Toward Intrusion Tolerance as a Service: Confidentiality in Partially Cloud-Based BFT Systems

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

***Abstract*—Recent work on intrusion-tolerance has shown that resilience to sophisticated network attacks requires system replicas to be deployed across at least three geographically distributed sites. While commodity data centers offer an attractive solution for hosting these sites due to low cost and management overhead, their use raises significant confidentiality concerns: system operators may not want private data or proprietary algorithms exposed to servers outside their direct control.**

We present a new model for Byzantine Fault Tolerant replicated systems that moves toward "intrusion tolerance as a service". Under this model, application logic and data are only exposed to servers hosted on the system operator's premises. Additional offsite servers hosted in data centers can support the needed resilience without executing application logic or accessing unencrypted state. We have implemented this approach in the open-source Spire system, and our evaluation shows that the performance overhead of providing confidentiality can be less than 4% in terms of latency.

I. Introduction

Intrusion tolerance, or the ability to operate correctly even while partially compromised by an attacker, is an increasingly important concern for high value systems. Critical infrastructure, such as power grid Supervisory Control and Data Acquisition (SCADA), represents one example. Because these systems are targets for sophisticated nation-state-level attacks, there has been considerable research on how to make them intrusion tolerant through Byzantine Fault Tolerant (BFT) replication of the control servers (e.g. [1], [2], [3], [4], [5]).

While this line of work has led to solutions that can provide strong correctness and performance guarantees in the face of sophisticated attacks, in order for it to have an impact on real systems, it must be deployed, and correctly deploying and managing intrusion-tolerant replicated systems remains challenging. There have been efforts to make this easier (e.g. through the development of tools like BFT-SMaRt [6], [7]), but it still requires a relatively high level of expertise. In the power grid domain, it is unlikely to be feasible for every utility operator to develop and maintain this expertise in-house [8], and this is likely to be true across other domains as well.

Further complicating the deployment and management of intrusion-tolerant services, recent work shows that resilience to sophisticated network attacks that can isolate a targeted site from the rest of the network requires a significantly higher number of system replicas than traditional BFT systems and requires them to be deployed across at least three geographically distributed sites [4]. Commodity data centers offer an attractive solution to reduce the cost and management overhead of constructing and operating the additional required sites, and the work in [4] shows how existing power grid control centers can be augmented with additional data center sites to create a cost-effective solution. However, using commodity data centers raises significant *confidentiality* concerns [8]: for many applications, system operators are likely to consider exposing private data or proprietary algorithms to servers outside their direct control to be an unacceptable trade-off.

We present a new model for BFT systems that moves toward offering "intrusion tolerance as a service". Under this model, application logic and data are only exposed to servers hosted on the system operator's premises. However, the intrusion-tolerant system architecture can be designed by a service provider, and additional offsite servers can be hosted in data centers managed by the service provider to provide the needed resilience to system compromises and network attacks. These offsite servers participate in the BFT replication protocol, but do not execute application logic and only store encrypted state and updates. We show that our approach is able to provide the same resilience to system compromises and network attacks as in [4], *without requiring application state and logic to be exposed to data center replicas*.

We implement our new partially cloud-based architecture and protocols in Confidential Spire, a SCADA system for the power grid based on Spire 1.2 [4], [9]. We evaluate the performance overhead of providing confidentiality in an emulated power grid SCADA setup and find that Confidential Spire only adds an overhead of about 2ms compared to Spire 1.2 when tolerating one intrusion (less than 4% increase in latency), and an overhead of 6.8ms when tolerating two intrusions (less than 13% increase in latency). In both cases, latency is below 100ms for all requests, meeting the timing requirements of power grid SCADA systems [10], [11].

The contributions of this work are:

- The design of the first BFT system that can leverage offsite data centers to achieve resilience to simultaneous network attacks and system compromises, without requiring confidential state or algorithms to be exposed to data center servers.

- Extensions to the basic system to provide well-defined confidentiality guarantees in the case that an on-premises server is compromised.
- An implementation and evaluation of the system in the context of SCADA for the power grid. We show that the performance overhead of providing confidentiality is acceptable, and the system can meet the latency requirements of power grid SCADA.

II. Background and Related Work

A. BFT Basics

Byzantine Fault Tolerant (BFT) state machine replication is a well-known technique to provide intrusion tolerance, enabling a system to guarantee safety (correctness and consistency of the system state) and liveness (progress in processing updates) even if up to some threshold number of replicas are compromised (e.g. [12]). The number of tolerated compromises is most often f out of $3f+1$ total replicas [12], although some systems can tolerate f out of $2f+1$ total replicas with additional assumptions or trusted hardware [13], [14], [15], [16]. Some systems additionally guarantee *performance* under attack, rather than only liveness [17], [18], [19], [20].

To support long system lifetimes, it is necessary to employ *proactive recovery*, which allows replicas to be periodically taken down and restored to a known clean state [12], [21]. Providing continuous availability with proactive recovery typically requires $3f + 2k + 1$ total replicas to simultaneously tolerate up to f compromised replicas and k recovering replicas [22].

