Making Intrusion Tolerance Accessible: A Cloud-Based Hybrid Management Approach to Deploying Resilient Systems

Maher Khan and Amy Babay
University of Pittsburgh, School of Computing and Information
{maherkhan, babay}@pitt.edu

***Abstract*—Even with the rise of cyberattacks on high-value systems, we still do not see widespread adoption of intrusion-tolerant replication protocols, despite their long history in the research community and potential to support the needed resiliency. A key barrier is that deploying and managing intrusion-tolerant systems in practice requires substantial investment in additional physical infrastructure, as well as specialized technical expertise.**

In this work, we address this gap by designing a hybrid management model: while the system operator manages their application, a service provider hosts and manages the intrusion-tolerant replication service using cloud infrastructure. We develop the protocols to support this system architecture, without revealing application state, algorithms, or client information to the cloud provider, even when application servers are compromised. We implement and evaluate our approach in the context of an industrial control system and show that it meets the system's performance and resilience requirements.

I. Introduction

Cyberattacks on high-value systems continue to increase, with power grid, pipeline, and hospital systems (among others) experiencing high profile attacks [16], [30], [2]. In this hostile environment, intrusion-tolerant or Byzantine Fault Tolerant (BFT) replication can improve systems' resilience, allowing them to operate correctly even while partially compromised by an attacker [13]. However, despite a long history of research on BFT replication, these techniques have not been widely adopted in industry.

There has been progress in making BFT replication easier to integrate into practical systems, with libraries such as UpRight [14] and BFT-SMaRt [9], [12]. However, a barrier to deployment is that integrating an existing application with a BFT library is not enough to withstand sophisticated attacks. A practical intrusion-tolerant system must not only employ BFT agreement, but must also use proactive recovery to periodically refresh system replicas [13], [38], employ diversity to ensure replicas cannot be compromised by shared vulnerabilities [20], [33], and be deployed across multiple geographic sites to overcome sophisticated network attacks that can isolate a site [7]. Such a system requires a relatively large number of diverse replicas distributed over multiple diverse geographic sites, and becomes complex to manage and expensive to build.

For critical infrastructure applications (e.g. power grid, pipeline, water treatment, and healthcare systems), it is unlikely that each system operator will be able to build such a system for themselves. Moreover, even if this was feasible, a system built by any single operator has inherent fragility in that the entire system is under a single management domain, and therefore subject to shared vulnerabilities and/or misconfigurations. In practice, a single-operator system is also likely to be more limited in its geographic span and physical redundancy due to the cost of building dedicated infrastructure. On top of these challenges, maintaining and managing the system over time requires specialized expertise to reason about the underlying BFT replication protocols.

Our objective is to make intrusion tolerance *accessible* to system operators by allowing operators to essentially purchase intrusion-tolerant-replication-as-a-service from a cloud service provider. We aim to make it possible for operators to deploy extremely resilient systems, while limiting the amount of new physical infrastructure and new software components or diverse variants that they need to deploy and manage.

Our core contribution is to introduce a *hybrid management model*, in which system operators maintain control of their applications but use a generic intrusion-tolerant replication service managed by a cloud service provider to provide the needed resilience. We show that this approach significantly reduces the management demands on the system operator, while withstanding a highly demanding threat model that includes both intrusions and network attacks. In fact, the hybrid management model further enhances resilience by introducing *management diversity*. We show that this makes it possible to tolerate a new class of failure that affects an entire management domain.

We define an *intrusion-tolerant ordering and encrypted storage service* that can be provided by a cloud-managed BFT Replication Engine. The management and physical deployment of this service is completely decoupled from the applications it serves. Neither the system operator nor the cloud provider needs to know any internal details of the other: each simply needs to know a single public key to use to authenticate received messages, and a single multicast address to send messages to.

This decoupling makes it feasible for the cloud service provider to build a highly resilient BFT engine by amortizing costs over many applications and system operators (customers). The BFT engine can be physically hosted in cloud data centers that are distributed over a broad geographic area and connected redundantly via multiple Internet Service

Providers (ISPs). The service provider also has an incentive to invest in expensive diversity techniques, such as N-version programming [5] to create diverse BFT engine variants, since these variants can be used for many applications. Moreover, as the service provider improves their BFT engine, all of their customers can immediately benefit with no additional effort. Because management is decoupled, the service provider can perform software upgrades of the BFT engine, improving performance and fixing bugs, without any involvement from or coordination with the operators they serve.

At the same time, this decoupling allows system operators to retain full control over their applications. For many high-value applications, running the application itself in the cloud is infeasible or undesirable. Some systems need specialized hardware or client communication infrastructure (e.g. clients that do not communicate over IP) that the cloud infrastructure does not support, while for many others, operators are unwilling to store potentially sensitive data in the cloud. Our approach allows system operators to host their applications *on-premises* in sites and servers they fully control, and enforces strict confidentiality of the system operator's data: only encrypted state and requests are stored in the cloud. No application state, state manipulation algorithms, or client information is exposed to the cloud. This is critical to make the use of the cloud acceptable in practice [24], [6].

To realize this vision, we design a new system architecture and protocols to support our hybrid management model and intrusion-tolerant ordering and encrypted storage service. To provide a simple interface based on a single public key, our architecture employs threshold signatures to authenticate all messages sent between the cloud and application domains. However, this requires developing new protocols to process requests, address replay attacks that become possible in this model, and enable transferring encrypted state checkpoints to the cloud and retrieving them as needed to recover from site failures, network attacks, or management domain failures.

The main contributions of our work are:

- We define a new hybrid management model for intrusion-tolerant systems, where system operators control their applications, but leverage *intrusion-tolerant ordering* and *encrypted storage* services from a cloud provider.
- We design a concrete system architecture that implements the hybrid management model and enforces the confidentiality of application state, algorithms, and client request patterns.
- We show that this system architecture can provide resilience to a broad threat model that includes intrusions and network attacks, and is able to recover from management domain failures that affect all replicas hosted by the system operator (on-premises).
- We implement and evaluate the architecture in the context of an industrial control application. We show that, while it increases latency by about 9ms (18%) compared to a fully system-operator-managed BFT system, it still meets the application's performance requirements.

II. Background and Related Work

A. BFT Basics

Byzantine Fault Tolerant (BFT) replication protocols enable systems to work correctly in the presence of compromised servers [13]. These protocols are based on state machine replication [37]: replicas run an agreement protocol to establish a total order on requests submitted to the system, and all correct replicas execute requests in the determined order. Traditionally, BFT replication protocols require $3f+1$ total replicas to tolerate f compromised/Byzantine replicas [13].

Proactive Recovery: Basic BFT replication only supports f compromises over the entire lifetime of the system. To support long system lifetimes, BFT systems employ proactive recovery, periodically taking down each replica and restoring it to a known good (non-compromised) state [13], [38]. Since a replica becomes unavailable while undergoing proactive recovery, guaranteeing continuous availability in the presence of f compromises and k simultaneous proactive recoveries requires increasing the total number of replicas to $3f+2k+1$ [38].

Network Attacks: Recent work has considered the impact of *network attacks* on BFT replication [7], [24]. To withstand sophisticated network attacks that can target and isolate a site (e.g. [42], [21]), replicas must be deployed across at least three geographically distributed sites: otherwise, an attack that isolates a single site can disconnect a majority of replicas, preventing the system from processing requests (at least $2f+k+1$ correct connected replicas are needed in a system with $3f+2k+1$ total replicas) [7]. While multi-site architectures distributing replicas across three or more sites can guarantee both network-attack resilience and intrusion tolerance, they require a relatively large infrastructure investment: the work in [7] shows that 18 replicas distributed across three sites (or 12 replicas across four sites) are needed to simultaneously tolerate one compromise ($f=1$) and one site disconnection while supporting one proactive recovery. We aim to support this level of resilience, while minimizing complexity and cost for the system operator.

B. Separating Agreement from Execution

Our hybrid-management approach builds on work that separates agreement from execution by dividing replicas into an *agreement cluster*, which runs a BFT protocol to totally order client requests, and an *execution cluster*, which executes requests in the order determined by the agreement cluster and generates client responses [44]. This separation also makes it possible to insert a *privacy firewall* between the execution and agreement clusters to prevent compromised replicas from leaking confidential data [44], [17]. The privacy firewall filters messages to prevent confidential data from leaving the execution cluster. UpRight [14], Eve [22], HyperLedger Fabric [4], and Spider [18] also separate agreement from execution, and some recent BFT protocols separate dissemination of requests from ordering [23], [15], [39].

