipe achieves up to 24× and 5.9× better throughput compared to state-of-the-art BFT protocol implementations.

Software availability. https://github.com/TUM-DSE/Recipe.git.

Acknowledgments. We thank our shepherd, Maarten van Steen, and Martin Kleppmann and Antonis Katsarakis for their helpful feedback. This work was supported in part by an ERC Starting Grant (ID: 101077577) and the Chips Joint Undertaking (JU), European Union (EU) HORIZON-JU-IA, under grant agreement No. 101140087 (SMARTY), the Intel Trustworthy Data Center of the Future (TDCoF), Google Research Grants and National Science Foundation grant PPoSS-2118512. The authors acknowledge the financial support by the Federal Ministry of Research, Technology and Space of Germany in the programme of "Souverän. Digital. Vernetzt.". Joint project 6G-life, project identification number: 16KISK002.

References

[1] 3 Pillars of Data Security: Confidentiality, Integrity and Availability. https://mark43.com/resources/blog/3-pillars-of-data-security-confidentiality-availability-integrity/. Last accessed: November 14, 2025.

[2] Alibaba cloud: Blockchain as a service. https://www.alibabacloud.com/product/baas. Last accessed: November 14, 2025.

[3] Arm Confidential Compute Architecture. https://www.arm.com/why-arm/architecture/security-features/arm-confidential-compute-architecture. Last accessed: May 2021.

[4] Blockchain on aws. https://aws.amazon.com/blockchain/. Last accessed: November 14, 2025.

[5] Case study: Ge healthcare takes dynamodb on-premises with scylladb's 'project alternator'. https://www.scylladb.com/users/case-study-ge-healthcare-takes-dynamodb-on-premises-with-scyllas-project-alternator/. Last accessed: November 14, 2025.

[6] Dataproc Confidential Compute . https://cloud.google.com/dataproc/docs/concepts/configuring-clusters/confidential-compute. Last accessed: November 14, 2025.

[7] Encrypt workload data in-use with Confidential Google Kubernetes Engine Nodes. https://cloud.google.com/kubernetes-engine/docs/how-to/confidential-gke-nodes. Last accessed: November 14, 2025.

[8] etcd. https://github.com/etcd-io/etcd. Last accessed: November 14, 2025.

[9] Hess corporation case study. https://aws.amazon.com/solutions/case-studies/hess-corporation/. Last accessed: November 14, 2025.

[10] The history of databases at netflix and how they use cockroachdb. https://www.cockroachlabs.com/blog/netflix-at-cockroachdb/. Last accessed: November 14, 2025.

[11] How are databases used in banking? https://globalbanks.com/how-are-databases-used-in-banking/. Last accessed: November 14, 2025.

[12] How big is rocksdb adoption? https://rocksdb.org/docs/support/faq.html. Last accessed: May 2021.

[13] Hyperledger fabric ordering service. https://github.com/hyperledger/fabric/tree/main/orderer#service-types. Last accessed: November 14, 2025.

[14] Ibm blockchain. https://www.ibm.com/blockchain. Last accessed: November 14, 2025.

[15] Intel Corporation. Attestation Service for Intel Software GuardExtensions (Intel SGX): API Documentation. https://api.trustedservices.intel.com/documents/sgx-attestation-api-spec.pdf. Last accessed: Jan, 2021.

[16] Intel DPDK. http://dpdk.org/. Last accessed: Jan, 2021.

[17] Intel tdx. https://www.intel.com/content/www/us/en/developer/articles/technical/intel-trust-domain-extensions.html. Last accessed: November 14, 2025.

[18] iPerf - The ultimate speed test tool for TCP, UDP nd SCTP. Last accessed: Aug, 2020.

[19] Minbft incompatibility. https://github.com/hyperledger-labs/minbft/issues/252.

[20] Oracle blockchain. https://www.oracle.com/blockchain/. Last accessed: November 14, 2025.

[21] . Ordering service implementations. Last accessed: November 14, 2025.

[22] SGX Monotonic Counters not supported. https://github.com/intel/linux-sgx/issues/424. Last accessed: November 14, 2025.

[23] Tamarin tls handshake proof. https://github.com/tamarin-prover/tamarin-prover/blob/develop/examples/classic/TLS_Handshake.spthy.