B. BFT and Network Attacks

The motivation for our work comes from the recent Spire intrusion-tolerant SCADA system for the power grid [4], [9], which showed that at least three geographically distributed sites are needed to withstand sophisticated network attacks that can target and isolate a site from the rest of the network. The intuition for this is the following: since BFT replication protocols require more than half of the system replicas (in fact, $2f+k+1$ out of $3f+2k+1$) to be up, correct, and connected in order to make progress, any system that distributes replicas across fewer than three sites can be prevented from making progress by isolating a single site. Clearly, if all replicas are located in a single site, a denial of service attack targeting that site can prevent it from communicating with remote clients and thus render it unable to receive and process their updates. If replicas are split across only two sites, targeting the larger of the two sites will disconnect a majority of the system replicas, leaving the rest unable to make progress without them.

Due to the high expense of constructing additional control centers with full capabilities for communicating with Remote Terminal Units (RTUs) and Programmable Logic Controllers (PLCs), and controlling power grid equipment, Spire introduced an architecture that uses two power grid control centers (which typically exist today for fault tolerance purposes) and supplements them with additional data center sites that do not need to communicate with RTUs and PLCs. The use of data centers can also reduce the management overhead of the higher number of replicas that Spire needs to support its threat model (12 total replicas to support $f = 1$ and 1 disconnected site).

However, data center replicas are still required to maintain a full copy of the system state and execute application logic to process incoming updates. This raises confidentiality concerns, as it requires SCADA operators to expose their private system state and algorithms to offsite replicas potentially hosted by a third party. Today, if a system operator wants to avoid trusting a third party with this information, they must take on the responsibility for managing the full deployment (and constructing their own additional sites to host system replicas).

Our goal in this paper is to address this issue through a new hybrid model for partially cloud based systems: system operators host and manage "on-premises" replicas distributed across two geographic sites that they manage and control, while a service provider hosts and manages additional replicas located in data center sites. In our model, not only do service-provider-managed replicas not need to communicate with clients, but they only see encrypted state and do not execute application logic.

C. Confidential BFT

Our goals are closely related to prior work on confidential BFT systems. These systems fall into two main categories: approaches based on secret sharing, and approaches based on privacy firewalls.

Secret Sharing. Confidential BFT systems based on secret sharing include DepSpace [23], Belisarius [24], and COBRA [25]. These systems protect confidentiality of the system state as long as no more than f replicas are compromised. To do this, clients encode data using an $(f + 1, n)$-threshold secret sharing scheme, where $f + 1$ shares out of n total shares are needed to reconstruct the confidential data. Since each replica only receives one share, this guarantees that up to f compromised replicas are unable to successfully reconstruct the data. COBRA [25] additionally provides for share renewal, allowing it to tolerate up to f compromises per renewal epoch (as opposed to over the entire life of the system).

This approach offers strong confidentiality guarantees that initially appear to fit our goal well: in fact, such an approach could enable management of the *entire* replicated system to be offloaded to a service provider, with all replicas hosted in data centers, while still guaranteeing that the system state remains confidential. However, due to practical limits on the types of operations that can be performed on the encrypted data, current systems typically support a limited set of operations, such as basic key-value storage operations [25] or tuple space storage [23], with Belisarius also offering the ability to perform addition on stored values [24]. Moreover, even if these systems supported general operations (e.g. via secure multiparty computation or homomorphic encryption), the operations themselves must be executed at all servers and therefore cannot be kept private. For certain applications, it may be desirable to keep application code or algorithms private

(e.g. because the algorithms are proprietary or otherwise sensitive).

Secret sharing has also been used to create encrypted BFT-replicated storage systems. DepSky [26] is a cloud-of-clouds storage system that uses BFT replication with secret sharing to store data across several different cloud providers and guarantees data availability, integrity and confidentiality as long as no more than f of $3f + 1$ clouds fail in a Byzantine manner. SCFS [27] is a distributed file storage system that can use DepSky as a backend to store encrypted files and shares of the encryption keys on several clouds. RockFS [28] is built on top of SCFS and improves on it by tolerating some malicious client-side behaviour: (1) it prevents illegal modification of client credentials by using a secret sharing scheme, (2) it prevents illegal access to the client's local cache by encrypting it, and (3) it allows administrators to undo any illegal modification of client files in the cloud by keeping a log of every modification.

While these systems enable confidential, intrusion-tolerant data storage in the cloud, they do not address Byzantine behaviour of the applications that generate or modify the data; both DepSky and SCFS assume that clients are not malicious, while RockFS can prevent or recover from specific types of malicious client behaviour. In contrast, we aim to support intrusion tolerance at the application level, not only for its data storage.

Privacy Firewall. Confidential BFT based on privacy firewalls was introduced by Yin et al. [29] and later built on by Duan and Zhang [30]. This approach is based on *separating agreement from execution* [29]: replicas are split into an *agreement cluster* that uses a BFT agreement protocol to establish a total order on incoming client updates and an *execution cluster* that executes the ordered stream of updates and generates client replies. A *privacy firewall* is constructed between the execution cluster and agreement cluster to filter replies sent by the execution cluster and to ensure that no confidential information is allowed to exit the execution cluster (as long as no more than a threshold f of the firewall nodes are compromised).