We similarly separate agreement from execution, but introduce the separation of *management*; our version of the

agreement cluster (BFT engine) is managed by the cloud service provider, while our execution cluster (application) is managed by the system operator. This introduces new privacy concerns, since system operators are often unwilling to expose confidential data, algorithms, or client information to the service provider managing the replication service. The work in [44] and [17] partially addresses privacy by encrypting client requests and replies and using a privacy firewall to filter messages. However, in those works, only the data itself is protected. Client identities, locations, and request patterns are not considered to be confidential, and clients communicate directly with the agreement cluster. In our model, the agreement cluster (BFT engine) runs in the cloud and should not have access to the system's clients, or any information about them.

In addition, we target a stronger threat model that includes network attacks and management domain failures. This changes the division of responsibilities between clusters by requiring cloud/agreement replicas to store (encrypted) state.

C. Cloud-Based BFT and Confidentiality

Prior work has used the cloud to reduce costs and/or simplify deployment for BFT systems. However, none of the existing works simultaneously (1) provide intrusion tolerance for arbitrary state machine replication applications (2) cleanly separate management of the BFT replication engine from the application, and (3) enforce confidentiality of application state, logic, and client IDs and request patterns.

Some prior work has investigated fully cloud-based BFT replication to make deployment easier (e.g. BFT-Dep [26]) or improve performance (e.g. Spider [18]), but these works assume the system operator is not concerned with confidentiality and is able to run the entire application in the cloud.

Other work has considered the confidentiality implications of cloud-based BFT systems, developing secret-sharing approaches to maintain confidentiality for BFT-replicated applications in the cloud (e.g. DepSpace[10], Belisarius [31], and COBRA [43]). These approaches have been used to build BFT-replicated storage systems, such as DepSky [8], SCFS [11], and RockFS [28]. However, these systems do not support arbitrary state machine replication applications. Practical secret-sharing schemes today support limited operations such as key-value storage [43], tuple space operations [10], or addition on stored values [31]. One possibility is to use a BFT storage solution to replicate encrypted state and consider the application as a client of the storage system, but in that case the application itself is not intrusion tolerant (and techniques like the ones we propose would be needed to change that). Alternatively, secure multiparty computation or homomorphic encryption could enable general operations on encrypted data, but these approaches are computationally expensive and, more fundamentally, do not keep the application logic confidential (only the data it operates on). Recent approaches using Trusted Execution Environments (TEEs) to provide confidential computing in the cloud via encrypted VMs [36] could keep both data and logic confidential. However, this requires additional assumptions on trusted hardware, and does not resolve the question of who is responsible for managing the encrypted VMs in the cloud.

To provide intrusion tolerance for general applications, while offloading part of the system management to a cloud provider, the work in [24] developed an architecture for partially cloud-based BFT systems. This approach is the closest to ours, in that management of the physical infrastructure is split between a system operator and an external service provider. The system operator manages on-premises sites, where they host replicas that communicate with clients, execute application logic, and participate in the BFT replication protocol. The service provider manages additional cloud sites, which host replicas that participate in the BFT replication protocol and store encrypted state. This approach supports multi-site network-attack-resilient configurations without requiring system operators to build and manage additional sites, and maintains confidentiality of application state, application logic, and client locations. But, it still exposes client IDs and request patterns to the cloud, and, even more importantly is an integrated system, where the assumption is that the system operator manages all of the software and simply uses cloud resources to avoid deploying additional physical sites.

III. Architecture Overview

Our architecture is designed to provide intrusion tolerance for any application that can work with the state machine replication model. Similar to [24], we consider *on-premises sites* that are managed by *system operators*, and *cloud sites* that are managed by *service providers*. Different from [24], we enforce a strict separation of responsibilities between the cloud and on-premises sites, as shown in Figure 1. The service provider is responsible for deploying and managing the *BFT Replication Engine* which runs *only* in the cloud. The BFT Replication Engine consists of a set of *cloud replicas (CRs)* running a BFT replication protocol (any one of the many

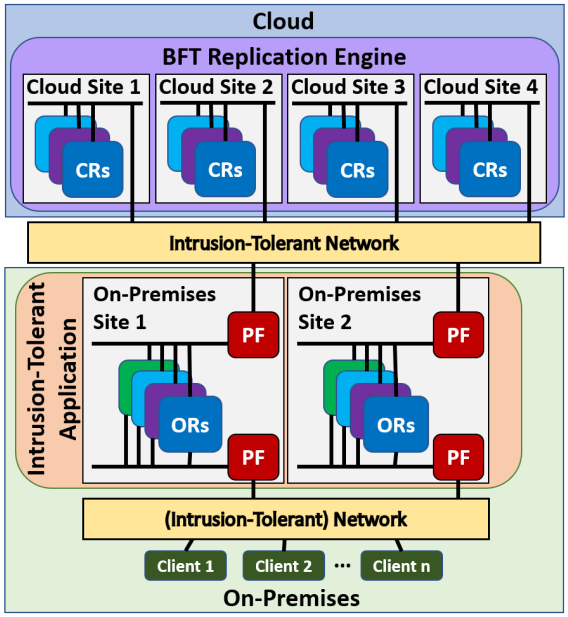


Fig. 1: Decoupled Intrusion-Tolerant Architecture

existing BFT protocols can be used). It provides an **intrusion-tolerant ordering and encrypted state maintenance service** that is used to turn the application, which runs only in the on-premises site(s), into an *Intrusion-Tolerant Application*. Note that the service provider running the BFT replication engine may be an existing cloud provider, or, we envision that our new architecture can allow a specialized intrusion-tolerance-as-a-service provider to emerge. The service provider can scale their service effectively by using the same infrastructure and BFT Replication Engine to support many applications.

The system operator is responsible for deploying and managing the Intrusion-Tolerant Application which consists of a set of *on-premises replicas (ORs)*. On-premises replicas accept incoming requests from *clients* and forward them to the BFT Replication Engine for ordering. The cloud replicas establish a total order on the requests, and the on-premises replicas execute requests according to this total order and return responses to clients.

On-Premises Domain: In our architecture, we consider clients and on-premises replicas to be part of the same management domain, and clients communicate only with the on-premises replicas. This is a good fit for many industrial control or enterprise applications where the on-premises replicas and the clients are managed by the same entity. For example, in an industrial control context, on-premises replicas may be replicas of a Supervisory Control and Data Acquisition (SCADA) server, and clients may be Remote Terminal Units or Programmable Logic Controllers that send data to and receive commands from the SCADA system. Or, we could consider replicas of an Electronic Health Record (EHR) database, and clients that are authorized devices or users accessing the service via a VPN. Clients authenticate themselves to the system by signing their requests using private keys; the corresponding public keys are known by the on-premises replicas.

Cloud Domain: The cloud replicas represent a separate management domain, managed by the cloud service provider. To preserve confidentiality, on-premises replicas encrypt each client request (including client IDs and sequence numbers) before forwarding it to the cloud replicas for ordering.

In addition to performing ordering, the BFT Replication Engine stores each ordered encrypted request. It periodically garbage collects these requests, replacing them with encrypted state checkpoints from the Intrusion-Tolerant Application. This enables on-premises replicas to recover their state entirely from the cloud, making it possible to tolerate sophisticated network attacks, as well as *management domain failures* in which *all* on-premises replicas lose their state (see Section IV).

Simplifying Interfaces via Threshold Signatures: To make the interface between the cloud and on-premises domains as simple as possible and avoid requiring either domain to know internal configurations of the other, we use threshold signatures. In our architecture, *all* messages between domains must be threshold-signed. With this approach, the cloud replicas only know a single public key to authenticate messages sent by on-premises replicas, and the on-premises replicas similarly only need to know a single public key to authenticate messages sent by the cloud replicas.

We use an $(f_o{+}1,n_o)$-threshold scheme for each on-premises site, where $f_o{+}1$ shares out of n_o total shares are needed to generate a valid signature (where n_o is the number of on-premises replicas per site). Thus, a valid threshold signature guarantees that at least one correct on-premises replica agreed to the content of the message. Key shares can be refreshed without changing the public service key [45], [35]. Similarly, messages sent from the cloud replicas to the on-premises replicas are signed using an $(f_c{+}1,n_c)$-threshold scheme (where n_c is the total number of cloud replicas).