[24] Unable to find Alternatives to Monotonic Counter Application Programming Interfaces (APIs) in Intel Software Guard Extensions (Intel SGX) for Linux to Prevent Sealing Rollback Attacks. https://www.intel.co.uk/content/www/uk/en/support/articles/000057968/software/intel-security-products.html. Last accessed: November 14, 2025.

[25] Web3. https://azure.microsoft.com/en-us/solutions/web3. Last accessed: November 14, 2025.

[26] YCSB. https://github.com/brianfrankcooper/YCSB. Last accessed: Jan, 2021.

[27] Ittai Abraham, Guy Gueta, and Dahlia Malkhi. Hot-stuff the linear, optimal-resilience, one-message bft devil. *CoRR*, abs/1803.05069, 2018.

[28] Ittai Abraham, Guy Gueta, Dahlia Malkhi, Lorenzo Alvisi, Rama Kotla, and Jean-Philippe Martin. Revisiting fast practical byzantine fault tolerance, 2017.

[29] Peter A. Alsberg and John D. Day. A principle for resilient sharing of distributed resources. In *Proceedings of the 2nd International Conference on Software Engineering*, ICSE '76, page 562–570, Washington, DC, USA, 1976. IEEE Computer Society Press.

[30] AMD. AMD Secure Encrypted Virtualization (SEV). https://developer.amd.com/sev/. Last accessed: Jan, 2021.

[31] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolić, Sharon Weed Cocco, and Jason Yellick. Hyperledger fabric: A distributed operating system for permissioned blockchains. In *Proceedings of the Thirteenth EuroSys Conference*, EuroSys '18, New York, NY, USA, 2018. Association for Computing Machinery.

[32] Hagit Attiya and Jennifer L. Welch. Sequential consistency versus linearizability. *ACM Trans. Comput. Syst.*, 12(2):91–122, May 1994.

[33] Pierre-Louis Aublin, Rachid Guerraoui, Nikola Knežević, Vivien Quéma, and Marko Vukolić. The next 700 bft protocols. *ACM Trans. Comput. Syst.*, 32(4), jan 2015.

[34] Wei Bai, Shanim Sainul Abdeen, Ankit Agrawal, Krishan Kumar Attre, Paramvir Bahl, Ameya Bhagat, Gowri Bhaskara, Tanya Brokhman, Lei Cao, Ahmad Cheema, Rebecca Chow, Jeff Cohen, Mahmoud Elhaddad, Vivek Ette, Igal Figlin, Daniel Firestone, Mathew George, Ilya German, Lakhmeet Ghai, Eric Green, Albert Greenberg, Manish Gupta, Randy Haagens, Matthew Hendel, Ridwan Howlader, Neetha John, Julia Johnstone, Tom Jolly, Greg Kramer, David Kruse, Ankit Kumar, Erica Lan, Ivan Lee, Avi Levy, Marina Lipshteyn, Xin Liu, Chen Liu, Guohan Lu, Yuemin Lu, Xiakun Lu, Vadim Makhervaks, Ulad Malashanka, David A. Maltz, Ilias Marinos, Rohan Mehta, Sharda Murthi, Anup Namdhari, Aaron Ogus, Jitendra Padhye, Madhav Pandya, Douglas Phillips, Adrian Power, Suraj Puri, Shachar Raindel, Jordan Rhee, Anthony Russo, Maneesh Sah, Ali Sheriff, Chris Sparacino, Ashutosh Srivastava, Weixiang Sun, Nick Swanson, Fuhou Tian, Lukasz Tomczyk, Vamsi Vadlamuri, Alec Wolman, Ying Xie, Joyce Yom, Lihua Yuan, Yanzhao Zhang, and Brian Zill. Empowering azure storage with RDMA. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 49–67, Boston, MA, April 2023. USENIX Association.

[35] Maurice Bailleu, Dimitra Giantsidi, Vasilis Gavrielatos, Do Le Quoc, Vijay Nagarajan, and Pramod Bhatotia. Avocado: A secure In-Memory distributed storage system. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pages 65–79. USENIX Association, 2021.

[36] Maurice Bailleu, Jörg Thalheim, Pramod Bhatotia, Christof Fetzer, Michio Honda, and Kapil Vaswani. SPEICHER: Securing lsm-based key-value stores using shielded execution. In *17th USENIX Conference on File and Storage Technologies (FAST)*, 2019.

[37] Sumeet Bajaj and Radu Sion. Trusteddb: a trusted hardware based database with privacy and data confidentiality. In *In Proceedings of the 2011 international conference on Management of data*, pages 205–216. ACM, 2011.