This approach again has properties that appear to fit our goal well: using the strategy of separating agreement and execution, an attractive possibility is for execution nodes to be hosted on-premises, while agreement nodes can be hosted in remote data centers and managed by a service provider. This would not require agreement nodes to carry out application logic or understand the updates they order (as ordering can be done on encrypted updates [29], [30]).

However, the existing solutions do not fully meet our needs. They assume a model where all agreement and execution replicas are hosted in a single site, and the objective is to prevent a compromised execution node from exfiltrating confidential information over the network. Therefore, their architectures place the agreement cluster between the clients and execution nodes, and only allow *agreement nodes* to communicate with clients or other entities outside the site. However, in many contexts, it may not be desirable or even possible for data centers to communicate with clients (e.g. in the power grid context, this is typically not feasible [4]).

The execute-verify model of Eve [31] and execute-order-validate paradigm of Hyperledger [32] address this issue by performing execution *before* ordering, but in doing so add complexity at the application layer (e.g. to deal with state rollback), without solving the following more fundamental problem: these approaches cannot be straightforwardly extended to support the multi-site model needed to cope with network attacks. We show that the observation in [4] implies that *system state* must be stored in at least three distinct sites in order to maintain continuous availability in the presence of network attacks, preventing a clean separation between execution and agreement unless system operators are willing to build and manage at least three (execution) sites. In addition, it is not clear how to apply the privacy firewall concept in a multi-site deployment, as it strongly relies on a specific physical network setup.

III. System and Threat Model

A. System Model

We introduce a new system model for partially cloud-based BFT systems. In this model, system management is shared between *system operators* who are responsible for managing an application and are typically experts in the application domain, and *service providers* who offer data center hosting capabilities (and potentially other services).

In our model, a system is physically deployed across locations owned and operated by the system operator (*on-premises sites*) and infrastructure operated by the service provider (*data center sites*). We refer to system replicas located in on-premises sites as *on-premises replicas* and replicas located in data center sites as *data center replicas*.

Our goal is to provide strong intrusion tolerance guarantees without significantly increasing the management overhead for system operators. In many applications that require fault tolerance, operators are likely to already maintain two on-premises sites (e.g. for primary-backup). In contrast to adding servers to an existing site, creating a new one involves provisioning the physical location/building to house it, hiring management personnel (since fault independence requires a sufficient geographical distance from existing sites), and for some applications, provisioning specialized equipment to communicate with client sites. Therefore, we assume two on-premises sites and design our architecture to avoid constructing any additional on-premises sites.

Due to privacy concerns (and potentially feasibility constraints), *clients* only communicate with *on-premises replicas* (they do not communicate directly with data center replicas).

B. Threat Model

We consider the same broad threat model as in [4], which includes both system-level compromises of the server replicas and network-level attacks that aim to disrupt communication among replicas and/or between replicas and clients.

We consider a broad range of network attacks, but, as in [4], we reduce this to a simpler model through the use of an intrusion-tolerant overlay network to connect the sites to one another [33], [34]. With the use of the intrusion-tolerant network, the network attacks we still need to address are reduced to sophisticated (resource-intensive) denial of service attacks that can target and isolate a single geographic site. We assume that at any time, up to one site (either on-premises or data center) may be subject to such an attack and thus disconnected from the rest of the network.

As in other intrusion-tolerant replicated systems, we assume that up to a threshold number f of replicas may be compromised. Compromised replicas may behave arbitrarily and collude with one another. As in prior work, we employ proactive recovery to allow the system to tolerate up to f compromises within a limited time window, as opposed to over the entire system lifetime. We assume that replicas are recovered one at a time, and that one replica's recovery finishes before the next replica's recovery starts.[1] Thus, at any time our threat model includes up to f compromised replicas, up to one replica that is unavailable because it is going through proactive recovery, and one disconnected (or otherwise unavailable) geographic site.

We assume each replica has access to a hardware-protected private key (e.g. using the TPM) that it can use for signing, but that cannot be deleted, modified, or exfiltrated from the machine. This key is used to bootstrap after proactive recovery, when the replica generates a new session-level signing key. We assume an attacker cannot break cryptographic protocols.

C. Service Properties

Our safety, liveness, and performance guarantees are essentially the same as those specified in [4], although we adapt them to a generic replicated system; where [4] specifically considered SCADA Masters, HMIs, RTUs, and PLCs, we state our guarantees in terms of generic servers and clients.

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 replicas is also identical.*

Our system guarantees safety as long as no more than f replicas are simultaneously compromised. Note that while safety as defined above is maintained in the presence of an unlimited number of compromised clients, compromised clients may still cause the system to take incorrect actions by submitting malicious updates; we only guarantee that all replicas will observe and execute these updates in a consistent way (this is a general limitation in BFT replication). We do not consider data center replicas as executing updates here, as we only care about the state as it is visible to clients.