Strengthening Confidentiality via Privacy Firewalls: We envision two separate networks: one connecting the clients with the on-premises sites, and the other connecting the cloud sites with the on-premises sites. Ideally, each on-premises replica connects to each of these networks using separate network interfaces, with another interface used for local-area communication with the other replicas in its site. To ensure that confidential data does not leave an on-premises site, even in the presence of a compromised on-premises replica, we can insert privacy firewalls (PFs) [44], [17] between the on-premises replicas and each of the two wide-area networks. The *cloud-side PF* filters messages sent from on-premises replicas to the cloud, and the *client-side PF* filters messages sent from the on-premises replicas to the clients. Since all messages that leave an on-premises site must be threshold signed, the privacy firewall implementation is simple: each privacy firewall knows the relevant public key and only forwards outgoing messages that have a valid threshold signature.

We consider each privacy firewall to be a black box, which can be a complex configuration from prior work (e.g. [44], [17]) or a single node. Note that while a single-node privacy firewall may appear to be a single point of failure, since we consider a wide-area setting, availability already depends on the router for the site, and a privacy firewall could be integrated with the site router, so it does not meaningfully expand the system's attack surface (see Section IV for details).[1]

IV. Threat Model

Our threat model tolerates a configurable number of *server compromises* and *site disconnections*, similar to [7]. In a server compromise, the attacker gains control of a server (on-premises or cloud) and can cause it to behave arbitrarily (Byzantine failure). In a site disconnection, an attacker disconnects a site (on-premises or cloud) from the network and prevents it from communicating with replicas or clients located in other sites (e.g. via a denial of service attack).

In order to force an attacker to exceed the tolerated number of server compromises within a limited time window, we support proactive recovery [38], [13]. During proactive

[1] If weaker confidentiality guarantees are acceptable, the privacy firewall can be left out of the architecture. In that case, our confidentiality guarantees are the same as in [24]. In our specification, privacy firewalls forward incoming messages to replicas within a site, but we can remove them by having on-premises replicas directly join the relevant multicast group. In our implementation, we use Spines [40] for inter-site multicast.

recovery, a replica becomes temporarily unavailable until its recovery is completed. To support the assumption that the attacker does not exceed the tolerated threshold, the system must employ diversity (e.g. using N-version programming [5], [25], OS diversity [19], compile-time diversification [32], and/or different hardware and ISPs). Note that while the system operator must still employ diversity for the on-premises replicas, our architecture significantly reduces their required number of diverse variants; see Section VIII for analysis.

Due to the full separation of the Intrusion-Tolerant Application and the BFT Replication Engine, the number of tolerated server compromises and site disconnections, as well as the number of supported proactive recoveries, can be configured separately for each of them.

On-premises Threat Model: Simultaneously, in each on-premises site, f_o on-premises replicas can be compromised, and k_o on-premises replicas can be performing proactive recovery. At the same time, d_o on-premises sites can be disconnected.

Cloud Threat Model: Simultaneously, f_c cloud replicas can be compromised, k_c cloud replicas can be performing proactive recovery, and d_c cloud sites can be disconnected.

Management Domain Failures: In addition to the basic threat model, our new hybrid management model makes it possible to recover from a *management domain failure*, in which *all* on-premises replicas managed by a system operator lose their state. This threat model can capture relevant practical attacks such as ransomware: if a ransomware attack on the on-premises replicas causes them all to lose access to their state (because it has been maliciously encrypted), the replicas can be taken down, cleaned, restarted, and the latest state restored from the cloud replicas. The system will suffer an outage during the recovery, but can seamlessly resume operations without losing the results of any previously executed requests.[2]

We do not tolerate cloud domain failures, as recovering cloud replicas may not be able to establish the latest ordinal executed by an on-premises replica (e.g. only a single replica has executed the ordinal, but it then becomes unreachable). However, highly resilient systems can still be built by ensuring sufficient resilience of the cloud domain. Cloud providers can deploy replicas across multiple data centers and specify the risk of simultaneous unavailability in their SLA. A specialized intrusion-tolerance-as-a-service provider can increase management diversity by deploying replicas across infrastructure managed by different underlying cloud providers. This type of cloud-of-clouds approach is also considered in other works [8], [11], which explicitly store data across multiple cloud providers (e.g. Amazon, Azure, etc).

Service Properties: Our safety and liveness guarantees are similar to other BFT works, although we adapt them to account for the separation between on-premises and cloud replicas. We also guarantee confidentiality of the application data. Specifically, we adopt the definitions for safety and confidentiality from [24], and the definition for liveness from [13].

Definition 1 (Safety)**.** *If two correct on-premises replicas execute the i^{th} update, then those updates are identical, and the state resulting from the execution of that update at the two on-premises replicas is also identical. [24]*

Our system guarantees safety as long as no more than f_o on-premises replicas in each on-premises site and no more than f_c total cloud replicas are compromised simultaneously (once a compromised replica goes through proactive recovery and is restored to a correct state, it is no longer compromised). Note that as in [24], the safety definition only includes on-premises replicas, as cloud replicas do not execute updates.

In general, we assume that correct (non-compromised) replicas follow the protocol correctly and do not lose their state. However, we also maintain safety in the case of an on-premises management domain failure (i.e. all on-premises replicas lose their state), as long as no more than f_c cloud replicas are compromised (and no correct cloud replicas lose their state). Compromised privacy firewalls cannot affect safety.

Definition 2 (Liveness)**.** *Clients eventually receive replies to their requests [13].*

To guarantee liveness, we require that the conditions of both the on-premises and cloud threat models are met: at most f_o on-premises replicas per on-premises site and f_c total cloud replicas are compromised, at most k_o on-premises replicas per on-premises site and k_c cloud replicas are undergoing proactive recovery, and at most d_o on-premises sites and d_c cloud sites are disconnected from the network. We also require that the privacy firewalls are up and correct (note that we can tolerate failed or compromised privacy firewalls, but, since a failed/compromised firewall effectively disconnects its site from the network, such failures count against the d_o tolerated on-premises disconnections).

Liveness also requires that correct system components that are not in the disconnected sites are able to communicate successfully. Specifically, we require that all correct on-premises replicas in a given on-premises site are able to communicate with each other and with the privacy firewalls for that site; all cloud replicas are able to communicate with all other correct, not-disconnected cloud replicas and with the cloud-side privacy firewalls for the not-disconnected on-premises sites; and the client-side privacy firewall in each on-premises site is able to communicate with clients. Communication between the cloud replicas must meet any network synchrony requirements of the specific BFT protocol being used.

Definition 3 (Complete Confidentiality)**.** *System state and state manipulation algorithms remain confidential (known only to on-premises replicas) [24].*

We guarantee complete confidentiality even if an unlimited number of cloud replicas are compromised. We maintain this guarantee when up to f_o on-premises replicas per on-premises site are compromised, as long as the privacy firewalls are up

[2]Even if the attack resulted from executing one of the updates, since the cloud replicas do not execute updates, they are not affected by the same attack. Thus, the operator could recover on-premises replicas to the latest checkpoint, and manually omit the update that triggered the attack.

and correct. If a privacy firewall and an on-premises replica in the same site are compromised, confidentiality can be violated.

Note that when an on-premises replica is compromised, it may be able to use side-channel attacks to potentially reveal confidential information, but, in contrast to all prior work, our architecture can *only* leak information if such attacks are successful. The work in [24] cannot maintain complete confidentiality in the presence of compromised on-premises replicas; solutions using privacy firewalls [44], [17] provide stronger guarantees for confidential state, but inherently reveal client information to agreement nodes; and in secret sharing based solutions [31], [43], replicas similarly communicate directly with clients, and execute state manipulation algorithms. We consider side-channel attacks outside the scope of this paper. However, the privacy firewall can provide mechanisms to make them more difficult (as discussed in [44]).

V. System Configuration

Our decoupled system architecture can be configured based on the threat model the operator wants to tolerate. For the on-premises threat model, we require the total number of on-premises sites $S_o \geq d_o+1$, and the number of replicas in each on-premises site $n_o \geq 2f_o+k_o+1$. This guarantees that at least one site with f_o+1 correct replicas is always available, which is the minimum needed to generate valid threshold signatures.

The required number of cloud sites and replicas depends on the BFT protocol used, but for protocols that normally use $3f+2k+1$ replicas to withstand f compromises and k proactive recoveries (the most common setting), we adapt the replica distribution formula from [7]. We require the total number of cloud sites $S_c \geq 2d_c+1$ and set the total number of cloud replicas $n_c = 3f_c + 2\left\lceil \frac{3f_c d_c + d_c + S_c k_c}{S_c - 2d_c} \right\rceil + 1$, with replicas distributed evenly across the sites. This guarantees that a quorum of cloud replicas is always available under our threat model. Note however, that if the n_c replicas do not divide evenly among the cloud sites, additional replicas may be needed (see Appendix for details and derivation).