[38] Johannes Behl, Tobias Distler, and Rüdiger Kapitza. Hybrids on steroids: Sgx-based high performance bft. EuroSys '17, page 222–237, New York, NY, USA, 2017. Association for Computing Machinery.

[39] Ethan Buchman, Jae Kwon, and Zarko Milosevic. The latest gossip on BFT consensus. *CoRR*, abs/1807.04938, 2018.

[40] M. Castro, P. Druschel, A.-M. Kermarrec, and Antony Rowstron. Scribe: a large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications*, 2002.

[41] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 2002.

[42] T-H. Hubert Chan, Rafael Pass, and Elaine Shi. Pala: A simple partially synchronous blockchain. *IACR Cryptol. ePrint Arch.*, 2018:981, 2018.

[43] T-H. Hubert Chan, Rafael Pass, and Elaine Shi. Pili: An extremely simple synchronous blockchain. *IACR Cryptol. ePrint Arch.*, 2018:980, 2018.

[44] Tushar Chandra, Vassos Hadzilacos, and Sam Toueg. An algorithm for replicated objects with efficient reads. pages 325–334, 07 2016.

[45] Stephen Chong, K. Vikram, and Andrew C. Myers. Sif: Enforcing confidentiality and integrity in web applications. In *USENIX Security Symposium*, 2007.

[46] David Chu, Aditya Balasubramanian, Dee Bao, Natacha Crooks, Heidi Howard, Lucky E. Katahanas, and Soujanya Ponnapalli. Rollbaccine : Herd immunity against storage rollback attacks in tees [technical report], 2025.

[47] David C. Y. Chu, Chris Liu, Natacha Crooks, Joseph M. Hellerstein, and Heidi Howard. Bigger, not badder: Safely scaling bft protocols. In *Proceedings of the 11th Workshop on Principles and Practice of Consistency for Distributed Data*, PaPoC '24, page 30–36, New York, NY, USA, 2024. Association for Computing Machinery.

[48] Byung-Gon Chun, Petros Maniatis, Scott Shenker, and John Kubiatowicz. Attested append-only memory: Making adversaries stick to their word. *SIGOPS Oper. Syst. Rev.*, 41(6):189–204, oct 2007.

[49] Allen Clement, Flavio Junqueira, Aniket Kate, and Rodrigo Rodrigues. On the (limited) power of non-equivocation. In *Proceedings of the 2012 ACM Symposium on Principles of Distributed Computing*, PODC '12, page 301–308, New York, NY, USA, 2012. Association for Computing Machinery.

[50] Alibaba Cloud. Alibaba Cloud's Next-Generation Security Makes Gartner's Report. https://www.alibabacloud.com/blog/alibaba-clouds-next-generation-security-makes-gartners-report_595367. Last accessed: Jan, 2021.

[51] CCF documentation. https://microsoft.github.io/CCF/master/. Last accessed: Jan, 2021.

[52] Victor Costan and Srinivas Devadas. Intel SGX Explained, 2016.

[53] Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott, and Thyla Van Der Merwe. A comprehensive symbolic analysis of tls 1.3. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, pages 1773–1788, 2017.

[54] Cas Cremers, Marko Horvat, Sam Scott, and Thyla van der Merwe. Automated analysis and verification of tls 1.3: 0-rtt, resumption and delayed authentication. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 470–485. IEEE, 2016.

[55] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's highly available key-value store. *ACM*

SIGOPS Operating Systems Review (SIGOPS), 2007.

[56] Jérémie Decouchant, David Kozhaya, Vincent Rahli, and Jiangshan Yu. Damysus: Streamlined bft consensus leveraging trusted components. In *Proceedings of the Seventeenth European Conference on Computer Systems*, EuroSys '22, page 1–16, New York, NY, USA, 2022. Association for Computing Machinery.

[57] Jérémie Decouchant, David Kozhaya, Vincent Rahli, and Jiangshan Yu. Oneshot: View-adapting streamlined bft protocols with trusted execution environments. In *2024 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1022–1033, 2024.

[58] Carole Delporte-Gallet, Hugues Fauconnier, Felix Freiling, Lucia Penso, and Andreas Tielmann. From crash-stop to permanent omission: Automatic transformation and weakest failure detectors. volume 4731, pages 165–178, 09 2007.