Definition 2 (Bounded Delay)**.** *The latency for an update introduced by a correct authorized client to be executed by at least $f + 1$ correct on-premises replicas (and thus have its effects made visible) is upper bounded.*

To guarantee bounded delay, we require that the conditions of our threat model are met: at most f replicas are compromised, at most one replica is undergoing proactive recovery, and at most one site is downed or disconnected due to network attack. In addition, the remaining replicas (i.e. all correct, non-recovering replicas located outside the disconnected site) must be able to communicate with one another, and the remaining correct *on-premises* replicas must be able to communicate with clients. Finally, communication among the remaining correct replicas must meet the network stability requirements of Prime [37], which is used as our underlying agreement protocol and requires that the latency variance between each pair of correct servers is bounded (see [4] for additional discussion).

Our new contribution is to combine the above guarantees with the following *confidentiality* property.

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

Our base system provides this guarantee as long as no *on-premises replica* is compromised. An unlimited number of data center replicas may be compromised without violating confidentiality. Note that a compromised client may always leak *its own* state or updates; our model does not prevent this, nor does any other confidential BFT work we are aware of. When we refer to system state in Definition 3, we refer to the full state of the system maintained by the on-premises replicas.

Note that this guarantee is not comparable to those of the confidential BFT systems discussed in Section II-C: if any on-premises server is compromised (over the entire lifetime of the system), it can cause confidentiality to be violated. However, we argue that the novel combination of guarantees we provide represents a significant advance over the state of the art. The Spire system provided a level of attack resilience in terms of safety and performance guarantees that was not possible before, but introduced a trade-off in terms of confidentiality. For a baseline system that provides fault tolerance through standard primary-backup mechanisms hosted fully on-premises, transitioning to the Spire architecture offers much stronger safety and performance guarantees, but at the cost of somewhat weaker confidentiality guarantees. In the baseline system, confidentiality may be violated if an on-premises server is compromised. However, if data center sites are introduced, confidentiality is violated if *either* a data center server or an on-premises server is compromised, and even in the case where no server is compromised, certain information

[1] This requires certain synchrony assumptions: an attacker must not be able to arbitrarily prolong a replica's recovery (see [35]). However, in practice these can be met: simple trusted devices can trigger recovery by cycling the power to a replica, and recovery intervals on the order of one replica per day are sufficient [36]. If an adversary can prevent a replica from collecting messages needed for recovery for a full day, that replica is effectively disconnected. The intrusion-tolerant overlay makes such disconnections very difficult, and our system technically allows recoveries of replicas in a disconnected site to overlap, as long as the total number of recovering replicas is no more than the size of the largest site plus one.

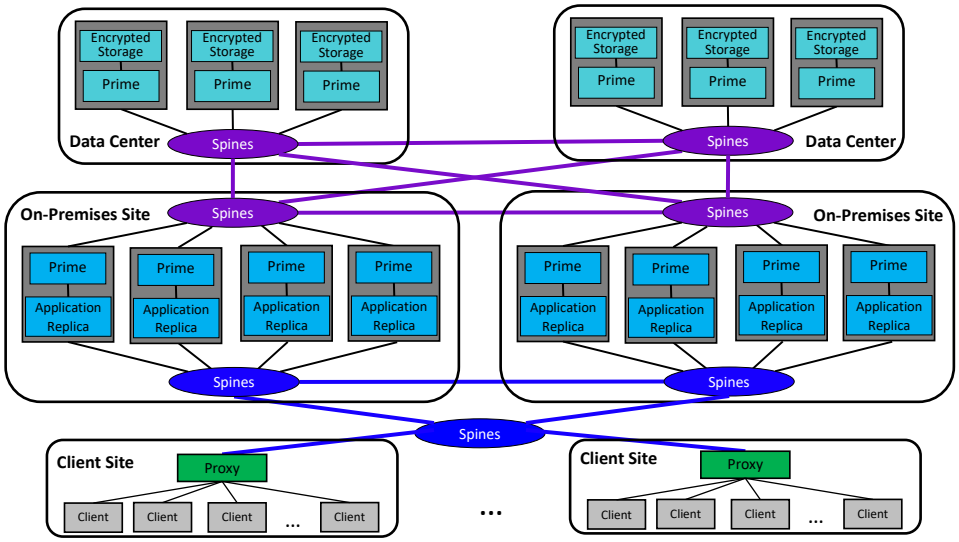

Fig. 1. System architecture overview, showing 2 on-premises sites (each containing 4 replicas) and 2 data centers (each containing 3 replicas).

is made accessible to the service provider managing the data center servers. Our architecture eliminates this trade-off: the strictly improved safety and performance guarantees are provided while maintaining *the same* level of confidentiality as in the baseline system. This is likely to substantially increase its acceptability to system operators. In Section V-D, we discuss how the system can at least limit the amount of state that can be disclosed if an on-premises server is compromised.

IV. Partially Cloud-Based BFT Architecture

The key observation behind our system design, and the reason that a straightforward separation of execution replicas running in on-premises sites and agreement replicas running in cloud data center sites cannot support our required guarantees, is that network-attack resilience requires system state to be stored in at least three distinct geographic sites. As discussed in Section II-B, the work in [4] observed that because BFT replication protocols require (more than) a majority of replicas to be connected in order to safely make progress and order updates, they cannot guarantee continuous availability in the presence of network attacks unless at least 3 sites are used: otherwise a network attack targeting a single site can isolate a majority of the system, leaving the remainder unable to make progress, and rendering the system unavailable.