Figure 1 shows a configuration with 4 on-premises replicas in each of 2 sites and 12 cloud replicas distributed across 4 sites. This configuration tolerates one compromise, one disconnection, and one proactive recovery in each of the on-premises and cloud threat models ($f_o = f_c = d_o = d_c = k_o = k_c = 1$). To make site disconnections more difficult to execute successfully, we use an *intrusion-tolerant network* that uses an overlay approach to connect sites with redundancy [29], [7]. However, we make this optional for the on-premises network.

For system operators who currently use a primary and a backup site for fault tolerance, the configuration in Figure 1 offers the full resilience benefits of our architecture with minimal additional infrastructure. Operators only need to contract with a cloud provider and, assuming they have a primary and backup server in each site already, add two servers to each site. However, other options are possible. For example, if cloud sites have very high availability, we can consider a configuration with $d_c=0$ that does not tolerate cloud site disconnections, but only requires one cloud site and may be useful for latency sensitive applications (see Section VIII). Similarly, a configuration with $d_o=0$ does not tolerate on-premises site disconnections, but allows a system operator currently using a single site to gain intrusion tolerance and the ability to recover from management domain failures without constructing and managing any additional sites.

VI. Protocols for Hybrid Management of BFT Systems

In order to support the system architecture described in Section III, we must develop new protocols for handling client requests, and for performing state transfer and recovery.

A. Introducing New Client Requests

Figure 2 illustrates the steps involved in processing each client request. First, a client signs its request with its private key and sends it to the on-premises sites (step 1, Figure 2). If a client does not have the capability to do this on their own, a proxy can sign the request on the client's behalf [7], [27]. Once the request reaches an on-premises site, it is received by the client-side privacy firewall, which multicasts the request to all on-premises replicas in that site (step 2, Figure 2).

To enforce confidentiality of client requests, on-premises replicas encrypt the request using shared symmetric keys known to all on-premises replicas (but not known to the cloud replicas). In contrast to prior works that encrypt client requests (e.g. [44], [17], [24]), the on-premises replicas encrypt not only the request body, but also the client headers so that the cloud replicas do not see client IDs and sequence numbers, and cannot easily learn client request patterns.

Next, on-premises replicas generate a threshold signature. Each replica encrypts the request, generates a partial signature share over it, and multicasts the signature share to the other replicas in its site. Upon collecting f_o+1 partial signatures, the replica combines the partial signatures to generate a full threshold signature and sends the signed, encrypted request to the cloud-side privacy firewall (step 3, Figure 2).

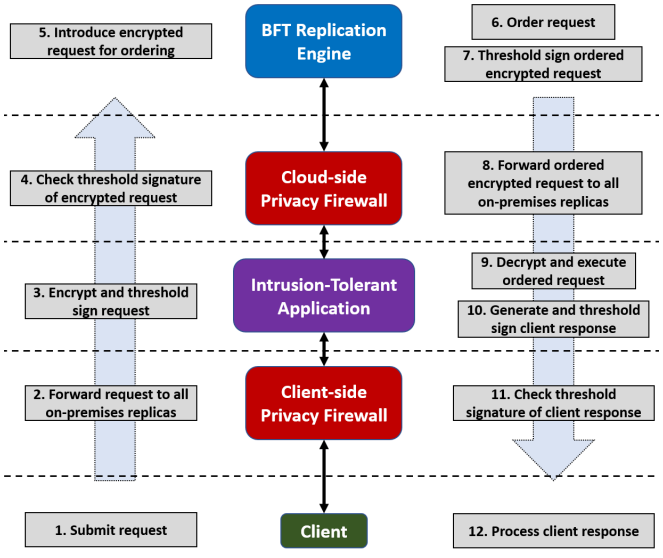


Fig. 2: Client Request Flow

Note that to generate a full threshold signature, all on-premises replicas must generate partial signature shares over *identical encrypted content*. To ensure this, similar to the work in [24], each on-premises replica maintains two shared secret keys known to all on-premises replicas: one for symmetric encryption/decryption of client requests, and the other for generating initialization vectors via a pseudorandom function, similar to the approach in [17]. This allows all replicas to generate identical encrypted requests for a given client request. To avoid signing each request individually, it is possible to batch requests (see Appendix for details.)

After verifying the threshold signature, the privacy firewall multicasts the signed encrypted request to the cloud replicas (step 4, Figure 2). Cloud replicas use the threshold signature to verify that the encrypted request is valid before introducing it for ordering by the BFT replication engine.

While the high-level process is relatively straightforward, fully encrypting and threshold signing client requests introduces new challenges: (1) we need to address a replay-attack vulnerability that encrypting client requests creates, and (2) we need a procedure for consistently updating encryption keys.

1) Preventing Replay Attacks: While encrypting the full client request (including headers) and threshold signing it enforces stronger confidentiality than previous approaches, this introduces a new vulnerability to *replay attacks*. A compromised on-premises replica can overload the BFT Replication Engine by storing threshold-signed encrypted requests and replaying them to the cloud replicas repeatedly. Since these *old* requests have valid threshold signatures, they will successfully pass through the privacy firewall and will be accepted as valid by the cloud replicas, causing them to waste processing resources and bandwidth ordering the duplicate requests.

Prior works have avoided this issue by using cleartext client IDs and sequence numbers in request headers to reject old/duplicate requests [44], [17], [24], but this exposes client information to the cloud replicas. A naive solution is for the cloud replicas to store a copy of every encrypted client request (e.g. in a hash table), and then use that to check for duplicates (when a new request arrives), which are then discarded. However, this is not practical as it requires unbounded memory (cloud replicas can never garbage collect old requests).

Validity Period. To address replays, we require on-premises replicas to append a *validity period* in cleartext to each encrypted request. The validity period is part of the content over which the threshold signature is generated (so a compromised replica cannot modify it). Cloud replicas store unique requests from the current validity period in a hash table, so they can discard requests from previous validity periods and duplicates from the current validity period.

On-premises replicas determine the validity period based on the latest global sequence number ($lseq$) they have executed, and cloud replicas reject a request if the upper bound ($ubound$) of its associated validity period is less than the $lseq$ they have totally ordered. So that all on-premises replicas will typically have the same view of the validity period, we only update the validity period every vp_size sequence numbers. The lower bound ($lbound$) of the validity period is set as: $\lfloor \frac{lseq}{vp\_size} \rfloor \times vp\_size$, and the $ubound$ is set as: $lbound + (2 \times vp\_size)$. Ideally, the validity period size (vp_size) should be at least the maximum number of outstanding client requests. Smaller validity periods will not violate correctness, but can reduce performance, because on-premises replicas may assign validity periods that become stale by the time the request reaches the cloud replicas. With each client limited to one outstanding request, we can set vp_size to at least $M \times \lfloor \frac{n_o}{f_o+1} \rfloor \times S_o$ (where M is maximum number of clients, n_o is number of on-premises replicas per site, and S_o is number of on-premises sites). Note that there can still be brief periods where on-premises replicas disagree on the validity period (because they execute requests at slightly different times). However, this does not affect liveness, since a replica will retransmit its partial signature share for a client request (with an updated validity period) if the request is not executed before a timeout.

2) Updating Encryption Keys: Privacy firewalls prevent encryption key leakage, but periodic key refreshes are still needed to be able to recover from potential side-channel attacks or a malicious operator with physical access copying keys. To ensure that all on-premises replicas know which key to use to decrypt each request, we tie key changes to the validity period (since it is the only available cleartext information). When the end of the validity period approaches, each on-premises replica generates a new key proposal signed by a persistent hardware-based key (e.g. using the TPM) and submits it for ordering. The key for the next validity period is determined based on the ordered key proposals, similar to [24] (although they based the key-change interval on client sequence numbers that are not available in cleartext for us).

B. Ordering and Executing Client Requests

Upon receiving a new valid client request (i.e. one that is within the validity period, not a duplicate, and has a valid threshold signature), cloud replicas inject the request into the BFT replication engine (step 5, Figure 2). This executes the BFT agreement protocol to assign the request a global sequence number or *ordinal* (step 6, Figure 2). Cloud replicas threshold-sign the ordered encrypted request (step 7, Figure 2), and then multicast it to the cloud-side privacy firewalls, which forward it to the on-premises replicas (step 8, Figure 2).

Requests may occasionally arrive out-of-order due to network disruptions, so on-premises replicas use a sliding window buffer to maintain ordering. Upon receiving the next expected ordinal, the on-premises replica executes the corresponding request (step 9, Figure 2), generates a response, and cooperates to create a threshold signature (step 10, Figure 2). This response is sent to the client through the client-side privacy firewall (step 11, Figure 2). The client (or accompanying proxy) validates the correctness of the response by verifying the threshold signature (step 12, Figure 2).