[59] Danny Dolev and Andrew Yao. On the security of public key protocols. *IEEE Transactions on information theory*, 29(2):198–208, 1983.

[60] John R. Douceur. The sybil attack. In Peter Druschel, Frans Kaashoek, and Antony Rowstron, editors, *Peer-to-Peer Systems*, pages 251–260, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[61] Aleksandar Dragojević, Dushyanth Narayanan, Miguel Castro, and Orion Hodson. FaRM: Fast remote memory. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, 2014.

[62] Aleksandar Dragojević, Dushyanth Narayanan, Edmund B. Nightingale, Matthew Renzelmann, Alex Shamis, Anirudh Badam, and Miguel Castro. No compromises: Distributed transactions with consistency, availability, and performance. In *Proceedings of the 25th Symposium on Operating Systems Principles*, SOSP '15, page 54–70, New York, NY, USA, 2015. Association for Computing Machinery.

[63] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, apr 1988.

[64] Vitor Enes, Carlos Baquero, Tuanir França Rezende, Alexey Gotsman, Matthieu Perrin, and Pierre Sutra. State-machine replication for planet-scale systems (extended version), 2020.

[65] C. Fetzer and F. Cristian. A highly available local leader election service. *IEEE Transactions on Software Engineering*, 25(5):603–618, 1999.

[66] Brad Fitzpatrick. Distributed caching with memcached. *Linux Journal*, 2004.

[67] Vasilis Gavrielatos, Antonis Katsarakis, and Vijay Nagarajan. Odyssey: The impact of modern hardware on strongly-consistent replication protocols. In *Proceedings of the 16th European Conference on Computer Systems EuroSys'21*, page 245–260, United States, April 2021. Association for Computing Machinery (ACM). 16th ACM EuroSys Conference on Computer Systems, EuroSys 2021 ; Conference date: 26-04-2021 Through 28-04-2021.

[68] Hemant Ghayvat, Sharnil Pandya, Pronaya Bhattacharya, Mohd Zuhair, Mamoon Rashid, Saqib Hakak, and Kapal Dev. Cp-bdhca: Blockchain-based confidentiality-privacy preserving big data scheme for healthcare clouds and applications. *IEEE Journal of Biomedical and Health Informatics*, 26(5):1937–1948, 2022.

[69] Dimitra Giantsidi, Maurice Bailleu, Natacha Crooks, and Pramod Bhatotia. Treaty: Secure distributed transactions. In *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 14–27, 2022.

[70] Dimitra Giantsidi, Emmanouil Giortamis, Nathaniel Tornow, Florin Dinu, and Pramod Bhatotia. Flexlog: A shared log for stateful serverless computing. In *Proceedings of the 32nd International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '23, page 195–209, New York, NY, USA, 2023. Association for Computing Machinery.

[71] Dimitra Giantsidi, Julian Pritzi, Felix Gust, Antonios Katsarakis, Atsushi Koshiba, and Pramod Bhatotia. *TNIC: A Trusted NIC Architecture: A hardware-network substrate for building high-performance trustworthy distributed systems*, page 1282–1301. Association for Computing Machinery, New York, NY, USA, 2025.

[72] Neil Giridharan, Florian Suri-Payer, Ittai Abraham, Lorenzo Alvisi, and Natacha Crooks. Autobahn: Seamless high speed bft. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, SOSP '24, page 1–23, New York, NY, USA, 2024. Association for Computing Machinery.

[73] Introducing Google Cloud Confidential Computing with Confidential VMs. https://cloud.google.com/blog/products/identity-security/introducing-google-cloud-confidential-computing-with-confidential-vms. Last accessed: Jan, 2021.

[74] C. Gray and D. Cheriton. Leases: An efficient fault-tolerant mechanism for distributed file cache consistency. In *Proceedings of the Twelfth ACM Symposium on Operating Systems Principles*, SOSP '89, page 202–210, New York, NY, USA, 1989. Association for Computing Machinery.

[75] C. Gray and D. Cheriton. Leases: An efficient fault-tolerant mechanism for distributed file cache consistency. *SIGOPS Oper. Syst. Rev.*, 23(5):202–210, nov 1989.

[76] Haryadi S. Gunawi, Mingzhe Hao, Tanakorn Leesatapornwongsa, Tiratat Patana-anake, Thanh Do, Jeffry Adityatama, Kurnia J. Eliazar, Agung Laksono, Jeffrey F. Lukman, Vincentius Martin, and Anang D. Satria. What Bugs Live in the Cloud? A Study of 3000+ Issues in Cloud Systems. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC)*, 2014.