In fact, exactly the same observation applies to the storage of system state. To see why this is the case, consider a system with exactly two on-premises sites. Under our threat model, any one site may be disconnected at any time, so the system must be able to make progress with only a single on-premises site up and connected to the data center sites. Consider that on-premises site A is up, connected to data center sites and client sites, and receiving, submitting for ordering, and executing incoming client updates, while on-premises site B is under denial-of-service attack and isolated from the rest of the network. Then, the attacker shifts focus to instead target on-premises site A: site A is now isolated, while site B rejoins the network and is now connected to the data center and client sites. As before, the conditions of our threat model are met, so the system *should* be able to process updates and make progress. However, on-premises site B has missed all of the client updates that were processed while it was disconnected. If data center sites do not store any system state, it is impossible for the replicas in site B to catch up and recover the state to resume safely executing updates. In order to support our threat model, a disconnected on-premises site must be able to rejoin the network, catch up, and resume processing updates without communicating with the other on-premises site.

Therefore, our approach is for data center replicas to store *encrypted* updates and state checkpoints. By encrypting updates and checkpoints with keys known only to the on-premises replicas, we can allow data center replicas to store them, without being able to decrypt and interpret them. This allows a disconnected on-premises site to rejoin the network, collect state, and resume processing updates based only on information obtained from data center replicas, but without requiring data center replicas to access unencrypted state or perform any application-specific logic.

A. System Architecture

An overview of our architecture is shown in Figure 1. Our high-level architecture is based on the Spire architecture [4]. System replicas are distributed across two on-premises sites and a configurable number of data center sites. Sites are connected through an instance of the Spines intrusion tolerant network [33], [34] to provide resilience to a broad range of network attacks. On-premises sites are additionally connected to client sites through a separate Spines instance. Proxies support clients that cannot be modified to use a BFT protocol. Clients in a single physical location may be grouped behind a single proxy, or each client can have its own proxy. A proxy collects updates from its respective client(s), digitally signs them so that server replicas can verify their authenticity, and

submits them to the system by sending them to on-premises servers. Client proxies also validate responses received from the server replicas: specifically, server replicas generate threshold signatures on responses using an $(f+1, n)$-threshold scheme, so the proxy can verify a single signature to confirm that at least one correct replica agreed on the message.

The key difference from [4] is the separation of functionality between on-premises and data center replicas. In our system, all replicas host an instance of the Prime intrusion-tolerant replication engine [38] and participate in the replication protocol. However, only on-premises servers host application replica instances. Client updates received by on-premises servers are encrypted before being submitted for ordering and sent to data center servers. Post-ordering, updates are decrypted and executed at (only) the on-premises application replicas, while those same updates are stored in encrypted form at the data center servers.

B. Replica Distribution

In configuring the system, replicas must be distributed across sites such that the system is able to safely process updates and meet its bounded delay guarantee under the full threat model we consider. Prime (when configured to support proactive recovery, as in [4]) requires a total of $3f + 2k + 1$ replicas to tolerate f compromised replicas and k unavailable replicas, where a replica may be unavailable either because it is going through proactive recovery or because it has been disconnected from the network (or because it has simply crashed). In order to guarantee progress, with bounded delay, at least $2f + k + 1$ of those replicas must be up, correct, and connected (with sufficient network stability).

The work in [4] showed that in order to ensure $2f + k + 1$ correct replicas are always available, it is necessary to ensure that no single site contains more than $k-1$ servers: otherwise the disconnection of a single site, plus an ongoing proactive recovery elsewhere in the system could cause more than k replicas to become unavailable at the same time, preventing the system from making progress. That work shows that providing this guarantee requires setting $k \geq \left\lceil \frac{3f+S+1}{S-2} \right\rceil$, where S is the total number of sites (on-premises + data center), and distributing replicas as evenly as possible across sites [4], [39].

However, an additional constraint under our threat model comes from the separation of functionality between on-premises and data center replicas: only on-premises replicas can execute updates and communicate with clients. To verify that a received message is correct, a client must be able to confirm that $f + 1$ servers agreed to it (to ensure at least one correct server was involved). This means that generating verifiable responses requires that $f+1$ *on-premises* replicas are available at any time: data center replicas cannot participate in generating client responses, as this requires knowledge of update contents, system state, and application logic.

In the worst case, under our threat model, one of the two on-premises sites may be disconnected, and the other may contain f compromised replicas and one replica undergoing proactive recovery. Therefore, in order to ensure that $f + 1$ correct replicas are available at all times, *each* of the two on-premises sites must contain at least $2f+2$ total replicas ($2f+2$ replicas -1 recovering replica $-f$ compromised replicas $= f+1$ available correct replicas). However, since we must still have k strictly greater than the size of the largest site, this adds the restriction $k \geq 2f + 3$.