C. Checkpoints and Nearest-First Recovery

As discussed in Section III, to enable recovery from network attacks and management domain failures, cloud replicas store

encrypted state checkpoints and any encrypted requests that have been ordered since the latest checkpoint. On-premises replicas can request this encrypted state to recover from state loss or prolonged disconnections.

However, as also discussed in Section III, *every* message leaving an on-premises site must be threshold signed. This requires new protocols for checkpointing and recovery across management domains. Two important challenges are: (1) Only threshold-signed recovery requests can be sent from on-premises replicas to the cloud replicas. This requires on-premises replicas to agree on which requests to send to the cloud. (2) Because client responses must also be threshold-signed, on-premises replicas must be able to recover these signatures in order to serve client retransmissions.

We address these issues with a new nearest-first recovery protocol that first attempts to perform recovery within a site, and only sends (threshold-signed) recovery requests outside the site if in-site recovery is unsuccessful. The protocol also enables on-premises replicas to collect the threshold-signed client response corresponding to each request it recovers. This strategy is necessary under our system model, but also improves recovery latency and wide-area bandwidth usage by localizing state transfer as much as possible. Below we describe the checkpointing and recovery protocols.

1) Checkpoint Creation: Each on-premises replica periodically generates a checkpoint representing its current state (including the latest response for each client), encrypts it, and then cooperates with other on-premises replicas in its site to create a threshold signature. The replica stores the signed encrypted checkpoint and multicasts it to the cloud replicas, which can verify the threshold signature and then store the encrypted checkpoint. Any replica can safely remove ordered encrypted requests older than the currently stored checkpoint.

2) Nearest-First Recovery: Recovery begins when a replica detects that it is missing ordered requests, e.g. due to a site disconnection, crash, or proactive recovery.

Requesting Recovery. An on-premises replica triggers recovery when it receives an ordered request beyond the upper bound of its sliding window buffer from the cloud replicas. The *recovering replica* separately requests missing ordered encrypted requests and client responses. For each, it sends a request with a list of ordinals (global sequence numbers) that it is missing to the other on-premises replicas in its site.

Responding to a Recovery Request. Upon receiving a recovery request, an on-premises replica will respond with its stored encrypted checkpoint if any ordinal in the recovery request is older than the checkpoint. Otherwise, the replica sends all of the requested ordered encrypted requests or client responses it has to the recovering replica. It also sends the associated threshold signatures for client responses; if it does not yet have the threshold signature for a client response, it sends its partial signature share instead. To prevent malicious replicas from wasting resources, all replicas rate-limit their responses to repeated recovery requests from the same replica.

Applying Recovery Responses. Upon receiving client responses, a recovering replica simply stores them. Upon receiving a checkpoint that is newer than its current state, the recovering replica verifies the threshold signature, and decrypts and applies the checkpoint to its local state (the latest client responses in the checkpoint are extracted and stored).

Upon receiving new ordered encrypted requests, the recovering replica decrypts and executes them (after verifying their threshold signatures) consecutively based on the ordinals. If it already collected a client response with threshold signature for the executed ordered request, then it simply stores this client response and moves on to the next ordinal. Otherwise, the recovering replica generates the client response, sends a partial signature to other on-premises replicas in its site, and waits to collect f_o+1 partial signature shares (including its own). By applying a checkpoint and/or executing requests, the recovering replica eventually catches up to the latest state.

Recovering an On-Premises Site. If all the replicas in a site are missing the *same* ordered encrypted requests (which they will eventually find out), then they can generate a threshold signature for the recovery request and send it to the cloud replicas. To do this, every recovery request for ordered encrypted requests includes a validity period and a partial signature share over the request; the validity period is based on the ordinal of the latest ordered encrypted request received from the cloud replicas. On receiving such a request, a cloud replica multicasts the requested ordered encrypted requests or encrypted checkpoint to the on-premises replicas.

To mitigate replay attacks by a malicious replica (which may re-send an old threshold-signed recovery request), cloud replicas rate-limit their responses to recovery requests. New recovery requests are not subject to this rate limit. If a cloud replica receives a request that is in the current validity period, and is not a duplicate (checked with hashes of other requests in the current validity period), then it responds immediately.

Recovery Procedure in Cloud: Cloud replicas use a similar nearest first recovery strategy when they detect a gap in the global sequence numbers from the underlying BFT protocol. However, they do not store or request client responses.

VII. Implementation

We have implemented Decoupled Spire, a SCADA system for the power grid, based on the open source Spire version 1.2 [41]. Spire 1.2 uses an integrated architecture in which all replicas fully participate in the BFT replication protocol, maintain application state, and execute requests [7].

Decoupled Spire Components. In Decoupled Spire, the on-premises sites host replicas of the SCADA master application. The clients are Programmable Logic Controllers (PLCs), Remote Terminal Units (RTUs), and Human Machine Interfaces (HMIs); we use the HMI and emulated PLCs/RTUs available in Spire 1.2. Like Spire 1.2, we use Prime [3], [34] as the BFT replication engine. We add a single-node *privacy firewall* to these components. Privacy firewalls run in each on-premises site and use the public service key for that site to verify threshold signatures on all outgoing messages. We use Spines [40] for our intrusion-tolerant networks: one Spines network connects all the cloud replicas to each other and

	Avg Latency	% <100ms	0.1^{st} percentile	1^{st} percentile	50^{th} percentile	99^{th} percentile	99.9^{th} percentile
Decoupled Spire (f_o=1, f_c=1, d_o=0, d_c=0)	41.8 ms	100	27.9 ms	30.1 ms	41.8 ms	53.1 ms	54.8 ms
Decoupled Spire (f_o=1, f_c=1, d_o=1, d_c=0)	41.9 ms	100	27.7 ms	30.1 ms	41.9 ms	53.1 ms	54.9 ms
Decoupled Spire (f_o=1, f_c=1, d_o=0, d_c=1)	58.7 ms	100	46.6 ms	48.0 ms	58.7 ms	69.5 ms	71.3 ms
Decoupled Spire (f_o=1, f_c=1, d_o=1, d_c=1)	58.9 ms	100	46.9 ms	48.1 ms	59.0 ms	69.6 ms	71.3 ms
Confidential Spire 2021 [24] (f=1)	50.1 ms	100	38.4 ms	39.6 ms	50.1 ms	60.9 ms	63.5 ms
Spire 2018 [7] (f=1)	49.9 ms	100	38.2 ms	39.2 ms	50.0 ms	60.5 ms	62.2 ms
Decoupled Spire (f_o=2, f_c=1, d_o=1, d_c=1)	58.9 ms	100	46.9 ms	48.2 ms	58.9 ms	69.7 ms	71.6 ms
Decoupled Spire (f_o=1, f_c=2, d_o=1, d_c=1)	60.5 ms	100	48.5 ms	49.5 ms	60.5 ms	71.7 ms	75.2 ms
Decoupled Spire (f_o=2, f_c=2, d_o=1, d_c=1)	62.0 ms	100	49.4 ms	50.6 ms	62.0 ms	74.3 ms	78.2 ms
Confidential Spire 2021 [24] (f=2)	56.5 ms	100	42.2 ms	43.7 ms	56.7 ms	69.8 ms	73.8 ms
Spire 2018 [7] (f=2)	53.4 ms	100	39.6 ms	41.1 ms	53.5 ms	64.1 ms	67.9 ms

TABLE I: Normal Operation Performance on LAN with emulated latencies between sites for 36000 updates over 1 hour

the cloud-side privacy firewalls, and a second Spines network connects the clients to the client-side privacy firewalls. Inside each on-premises site, replicas communicate using UDP over a switched LAN (with application-level retransmissions).

Separating Agreement and Execution. In contrast to Spire 1.2, our SCADA master replicas do not participate in the Prime replication protocol. Instead, each SCADA master is linked with a simple intrusion-tolerance layer that *prepares* each client request for ordering, sends it to the cloud replicas, and then receives, buffers, and verifies signatures on incoming ordered updates. Preparing a client request for ordering involves encrypting it, appending a validity period, and generating a threshold signature on it. Our implementation of encryption is similar to that in [24], which is based on [17]. We use the client request and a pseudo-random function key (refreshed each validity period and shared by all on-premises replicas, as discussed in Section VI-A2) to generate a hash-based message authentication code (HMAC). This HMAC is used as the initialization vector (IV), along with the shared encryption key, to encrypt the *entire* client request using AES-256 in CBC mode. This encrypted request is accompanied by a clear-text header that includes the validity period and IV, and a threshold signature covering the header and the encrypted request.