[77] Suyash Gupta, Sajjad Rahnama, Shubham Pandey, Natacha Crooks, and Mohammad Sadoghi. Dissecting bft consensus: In trusted components we trust! In *Proceedings of the Eighteenth European Conference on Computer Systems*, EuroSys '23, page 521–539, New York, NY, USA, 2023. Association for Computing Machinery.

[78] Marcus Hähnel, Weidong Cui, and Marcus Peinado. High-resolution side channels for untrusted operating systems. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, 2017.

[79] Maurice P. Herlihy and Jeannette M. Wing. Linearizability: A Correctness Condition for Concurrent Objects. *ACM Trans. Program. Lang. Syst.*

[80] Chi Ho, Danny Dolev, and Robbert Van Renesse. Making distributed applications robust. pages 232–246, 12 2007.

[81] Chi Ho, Robbert Van Renesse, Mark Bickford, and Danny Dolev. Nysiad: Practical protocol transformation to tolerate byzantine failures. In *5th USENIX Symposium on Networked Systems Design and Implementation (NSDI 08)*, San Francisco, CA, 2008. USENIX Association.

[82] Heidi Howard. *Distributed consensus revised.* PhD thesis, 09 2018.

[83] J. Huang, X. Ouyang, J. Jose, M. Wasi-ur-Rahman, H. Wang, M. Luo, H. Subramoni, C. Murthy, and D. K. Panda. High-Performance Design of HBase with RDMA over InfiniBand. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium (IPDPS)*, 2012.

[84] Peng Huang, Chuanxiong Guo, Jacob R. Lorch, Lidong Zhou, and Yingnong Dang. Capturing and enhancing in situ system observability for failure detection. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 1–16, Carlsbad, CA, October 2018. USENIX Association.

[85] Patrick Hunt, Mahadev Konar, Flavio P. Junqueira, and Benjamin Reed. ZooKeeper: Wait-free Coordination for Internet-scale Systems. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference.*

[86] Hyperledger Fabric. BFT-smart in Hyperledger Fabric. Last accessed: November 14, 2025.

[87] Intel Software Guard Extensions (Intel SGX). https://software.intel.com/en-us/sgx. Last accessed: Jan, 2021.

[88] Sagar Jha, Jonathan Behrens, Theo Gkountouvas, Matthew Milano, Weijia Song, Edward Tremel, Robbert Van Renesse, Sydney Zink, and Kenneth P. Birman. Derecho: Fast state machine replication for cloud services. *ACM Trans. Comput. Syst.*, 36(2), apr 2019.

[89] J. Jose, H. Subramoni, M. Luo, M. Zhang, J. Huang, M. Wasi-ur-Rahman, N. S. Islam, X. Ouyang, H. Wang, S. Sur, and D. K. Panda. Memcached Design on High Performance RDMA Capable Interconnects. In *International Conference on Parallel Processing (ICPP)*, 2011.

[90] Anuj Kalia, Michael Kaminsky, and David Andersen. Datacenter RPCs can be General and Fast. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2019.

[91] Anuj Kalia, Michael Kaminsky, and David G. Andersen. FaSST: Fast, Scalable and Simple Distributed Transactions with Two-Sided (RDMA) Datagram RPCs. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016.

[92] Rüdiger Kapitza, Johannes Behl, Christian Cachin, Tobias Distler, Simon Kuhnle, Seyed Vahid Mohammadi, Wolfgang Schröder-Preikschat, and Klaus Stengel. Cheapbft: Resource-efficient byzantine fault tolerance. In *Proceedings of the 7th ACM European Conference on Computer Systems*, EuroSys '12, page 295–308, New York, NY, USA, 2012. Association for Computing Machinery.

[93] Antonios Katsarakis, Vasilis Gavrielatos, M.R. Siavash Katebzadeh, Arpit Joshi, Aleksandar Dragojevic, Boris Grot, and Vijay Nagarajan. Hermes: A Fast, Fault-Tolerant and Linearizable Replication Protocol. In *Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2020.

[94] Taehoon Kim, Joongun Park, Jaewook Woo, Seungheun Jeon, and Jaehyuk Huh. ShieldStore: Shielded In-Memory Key-Value Storage with SGX. In *Proceedings of the Fourteenth EuroSys Conference 2019 (EuroSys)*, 2019.