	2 on-premises + 1 data center	**2 on-premises + 2 data centers**	**2 on-premises + 3 data centers**
$f = 1$	6+6+6 (18)	4+4+3+3 (14)	4+4+2+2+2 (14)
$f = 2$	9+9+9 (27)	6+6+5+4 (21)	6+6+3+3+3 (21)
$f = 3$	12+12+12 (36)	8+8+6+6 (28)	8+8+4+4+4 (28)

TABLE I
SYSTEM CONFIGURATIONS TOLERATING A PROACTIVE RECOVERY, DISCONNECTED SITE, AND 1, 2, OR 3 INTRUSIONS.

Together, the two above restrictions give us the requirement:

$$k \geq \max\left(2f + 3, \left\lceil \frac{3f + S + 1}{S - 2} \right\rceil\right)$$

After finding the minimal value of k using this formula, the total number of required replicas is calculated from the original formula: $n = 3f + 2k + 1$. To distribute these replicas across the sites, we must first ensure that at least $2f + 2$ replicas are placed in each on-premises site, and then distribute the remaining replicas across the sites such that the total number of replicas per site is as even as possible. The results of this process for several different system options are shown in Table I. In Table I, we consider configurations tolerating 1-3 intrusions ($f = 1$, $f = 2$, and $f = 3$), with replicas distributed across two on premises sites and 1-3 data centers. The first 2 numbers per cell denote the number of replicas in each on-premises site, while the following numbers represent the number of replicas in the data centers, and the final number in parentheses represents the total number of replicas. For example, configuration "4+4+3+3" represents 4 replicas in each on-premises site, and 3 replicas in each data center, for a total of 14 required replicas.

While the total number of replicas is considerably higher than the typical $3f+1$, this is because we (1) support proactive recovery (which requires $3f+2k+1$ replicas) and (2) provide stronger guarantees, tolerating not only f compromises, but also network attacks that can disconnect an entire site. This threat model was first introduced in [4], which showed that for the case of $f = 1$, 12 replicas are needed. We slightly increase that number to 14, but we believe this is justified to provide the confidentiality needed to trust a cloud provider and thus avoid the need for the system operator to manage the large set of sites and replicas themselves. If we consider a system operator who already supports fault tolerance, deploying primary and backup sites, each of which includes primary and backup replicas, we only require that they add 2 on-premises servers per site: the remaining sites and replicas are fully managed by the cloud provider.

V. PROTOCOLS FOR PARTIALLY CLOUD-BASED BFT

Having described how to distribute replicas and set up the system, we next describe the protocols used to submit and process updates.

A. Introducing Client Updates

Clients submit updates to the system through proxies. The proxy digitally signs each update using its private key before sending it to the on-premises servers. On-premises servers can then verify the signature on the update, encrypt it, and inject it into Prime for ordering.

However, our new model introduces a challenge, as data center replicas need to verify that each update submitted for ordering actually came from a correct client (and was not maliciously generated by an adversary), yet data center replicas do not have the ability to decrypt client updates. In fact, they should not be required to maintain any information about client identities or public keys (in some cases, client IP address or locations may be a sensitive type of state that system operators would like to avoid revealing [8]). Requiring updates to be signed by the on-premises server injecting them is not sufficient, as any individual server could be compromised and manufacture a large number of spurious updates, forcing the system to work to order the bogus updates.

Our approach is for on-premises servers to cooperate to generate a threshold signature on each introduced update. To do this, we require each client to send its updates to $2f+k+1$ on-premises replicas, which guarantees that at least $f+1$ correct replicas will receive the request. Upon receiving a client request, the on-premises replica first checks its validity, then encrypts the request message body. Then, it creates a partial threshold signature (using an $(f+1, n)$-threshold scheme) for the encrypted client update and multicasts this partial threshold signature to all other on-premises replicas. Upon collecting $f+1$ partial signatures, an on-premises replica combines them to form the full threshold signature, and injects the threshold-signed update into Prime for ordering. All other replicas, including the data center replicas, can verify the full threshold signature to validate that a request is legitimate.

Client update encryption presents one remaining challenge: the individual on-premises replicas all need to perform the encryption independently but come up with *the same encrypted content*, so that the threshold signature shares that they independently generate will combine correctly. To do this, we assume on-premises replicas maintain two shared secret keys per client: the first is the shared key used to perform symmetric encryption and decryption of updates for that client, while the second is used as one of the inputs to a pseudorandom function used to generate initialization vectors, similarly to the approach in [30]. Details about our implementation can be found in Section VI-B. For now, we assume that all of these per-client key pairs are stored in persistent read-only memory and reloaded from there after proactive recovery, although we discuss how to weaken this restriction in Section V-D.

B. Ordering Updates and Disseminating Results

Once an on-premises server generates a full threshold signature on an encrypted client update and injects it into Prime, it is assigned an *ordinal*, or sequence number in the global total ordering through the Prime agreement protocol [37], and then delivered to the application to be decrypted and executed (in the case of on-premises servers) or simply stored in encrypted form along with its ordinal (at data center servers).

As part of executing an ordered update, application replicas may generate a response message that needs to be sent to a client. To generate a single response that can be verified by a client proxy based on a single service public key, application replicas generate a threshold signature on the response, again using an $(f+1, n)$-threshold scheme to ensure the message is agreed on by at least one correct replica. This is the same approach as in [4], but in our case, only on-premises replicas can participate in generating the response. Our replica distribution framework (Section IV-B) guarantees that it is always possible to generate such a signature under the conditions of our threat model.