VIII. Evaluation

The main benefit of Decoupled Spire is its clean separation of the cloud and on-premises domains, which simplifies management while supporting a strong threat model. Here, we quantify the performance overhead of this separation (the main tradeoff for system operators) by comparing Decoupled Spire with Spire 1.2 [7] and Confidential Spire [24]. We show that this tradeoff is acceptable for this latency-sensitive application.

All experiments are done using a local area network with emulated latencies between sites that reflect a realistic geographic distribution that spans 250 miles of the US East Coast (similar to [7]). This corresponds to emulated latencies of 2 to 5 ms between each pair of sites. We emulate ten power grid substations, where each introduces a new request every second for a total of 36,000 requests in a one hour period.

Normal Operation Performance (f=1). Table I shows client request latencies over a one-hour experiment for each configuration. We can see that Decoupled Spire (f_o=1,f_c=1,d_o=1,d_c=1), which tolerates one compromise, one proactive recovery and one site disconnection in both on-premises and cloud domains, has an average latency of 58.9ms, compared to about 50ms for Spire 1.2 (f=1) and Confidential Spire (f=1)[3], for an overhead of about 9ms (18%). The overhead comes from the additional wide-area communications on the critical path of request processing: in Decoupled Spire, on-premises replicas must send each client request to the cloud replicas before it is introduced for ordering, whereas, in Spire and Confidential Spire, the control center replicas directly inject requests via their local BFT replication instances. In our experimental setting, this extra wide-area delay adds a total of 4ms to the processing of each request. The remaining 5ms is due to the additional processing done by the on-premises and cloud replicas (more messages, encryption, decryption, duplicate check, and threshold signing and verification).

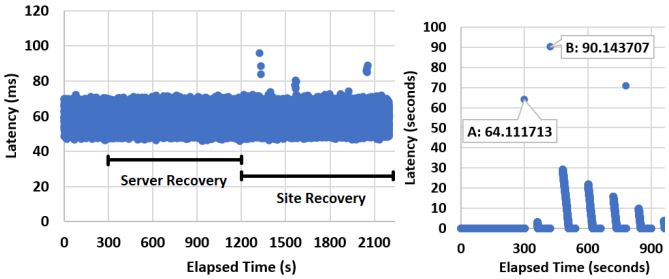
Decoupled Spire (f_o=1,f_c=1,d_o=0,d_c=1), which does not tolerate any on-premises site disconnection, has almost exactly the same overhead as Decoupled Spire (f_o=1,f_c=1,d_o=1,d_c=1), since it has the same additional wide-area communications and processing in the critical path.

Interestingly, Decoupled Spire (f_o=1,f_c=1,d_o=1,d_c=0), which does not tolerate any cloud site disconnection, achieves about 8ms (16%) *less* latency compared to Spire 1.2 (f=1) and Confidential Spire (f=1), and 17ms (29%) less latency compared to Decoupled Spire (f_o=1,f_c=1,d_o=1,d_c=1). This is because the entire agreement protocol runs in a single cloud site: even with the additional wide-area delays for sending the request to the cloud and back, eliminating wide-area communication overhead in the agreement protocol results in lower total latency. For latency-sensitive applications, this may be attractive if service providers can guarantee near 100% uptime for their cloud sites and effectively bolster them against DoS attacks. Decoupled Spire (f_o=1,f_c=1,d_o=0,d_c=0), which does not tolerate any site disconnection, has similar performance, since it also uses just one cloud site.

Increasing the Number of Tolerated Intrusions. In Decoupled Spire, increasing f_o from 1 to 2 (lower half of Table I) has negligible effect on latency, since most of the additional communication happens inside the on-premises site over a LAN. This can be useful for system operators who want to increase the resiliency in their on-premises sites (which may be less protected) without needing any higher resiliency in the cloud (which may be better protected).

Table I also shows that Decoupled Spire with f_o=f_c=2 increases latency by about 3.1ms compared to Decoupled Spire with f_o=f_c=1. Interestingly, Spire 1.2 also adds about 3.3ms

[3]In our experiments, the overhead of Confidential Spire compared to Spire 1.2 is smaller than the one reported in [24], likely due to hardware differences.



(a) Server and Site Recovery (b) Domain Recovery

Fig. 3: Performance During Attack Recovery

when increasing f from 1 to 2. This is because most of the increase comes from the increased wide-area communications in the BFT agreement protocol which is about the same for both systems (both increase the number of BFT replicas from 12 to 19). Confidential Spire shows a larger increase (6.4ms) when f is increased from 1 to 2, as the number of replicas in the BFT protocol increases from 14 to 21.

Performance during Failures and Recovery. We evaluated the performance of Decoupled Spire (f_o=1,f_c=1,d_o=1,d_c=1) while recovering after a failure or an attack. We emulated this in 1-hour long experiments by repeatedly killing and restarting the SCADA application of a single server (for server recovery), all servers in an on-premises site (for site recovery), or all servers in both on-premises sites (for domain recovery). For each experiment, we kill the SCADA application, wait 1 minute, restart the SCADA application, again wait 1 minute, then repeat. A restarted server must collect the latest state from other replicas, so it emulates the proactive recovery process after state has been corrupted by an attacker.

Our SCADA application requires client request latencies within a 100ms threshold under normal operations, and can tolerate up to 200ms for a few requests [1]. Figure 3a shows that our Decoupled Spire achieves just that since no request crosses 100ms during server or site recoveries. This is because during server recovery, the entire recovery process happens within the on-premises site, while the path for processing client requests through the other on-premises site remains unaffected. We see a few small spikes in latencies (but none over 100ms) during site recovery, since the recovering replicas need to pull the latest encrypted checkpoint and encrypted ordered requests from the cloud replicas which generates extra wide-area network traffic that affects request latencies.

During a domain recovery, we see large latency spikes while both on-premises sites are recovering, but the performance quickly returns to normal as soon as either on-premises site finishes recovering. Figure 3b shows a few requests with very high latencies (e.g. 64s for point A and 90s for point B). This is because those requests were submitted just before both the on-premises sites went down, and hence were ordered and held by the cloud replicas until the on-premises sites recovered.

Discussion on Throughput. In our current implementation, the throughput of the system is primarily limited by the BFT replication engine (in our case, Prime [3], [34]), so we do not focus on throughput in the evaluation. However, there are three additional factors in our architecture that can impact throughput: threshold signing for new client requests at on-premises replicas, threshold signing ordered encrypted updates at cloud replicas, and threshold signing client responses. Of these, the threshold signing of ordered encrypted updates is the only new addition compared to Confidential Spire [24].

	Number of Diverse Variants		
	App	BFT	On-Premises (App/BFT)
Decoupled Spire (f_o=1, f_c=1, d_o=1, d_c=1)	4	12	4 / 0
Confidential Spire 2021 [24] (f=1)	8	14	8 / 8
Spire 2018 [7] (f=1)	12	12	6 / 6
Decoupled Spire (f_o=2, f_c=2, d_o=1, d_c=1)	6	19	6 / 0
Confidential Spire 2021 [24] (f=2)	12	21	12 / 12
Spire 2018 [7] (f=2)	19	19	10 / 10

TABLE II: Diversity Analysis (number of diverse variants)

Diversity Analysis. To support failure independence assumptions, each instance of the application, and each instance of the BFT engine should be different from all other instances. Decoupled Spire significantly reduces the number of diverse variants required for the application. Because we tolerate f_o compromises per on-premises site, a system operator can set up diverse replicas in one site, and then simply duplicate that same setup for their additional site(s). In contrast, Confidential Spire requires that all application replicas are diverse, and Spire runs an application instance at *every* replica, requiring a much larger number of diverse variants. While we require about the same number of variants for the BFT replication engine as previous systems, the service provider approach has an important benefit here: the service provider can use the same set of (diverse) replicas to serve many applications, which can help make it cost-effective to invest in expensive diversity techniques (e.g. N-version programming).

Table II summarizes the number of required diverse application and BFT engine variants in each architecture, as well as how many of those variants need to be deployed on-premises. For example, for the case of f=1, we require only 4 diverse application variants compared to 8 for Confidential Spire or 12 for Spire. The total number of diverse components that need to be deployed on-premises is dramatically reduced, since both Spire and Confidential Spire require BFT replicas (with diverse variants) to be deployed on-premises.