[95] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. Zyzzyva: Speculative byzantine fault tolerance. *ACM Trans. Comput. Syst.*, 27(4), jan 2010.

[96] Tsung-Ting Kuo, Hyeon-Eui Kim, and Lucila Ohno-Machado. Blockchain distributed ledger technologies for biomedical and health care applications. *Journal of the American Medical Informatics Association*, 24(6):1211–1220, 09 2017.

[97] Dmitrii Kuvaiskii, Oleksii Oleksenko, Sergei Arnautov, Bohdan Trach, Pramod Bhatotia, Pascal Felber, and Christof Fetzer. SGXBOUNDS: Memory Safety for Shielded Execution. In *Proceedings of the 12th ACM European Conference on Computer Systems (EuroSys)*, 2017.

[98] Dmitrii Kuvaiskii, Dimitrios Stavrakakis, Kailun Qin, Cedric Xing, Pramod Bhatotia, and Mona Vij. Gramine-tdx: A lightweight os kernel for confidential vms. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, CCS '24, page 4598–4612, New York, NY, USA, 2024. Association for Computing Machinery.

[99] Avinash Lakshman and Prashant Malik. Cassandra: structured storage system on a p2p network. In *Proceedings of the 28th ACM Symposium on Principles of distributed computing (PODC)*. ACM, 2009.

[100] Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Transactions on Computers*, C-28(9):690–691, 1979.

[101] Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, may 1998.

[102] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine Generals Problem. *ACM Trans. Program. Lang. Syst.*, 1982.

[103] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Krste Asanović, and Dawn Song. Keystone: an open framework for architecting trusted execution environments. In *Proceedings of the Fifteenth European Conference on Computer Systems (EuroSys)*, 2020.

[104] LevelDB. http://leveldb.org/. Last accessed: Dec, 2018.

[105] Dave Levin, John R Douceur, Jacob R Lorch, and Thomas Moscibroda. Trinc: Small trusted hardware for large distributed systems. In *NSDI*, volume 9, pages 1–14, 2009.

[106] Jian Liu, Wenting Li, Ghassan O. Karame, and N. Asokan. Scalable byzantine consensus via hardware-assisted secret sharing. *CoRR*, abs/1612.04997, 2016.

[107] N. A. Lynch and A. A. Shvartsman. Robust emulation of shared memory using dynamic quorum-acknowledged broadcasts. In *Proceedings of IEEE 27th International Symposium on Fault Tolerant Computing (FtCS)*, 1997.

[108] Mads Frederik Madsen and Søren Debois. On the subject of non-equivocation: Defining non-equivocation in synchronous agreement systems. In *Proceedings of the 39th Symposium on Principles of Distributed Computing*, PODC '20, page 159–168, New York, NY, USA, 2020. Association for Computing Machinery.

[109] Simon Meier, Benedikt Schmidt, Cas Cremers, and David Basin. The tamarin prover for the symbolic analysis of security protocols. In *Proceedings of the 25th International Conference on Computer Aided Verification (CAV)*, 2013.

[110] Melanox. RDMA Aware Networks Programming User Manual. Last accessed: November 14, 2025.

[111] Ralph C. Merkle. Secure communications over insecure channels. *Commun. ACM*, 21(4):294–299, apr 1978.

[112] Ines Messadi, Markus Horst Becker, Kai Bleeke, Leander Jehl, Sonia Ben Mokhtar, and Rüdiger Kapitza. Splitbft: Improving byzantine fault tolerance safety using trusted compartments. In *Proceedings of the 23rd ACM/IFIP International Middleware Conference*, Middleware '22, page 56–68, New York, NY, USA, 2022. Association for Computing Machinery.

[113] Microsoft Azure. Azure confidential computing. https://azure.microsoft.com/en-us/solutions/confidential-compute/. Last accessed: Jan, 2021.

[114] Masanori Misono, Dimitrios Stavrakakis, Nuno Santos, and Pramod Bhatotia. Confidential vms explained: An empirical analysis of amd sev-snp and intel tdx. *Proc. ACM Meas. Anal. Comput. Syst.*, 8(3), December 2024.

[115] Christopher Mitchell, Yifeng Geng, and Jinyang Li. Pilaf: Using One-Sided RDMA Reads to Build a Fast, CPU-Efficient Key-Value Store. *Atc '13*, 2013.