C. Checkpoints and State Transfer

Since storing every ordered client request will eventually exhaust replicas' storage capacity, we keep only a limited number of the latest encrypted client requests and replace older requests by encrypted checkpoints. At specified checkpoint intervals (i.e. every C ordered updates), each on-premises replica creates and encrypts a checkpoint that represents its state up through the execution of the last ordered client update (similar to [12] and others). Note that an on-premises replica does not consider itself to have fully executed a particular update until it has generated and sent a threshold-signed client response message for any outgoing messages that were generated as a result of its execution. The latest (threshold-signed) outgoing message for each client is included in the system state, since these may need to be retransmitted.

After generating an encrypted checkpoint, the on-premises replica then creates and signs a checkpoint message that contains the encrypted checkpoint, as well as the (cleartext) sequence number it corresponds to (the global sequence number of the last ordered update that was executed and reflected in the state). The replica then multicasts this checkpoint message to all other replicas (including both on-premises and data center replicas). When a replica (on-premises or data center) receives $f+1$ identical encrypted checkpoints from different replicas for the same sequence number, then this encrypted checkpoint can be marked as *correct*: at least 1 correct replica has agreed that this checkpoint represents the system state at the given sequence number.

Data center replicas do not create their own checkpoints. Instead, when a data center replica collects a *correct* encrypted checkpoint, it creates and signs a checkpoint message containing that encrypted checkpoint, and then multicasts this checkpoint message to all other replicas. When a replica (on-premises or data center) receives $2f+k+1$ identical encrypted checkpoints from different replicas for the same sequence number, then this encrypted checkpoint can be marked as *stable*: even if f replicas sending checkpoints are malicious, and k immediately become disconnected/unavailable, at least $f+1$ correct replicas still remain that can help another replica catch up to this checkpoint.

Upon collecting a *stable* checkpoint for a given sequence number, a replica may safely garbage collect stored updates and checkpoints for all prior sequence numbers (similar to [12] and others), as long as it has also fully executed all sequence numbers up through the sequence of the stable checkpoint (note that since data center replicas do not participate in generating client responses, they consider an update to be fully executed as soon as it is ordered).

When a replica detects that it has fallen behind (e.g. because it went through proactive recovery, or was disconnected and missed some updates), it submits a state transfer request to Prime for ordering. When this request is ordered, the other replicas (including data center replicas) execute it by sending the recovering replica their *stable* encrypted checkpoint, *digests* for any correct checkpoints they have with sequence numbers higher than the stable checkpoint, and the list of ordered, encrypted client requests with sequence numbers between the *stable* checkpoint and the global sequence number of the state transfer request from this recovering replica.

In order to catch up to the latest state, the recovering replica waits to receive a set of state transfer responses such that it has (1) a *correct* checkpoint, with at least $f + 1$ matching checkpoints/digests to guarantee its validity, and (2) a set of updates such that for every sequence number between its latest correct checkpoint and its target recovery ordinal (i.e. the sequence number at which its state transfer request was ordered), it has $f + 1$ identical updates from distinct replicas. Once these requirements are met, if the recovering replica is a data center replica, then it simply stores the latest correct checkpoint and all following correct updates in the already encrypted format. If the recovering replica is an on-premises replica, it additionally decrypts the encrypted checkpoint and the list of client requests, and then applies the decrypted checkpoint and client requests in order of increasing global ordinals to bring its application state up to date.

D. Key Renewal

As described so far, a single on-premises compromise can leak encryption keys. If the keys are sent to a data center replica, it will be able to decrypt all following updates and checkpoints. While system operators could recover from such a situation (if it was detected) by manually backing up the system, taking replicas down, bringing them back up with new keys, and re-instantiating the system, this is a labor intensive operation that is likely to require system downtime.

Therefore, we extend the basic protocol with an automatic key renewal mechanism that, combined with proactive recovery, limits the amount of confidential state a compromised on-premises replica can disclose. The basic idea is that on-premises servers maintain a separate shared symmetric encryption key and shared pseudorandom function key for each client in the system, and a given client key pair (encryption key + pseudorandom function key) is only valid for a fixed, predetermined range of client update sequence numbers. When servers get near the end of the range of sequence numbers the current client key pair is valid for, they each (independently) randomly generate a new pair of keys and propose their generated key pair by injecting it into Prime for ordering together with the proposed client sequence number range it should be valid for.

Since all correct replicas observe the ordered stream of messages from Prime in the same way, they can use the global ordering of proposals to determine the new key pair in a consistent way. For example, we can determine the keys as a combination of the first $f+1$ proposals, guaranteeing that it includes random input from at least one correct replica, so that the process cannot be controlled solely by malicious replicas. Since the replica distribution process described in Section IV-B guarantees that $f+1$ correct on-premises are always available under our threat model, it is always possible to collect $f + 1$ proposals, so this approach is live. A correct server will not agree to inject a client message for ordering (i.e. will not generate its signature share as described in Section V-A) unless it has received $f+1$ valid proposals ordered by Prime for the sequence range covering that message and thus determined the correct key to use for encryption.