IX. Conclusion

We have presented a decoupled cloud-based BFT architecture that allows system operators to deploy and manage intrusion-tolerant applications, while completely offloading the BFT replication protocol to a cloud service provider. It does this while preserving the confidentiality of application state, algorithms, and client request patterns even if a threshold number of on-premises replicas, and any number of cloud replicas in the cloud sites are compromised. It supports a broad threat model including server compromises, network attacks, and management domain failures. We implemented and evaluated the system and showed that it introduces a latency overhead of only about 9ms (18%) compared to an integrated system-operator-managed BFT architecture.

REFERENCES

[1] IEEE standard communication delivery time performance requirements for electric power substation automation. *IEEE Std 1646-2004*, pages 1–24, 2005.

[2] The growing threat of ransomware attacks on hospitals. https://www.aamc.org/news-insights/growing-threat-ransomware-attacks-hospitals, July 2021. Retrieved July 28, 2023.

[3] Yair Amir, Brian Coan, Jonathan Kirsch, and John Lane. Prime: Byzantine replication under attack. *IEEE Transactions on Dependable and Secure Computing*, 8(4):564–577, July 2011.

[4] 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.

[5] Algirdas Avizienis. The n-version approach to fault-tolerant software. *IEEE Transactions on Software Engineering*, (12):1491–1501, 1985.

[6] Amy Babay, John Schultz, Thomas Tantillo, and Yair Amir. Toward an intrusion-tolerant power grid: Challenges and opportunities. In *IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 1321–1326, Vienna, Austria, July 2018.

[7] Amy Babay, Thomas Tantillo, Trevor Aron, Marco Platania, and Yair Amir. Network-attack-resilient intrusion-tolerant SCADA for the power grid. In *48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 255–266, Luxembourg City, Luxembourg, June 2018.

[8] Alysson Bessani, Miguel Correia, Bruno Quaresma, Fernando André, and Paulo Sousa. Depsky: dependable and secure storage in a cloud-of-clouds. *ACM Transactions on Storage (TOS)*, 9(4):1–33, 2013.

[9] Alysson Bessani, João Sousa, and Eduardo EP Alchieri. State machine replication for the masses with BFT-SMART. In *44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 355–362, 2014.

[10] Alysson Neves Bessani, Eduardo Pelison Alchieri, Miguel Correia, and Joni Silva Fraga. DepSpace: A byzantine fault-tolerant coordination service. In *Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008*, Eurosys '08, page 163–176, New York, NY, USA, 2008. Association for Computing Machinery.

[11] Alysson Neves Bessani, Ricardo Mendes, Tiago Oliveira, Nuno Ferreira Neves, Miguel Correia, Marcelo Pasin, and Paulo Verissimo. Scfs: A shared cloud-backed file system. In *USENIX Annual Technical Conference*, pages 169–180. Citeseer, 2014.

[12] BFT-SMaRt. https://github.com/bft-smart. Retrieved July 28, 2023.

[13] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems*, 20(4):398–461, November 2002.

[14] Allen Clement, Manos Kapritsos, Sangmin Lee, Yang Wang, Lorenzo Alvisi, Mike Dahlin, and Taylor Riche. Upright cluster services. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*, SOSP '09, page 277–290, New York, NY, USA, 2009. Association for Computing Machinery.

[15] George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Narwhal and tusk: a DAG-based mempool and efficient BFT consensus. In *Proceedings of the Seventeenth European Conference on Computer Systems*, pages 34–50, 2022.

[16] Dragos Inc. Crashoverride analysis of the threat to electric grid operations. https://dragos.com/blog/crashoverride/CrashOverride-01.pdf. Retrieved July 28, 2023.

[17] Sisi Duan and Haibin Zhang. Practical state machine replication with confidentiality. In *IEEE 35th Symposium on Reliable Distributed Systems (SRDS)*, pages 187–196, 2016.

[18] Michael Eischer and Tobias Distler. Resilient cloud-based replication with low latency. In *Proceedings of the 21st International Middleware Conference*, Middleware '20, page 14–28, New York, NY, USA, 2020. Association for Computing Machinery.

[19] Miguel Garcia, Alysson Bessani, Ilir Gashi, Nuno Neves, and Rafael Obelheiro. OS diversity for intrusion tolerance: Myth or reality? In *2011 IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN)*, pages 383–394. IEEE, 2011.

[20] Miguel Garcia, Alysson Bessani, Ilir Gashi, Nuno Neves, and Rafael Obelheiro. Analysis of operating system diversity for intrusion tolerance. *Software: Practice and Experience*, 44(6):735–770, 2014.

[21] Min Suk Kang, Soo Bum Lee, and Virgil Gligor. The crossfire attack. In *IEEE Symposium on Security and Privacy (SP)*, pages 127–141, May 2013.

[22] Manos Kapritsos, Yang Wang, Vivien Quema, Allen Clement, Lorenzo Alvisi, and Mike Dahlin. All about eve: Execute-verify replication for multi-core servers. In *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, pages 237–250, Hollywood, CA, October 2012. USENIX Association.

[23] Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. All you need is DAG. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, pages 165–175, 2021.

[24] Maher Khan and Amy Babay. Toward intrusion tolerance as a service: Confidentiality in partially cloud-based BFT systems. In *51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 14–25, Virtual Event, June 2021.

[25] John C Knight and Nancy G Leveson. An experimental evaluation of the assumption of independence in multiversion programming. *IEEE Transactions on Software Engineering*, (1):96–109, 1986.

[26] Bijun Li and Rüdiger Kapitza. Bft-dep: automatic deployment of byzantine fault-tolerant services in paas cloud. In *IFIP International Conference on Distributed Applications and Interoperable Systems*, pages 109–114. Springer, 2016.

[27] Bijun Li, Nico Weichbrodt, Johannes Behl, Pierre-Louis Aublin, Tobias Distler, and Rüdiger Kapitza. Troxy: Transparent access to byzantine fault-tolerant systems. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 59–70, 2018.

[28] David R Matos, Miguel L Pardal, Georg Carle, and Miguel Correia. RockFS: Cloud-backed file system resilience to client-side attacks. In *Proceedings of the 19th International Middleware Conference*, pages 107–119, 2018.

[29] Daniel Obenshain, Thomas Tantillo, Amy Babay, John Schultz, Andrew Newell, Md Edadul Hoque, Yair Amir, and Cristina Nita-Rotaru. Practical intrusion-tolerant networks. In *IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 45–56, June 2016.

[30] Office of Cybersecurity, Energy Security, and Emergency Response. Colonial pipeline cyber incident. https://www.energy.gov/ceser/colonial-pipeline-cyber-incident. Retrieved July 28, 2023.

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

[32] Vasilis Pappas, Michalis Polychronakis, and Angelos D Keromytis. Smashing the gadgets: Hindering return-oriented programming using in-place code randomization. In *2012 IEEE Symposium on Security and Privacy*, pages 601–615. IEEE, 2012.

[33] Marco Platania, Daniel Obenshain, Thomas Tantillo, Ricky Sharma, and Yair Amir. Towards a practical survivable intrusion tolerant replication system. In *2014 IEEE 33rd International Symposium on Reliable Distributed Systems*, pages 242–252. IEEE, 2014.

[34] Prime: Byzantine replication under attack. www.dsn.jhu.edu/prime. Retrieved July 28, 2023.

[35] Tom Roeder and Fred B. Schneider. Proactive obfuscation. *ACM Transactions on Computer Systems*, 28(2):4:1–4:54, July 2010.

[36] Mark Russinovich, Manuel Costa, Cédric Fournet, David Chisnall, Antoine Delignat-Lavaud, Sylvan Clebsch, Kapil Vaswani, and Vikas Bhatia. Toward confidential cloud computing. *Communications of the ACM*, 64(6):54–61, 2021.

[37] Fred B. Schneider. The state machine approach: A tutorial. In Barbara Simons and Alfred Spector, editors, *Fault-Tolerant Distributed Computing*, pages 18–41, New York, NY, 1990. Springer New York.

[38] Paulo Sousa, Alysson Bessani, Miguel Correia, Nuno Ferreira Neves, and Paulo Verissimo. Highly available intrusion-tolerant services with proactive-reactive recovery. *IEEE Transactions on Parallel and Distributed Systems*, 21(4):452–465, 2010.

[39] Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. Bullshark: DAG BFT protocols made practical. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 2705–2718, 2022.

[40] The Spines Messaging System. www.spines.org. Retrieved July 28, 2023.

[41] Spire: Intrusion-tolerant SCADA for the power grid. http://www.dsn.jhu.edu/spire/. Retrieved July 28, 2023.

[42] Ahren Studer and Adrian Perrig. The coremelt attack. In *14th European Symposium on Research in Computer Security (ESORICS)*, pages 37–52, 2009.