[116] Christopher Mitchell, Yifeng Geng, and Jinyang Li. Using One-Sided RDMA Reads to Build a Fast, CPU-Efficient Key-Value Store. In *Proceedings of the 2013 USENIX Conference on Annual Technical Conference (ATC)*, 2013.

[117] Iulian Moraru, David G. Andersen, and Michael Kaminsky. There is more consensus in egalitarian parliaments. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP '13, page 358–372, New York, NY, USA, 2013. Association for Computing Machinery.

[118] Jianyu Niu, Xiaoqing Wen, Guanlong Wu, Shengqi Liu, Jiangshan Yu, and Yinqian Zhang. Achilles: Efficient tee-assisted bft consensus via rollback resilient recovery. In *Proceedings of the Twentieth European Conference on Computer Systems*, EuroSys '25, page 193–210, New York, NY, USA, 2025. Association for Computing Machinery.

[119] National Library of Medicine. Beyond the hipaa privacy rule: Enhancing privacy, improving health through research. https://www.ncbi.nlm.nih.gov/books/NBK9579/. Last accessed: November 14, 2025.

[120] Rui Oliveira, José Pereira, and André Schiper. Primary-backup replication: From a time-free protocol to a time-based implementation. pages 14–23, 02 2001.

[121] Diego Ongaro and John Ousterhout. In Search of an Understandable Consensus Algorithm. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference (ATC)*, 2014.

[122] Diego Ongaro, Stephen M. Rumble, Ryan Stutsman, John Ousterhout, and Mendel Rosenblum. Fast crash recovery in ramcloud. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, page 29–41, New York, NY, USA, 2011. Association for Computing Machinery.

[123] Ricardo Padilha and Fernando Pedone. Belisarius: Bft storage with confidentiality. In *2011 IEEE 10th International Symposium on Network Computing and Applications*, pages 9–16, 2011.

[124] Bryan Parno, Jonathan M. McCune, and Adrian Perrig. Bootstrapping Trust in Commodity Computers. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy (S&P)*, 2010.

[125] Bryan Parno, Jonathan M. McCune, and Adrian Perrig. *Bootstrapping Trust in Modern Computers*. Springer, August 2011.

[126] Marius Poke, Torsten Hoefler, and Colin W. Glass. Allconcur: Leaderless concurrent atomic broadcast (extended version). *ArXiv*, abs/1608.05866, 2016.

[127] Raluca Ada Popa, Catherine Redfield, Nickolai Zeldovich, and Hari Balakrishnan. CryptDB: protecting confidentiality with encrypted query processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles (SOSP)*, 2011.

[128] Daniel Porto, João Leitão, Cheng Li, Allen Clement, Aniket Kate, Flavio Junqueira, and Rodrigo Rodrigues. Visigoth fault tolerance. In *Proceedings of the Tenth European Conference on Computer Systems (EuroSys)*, 2015.

[129] C. Priebe, K. Vaswani, and M. Costa. EnclaveDB: A Secure Database using SGX (S&P). In *IEEE Symposium on Security and Privacy*, 2018.

[130] Redis. https://redis.io/.

[131] Benjamin C. Reed and Flavio Paiva Junqueira. A simple totally ordered broadcast protocol. In *LADIS '08*, 2008.

[132] Robbert Van Renesse and Fred B. Schneider. Chain replication for supporting high throughput and availability. In *6th Symposium on Operating Systems Design & Implementation (OSDI 04)*, San Francisco, CA, 2004. USENIX Association.

[133] RISC-V. Keystone Open-source Secure Hardware Enclave. https://keystone-enclave.org/. Last accessed: Jan, 2021.

[134] RocksDB, A persistent key-value store. https://rocksdb.org/. Last accessed: Dec, 2018.

[135] Patrick Sabanic, Masanori Misono, Teofil Bodea, Julian Pritzi, Michael Hackl, Dimitrios Stavrakakis, and Pramod Bhatotia. Confidential serverless computing, 2025.

[136] Alex Shamis, Peter Pietzuch, Burcu Canakci, Miguel Castro, Cedric Fournet, Edward Ashton, Amaury Chamayou, Sylvan Clebsch, Antoine Delignat-Lavaud, Matthew Kerner, Julien Maffre, Olga Vrousgou, Christoph M. Wintersteiger, Manuel Costa, and Mark Russinovich. IA-CCF: Individual accountability for permissioned ledgers. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 467–491, Renton, WA, April 2022. USENIX Association.

[137] Aneesh Sharma, Jerry Jiang, Praveen Bommannavar, Brian Larson, and Jimmy Lin. Graphjet: Real-time content recommendations at twitter. *Proc. VLDB Endow.*, 9(13):1281–1292, sep 2016.

[138] Shweta Shinde, Zheng Leong Chua, Viswesh Narayanan, and Prateek Saxena. Preventing page faults from telling your secrets. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '16, page 317–328, New York, NY, USA, 2016. Association for Computing Machinery.

[139] Livio Soares and Michael Stumm. FlexSC: Flexible System Call Scheduling with Exception-less System Calls. In *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2010.

[140] João Sousa and Alysson Bessani. From byzantine consensus to bft state machine replication: A latency-optimal transformation. *Proceedings - 9th European Dependable Computing Conference, EDCC 2012*, 05 2012.

[141] Dimitrios Stavrakakis, Dimitra Giantsidi, Maurice Bailleu, Philip Sändig, Shady Issa, and Pramod Bhatotia. Anchor: A library for building secure persistent memory systems. *Proc. ACM Manag. Data*, 1(4), December 2023.

[142] Guangda Sun, Mingliang Jiang, Xin Zhe Khooi, Yunfan Li, and Jialin Li. Neobft: Accelerating byzantine fault tolerance using authenticated in-network ordering. In *Proceedings of the ACM SIGCOMM 2023 Conference*, ACM SIGCOMM '23, page 239–254, New York, NY, USA, 2023. Association for Computing Machinery.

[143] Florian Suri-Payer, Matthew Burke, Zheng Wang, Yunhao Zhang, Lorenzo Alvisi, and Natacha Crooks. Basil. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles CD-ROM*. ACM, oct 2021.

[144] Jeff Terrace and Michael J. Freedman. Object storage on craq: High-throughput chain replication for read-mostly workloads. In *Proceedings of the 2009 Conference on USENIX Annual Technical Conference*, USENIX'09, page 11, USA, 2009. USENIX Association.

[145] Jörg Thalheim, Harshavardhan Unnibhavi, Christian Priebe, Pramod Bhatotia, and Peter Pietzuch. Rkt-io: A direct i/o stack for shielded execution. In *Proceedings of the Sixteenth European Conference on Computer Systems (ACM EuroSys 21)*, 2021.

[146] Bohdan Trach, Rasha Faqeh, Oleksii Oleksenko, Wojciech Ozga, Pramod Bhatotia, and Christof Fetzer. T-lease: A trusted lease primitive for distributed systems. In *Proceedings of the 11th ACM Symposium on Cloud Computing*, SoCC '20, page 387–400, New York, NY, USA, 2020. Association for Computing Machinery.

[147] Robbert Van Renesse and Deniz Altinbuken. Paxos made moderately complex. *ACM Comput. Surv.*, 47(3), feb 2015.

[148] Petros Venetis, Alon Halevy, Jayant Madhavan, Marius Paşca, Warren Shen, Fei Wu, Gengxin Miao, and Chung Wu. Recovering semantics of tables on the web. *Proc. VLDB Endow.*, 4(9):528–538, jun 2011.

[149] Giuliana Veronese, Miguel Correia, Alysson Bessani, Lau Lung, and Paulo Veríssimo. Efficient byzantine fault-tolerance. *Computers, IEEE Transactions on*, 62:16–30, 01 2013.

[150] Werner Vogels. Eventually consistent. *Commun. ACM*, 52(1):40–44, jan 2009.

[151] Weili Wang, Sen Deng, Jianyu Niu, Michael K. Reiter, and Yinqian Zhang. Engraft: Enclave-guarded raft on byzantine faulty nodes. CCS '22, page 2841–2855, New York, NY, USA, 2022. Association for Computing Machinery.

[152] Ofir Weisse, Valeria Bertacco, and Todd Austin. Regaining lost cycles with hotcalls: A fast interface for sgx secure enclaves. *SIGARCH Comput. Archit. News*, 45(2):81–93, jun 2017.

[153] Sravya Yandamuri, Ittai Abraham, Kartik Nayak, and Michael Reiter. Brief Announcement: Communication-Efficient BFT Using Small Trusted Hardware to Tolerate Minority Corruption. In Seth Gilbert, editor, *35th International Symposium on Distributed Computing (DISC 2021)*, volume 209 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 62:1–62:4, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.