[43] Robin Vassantlal, Eduardo Alchieri, Bernardo Ferreira, and Alysson Bessani. COBRA: Dynamic proactive secret sharing for confidential BFT services. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1335–1353. IEEE, 2022.

[44] Jian Yin, Jean-Philippe Martin, Arun Venkataramani, Lorenzo Alvisi, and Mike Dahlin. Separating agreement from execution for byzantine fault tolerant services. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, page 253–267, New York, NY, USA, 2003. Association for Computing Machinery.

[45] Lidong Zhou, Fred B. Schneider, and Robbert Van Renesse. APSS: Proactive secret sharing in asynchronous systems. *ACM Transactions on Information and System Security*, 8(3):259–286, aug 2005.

X. Appendix

A. Calculating the number of required cloud replicas

To calculate the number of cloud replicas, we generalize the replica distribution formula from [7]. That work showed how to distribute replicas across sites to withstand exactly one site disconnection, while tolerating f intrusions and exactly one proactive recovery. We adapt this approach to tolerate any number of site disconnections, and any number of simultaneous proactive recoveries. We assume a BFT replication protocol that normally uses $3f+2k+1$ replicas to simultaneously withstand f intrusions and k proactive recoveries.

Similar to [7], we let the k parameter in the standard formula represent the total number of replicas that may be simultaneously *unavailable* (not only going through proactive recovery). For clarity, we use u to represent this total number of unavailable replicas, since we use k_c to represent the number of cloud replicas that may be going through proactive recovery. We let f_c represent the number of tolerated intrusions in the cloud, and let n_c represent the total number of cloud replicas. Therefore, in order for the BFT protocol to make progress, we require that $2f_c+u+1$ out of $n_c=3f_c+2u+1$ total replicas are correct, available, and connected.

If we assume replicas are distributed as evenly as possible across sites, then, to tolerate d_c site disconnections out of S_c total sites, in a system with n_c total replicas, we require:

$$u \geq d_c \left\lceil \frac{n_c}{S_c} \right\rceil + k_c \tag{1}$$

This guarantees that the tolerated number of unavailable replicas is at least the number of replicas in the d_c disconnected sites, plus the k_c replicas that may be down for proactive recovery.

To get a *lower bound* for the required value of u, we can drop the ceiling function and calculate:

$$u \geq d_c \left(\frac{n_c}{S_c} \right) + k_c \tag{2}$$

Substituting the formula for n_c into (2) gives us:

$$u \geq d_c \left(\frac{3f_c+2u+1}{S_c} \right) + k_c \tag{3}$$

Solving (3) for u, we get:

$$u \geq \frac{3d_c f_c + d_c + S_c k_c}{S_c - 2d_c} \tag{4}$$

Since we require u to be an integer (a whole number of replicas), we can apply the ceiling function and choose u as:

$$u = \left\lceil \frac{3d_c f_c + d_c + S_c k_c}{S_c - 2d_c} \right\rceil \tag{5}$$

To get a lower bound on the required number of replicas n_c, we can substitute the above lower bound for u into the formula $3f_c+2u+1$ to get:

$$n_c = 3f_c + 2\left\lceil \frac{3d_c f_c + d_c + S_c k_c}{S_c - 2d_c} \right\rceil + 1 \tag{6}$$

In the case where the n_c resulting from equation (6) is evenly divisible by S_c, we can directly use this n_c as our number of cloud replicas, and distribute the replicas evenly across the S_c sites. This is guaranteed to satisfy the requirement in inequality (1). The reasoning for this is as follows: from inequalities (2)-(4) and equation (5), we have shown that setting u as in (5) satisfies inequality (2). When n_c is evenly divisible by S_c, the right-hand side of inequality (2) is exactly equal to the right-hand side of inequality (1). Thus, this choice of u also satisfies inequality (1) in this case.

However, when the n_c resulting from equation (6) is not evenly divisible by S_c, we are not guaranteed that inequality (1) is satisfied. In this case, we can calculate an upper bound on the required value of u, and then test each value of u between the lower bound and the upper bound to find one that satisfies inequality (1).

By the definition of the ceiling function, we know:

$$d_c \left\lceil \frac{n_c}{S_c} \right\rceil + k_c < d_c \left(\frac{n_c}{S_c} + 1 \right) + k_c \tag{7}$$

We want to find a value of u that is guaranteed to satisfy inequality (1). Based on inequality (7), we know the following choice of u is safe:

$$u = d_c \left(\frac{n_c}{S_c} + 1 \right) + k_c \tag{8}$$

Substituting the formula for n_c into (8) we get:

$$u = d_c \left(\frac{3f_c+2u+1}{S_c} + 1 \right) + k_c \tag{9}$$

Solving (9) for u, we get:

$$u = \frac{3d_c f_c + d_c + S_c d_c + S_c k_c}{S_c - 2d_c} \tag{10}$$

Since we require u to be an integer, we can apply the ceiling function:

$$u = \left\lceil \frac{3d_c f_c + d_c + S_c d_c + S_c k_c}{S_c - 2d_c} \right\rceil \tag{11}$$

Setting u as in (11) satisfies inequality (1). But, this is not necessarily the minimum value. To find the minimum integer value of u that satisfies (1), we consider every value of u between the values in (5) and (11):

$$\left\lceil \frac{3d_c f_c + d_c + S_c k_c}{S_c - 2d_c} \right\rceil \leq u \leq \left\lceil \frac{3d_c f_c + d_c + S_c d_c + S_c k_c}{S_c - 2d_c} \right\rceil \tag{12}$$

To find the number of required replicas n_c, we test each integer in this range as a possible value for u, and calculate n_c using the usual formula:

$$n_c = 3f_c + 2u + 1 \tag{13}$$

We start from the lower bound, and for each u and corresponding n_c, we check whether inequality (1) is satisfied. We choose the smallest u that satisfies this inequality, and use the

corresponding n_c from equation (13) as the total number of replicas, distributing them as evenly as possible across the S_c sites.

Note that a clear implication of the lower bound (5) and upper bound (11) for u is that we require the quantity $S_c - 2d_c$ (the denominator in the formulas for u) to be strictly greater than 0. Thus, we require the total number of cloud sites:

$$S_c \geq 2d_c + 1 \tag{14}$$

B. Batching of Client Requests

To generate threshold signatures on new client requests efficiently, it is important to be able to *batch* requests, such that replicas do not need to sign every request individually. Unfortunately, client requests may not arrive in the same order at every on-premises replica, so replicas may not generate identical batches (and requiring replicas to agree on the batch contents is essentially equivalent to running the agreement protocol). Without identical batches, the partial signature shares generated over these batches will not combine correctly. Therefore, we allow replicas to contribute partial signatures to batches received from other replicas as described below.

Each on-premises replica batches received client requests with a limit on the maximum count and/or time, sorting batched requests by their client IDs. Next, the on-premises replica encrypts the batch, generates a partial signature over it, and sends the encrypted batch (with its partial signature) to all other on-premises replicas in its site. Upon receiving a batch of client requests, an on-premises replica decrypts it and verifies each of the client requests with the respective client's public key. Once the entire batch is verified, the on-premises replica generates a partial signature for the encrypted batch and sends it back. Upon collecting $f_o + 1$ partial signatures (including its own), an on-premises replica can generate a threshold signature. It sends the encrypted batch (with the threshold signature) to the cloud replicas through the cloud-side privacy firewall, which verifies the threshold signature before forwarding it.

To optimize the batching process, when an on-premises replica's batch of encrypted requests matches that of another on-premises replica's, it uses the accompanying partial signature from the other on-premises replica for its own batch of encrypted requests. Since we require the client requests inside a batch to be sorted by client IDs, and there is typically very little delay variation or chance of message loss within a site, we can expect that new batches from different on-premises replicas within a site to almost always match, and hence threshold signatures can be generated quickly.

Note that we do not need to change our validity period procedure for batching requests. Since we limit each client to one outstanding request at a time, in the worst case scenario, a malicious replica can space out the client requests to one per batch, but the validity period takes into account the total number of clients. Hence, the malicious replica will not be able to quickly fill up the validity period and slow down processing of new requests.

The cloud replicas treat this encrypted batch of requests same as a single encrypted request (check validity period and threshold signature, order the batch, threshold sign the ordered encrypted batch of requests, and finally send this back to the on-premises replicas). Upon receiving a ordered encrypted batch of requests, the on-premises replica checks the threshold signature, decrypts the batch, and then executes each request in the same order as they are in the batch. Since each batch has an ordinal number, batches are processed consecutively according to their ordinal numbers.