While this process allows replicas to agree on new keys to use for encrypting client updates such that all (correct) replicas will apply the same new keys starting at the same client sequence number, there are still several issues to resolve to provide well defined confidentiality guarantees.

Encrypting Key Proposals. First, the new key proposals themselves must also be encrypted, since they are disseminated to data center replicas as part of the ordering process. It is not possible to avoid storing these updates at the data center replicas for exactly the same reason that data center replicas must store general client updates: in order to ensure continuous availability under our threat model, on-premises replicas that have been disconnected and are rejoining the system must be able to recover the state and resume executing updates based only on input from the data center replicas.

But, what keys can we use to encrypt the key proposal messages? Clearly, it is not safe to use the previous client encryption key, since the purpose of the key refresh is to recover from the case where the previous key was compromised. But, if some other key is used, then rejoining/recovering on-premises replicas must be able to recover that key from the data centers, which means that key needs to be stored in encrypted form, and we have the same problem again.

To solve this issue, we rely on a hardware-based root of trust. We assume each on-premises replica is configured at the time the system is set up with a shared symmetric encryption key that can only be accessed from within trusted hardware (e.g. TPM or Intel SGX [40]) and persists across reboots. An attacker who compromises a server but does not have physical access may use the key for encryption while it has access to the machine, but cannot exfiltrate, modify, or delete the key. This permanent key is used to encrypt new key proposals: with this approach, they cannot be decrypted by data center replicas (or external observers), but can be decrypted by recovering/rejoining on-premises replicas (without requiring the recovering/rejoining replicas to retrieve keys from data

center replicas). We note that this assumption of a limited degree of trusted hardware is not an unreasonable requirement, as proactive recovery already requires each replica to maintain a persistent hardware-based (TPM) asymmetric private signing key that it uses to authenticate itself and establish new session-level signing keys during its recovery process.

Adapting State Transfer. Given that key proposal messages will eventually be garbage collected, we must also extend state checkpoints to additionally include the current encryption and pseudorandom function keys for each client and their validity periods (i.e. the highest sequence number they can be used for), as well as any valid pending key proposal messages. By *pending* key proposal message, we mean a key proposal that has been ordered, but not yet used to generate a new key, as not enough proposals for the same client and validity period were ordered before the checkpoint was taken. With this extension, checkpoints are also encrypted using the hardware-protected symmetric key (although it is also possible to treat checkpoints as another logical client, with a new session-level key agreed on for each checkpoint. In this case, it is only necessary to encrypt the part of the checkpoint containing the session keys with the persistent hardware-protected key).

Limiting Disclosure. Finally, in order to guarantee limits on the amount of confidential state a compromised on-premises replica can expose, we must ensure that compromised replicas cannot control the selection of *future* encryption keys that will be used after they have gone through the proactive recovery process and been restored to a correct state. To do this, we enforce that new key proposals are only accepted as valid if they are introduced at the correct logical time. That is, we define a sequence number slack parameter x that represents how far in advance of the sequence range a key is intended to be active for it can be proposed. For example, if we consider $x = 10$ and a key validity period of 100 updates, a new key proposal for range 101-200 will not be considered as valid (and included in the computation of the actual new key) unless it is ordered *after* update 90 for the relevant client in the global total ordering created by Prime. Since all correct replicas observe the ordered stream of updates in the same way, all will make the same decision as to a key proposal's validity.

An additional concern may be that a compromised replica could, while it is compromised, generate, encrypt, and sign proposals for future client sequence numbers, and send them to a malicious external collaborator to inject at the appropriate time. However, since such messages are required to be signed with the replica's session-level signing key, which is refreshed following a proactive recovery, this is not a problem.

Our key renewal procedure does not provide complete confidentiality (in the sense of Definition 3) in the presence of a compromised on premises replica, but it limits the damage such a replica can do. In particular, for a client key validity period V and slack parameter x, it guarantees that any keys leaked by a compromised replica will only be able to decrypt a maximum of $V + x$ updates per client that are issued *after* the replica is recovered (of course, the compromised replica may leak all updates issued while it is compromised). In addition, since checkpoints are encrypted with keys that cannot be exfiltrated from their physical machine, no checkpoint constructed after the replica is recovered can be decrypted using keys it leaked while compromised. Thus, as long as replicas are periodically proactively recovered and clients continue to issue updates, the system will eventually return to a situation where its state is fully confidential, if no new on-premises compromises occur. Unfortunately, this does not apply to state manipulation algorithms: since those are likely to change rarely, once a replica with access to those algorithms is compromised, we can no longer provide guarantees of their confidentiality.

VI. Confidential Spire Implementation

We have implemented our architecture and protocols in Confidential Spire, a SCADA system for the power grid that provides the Safety, Bounded Delay, and Confidentiality guarantees defined in Section III-C under the threat model stated in Section III-B. Our Confidential Spire implementation is built on the open source Spire version 1.2 [9], which implements the architecture described in [4], and provides Safety and Bounded Delay (but not Confidentiality) under our same threat model.

In Confidential Spire, SCADA control centers serve as the on-premises sites, and the clients submitting updates to the system are Remote Terminal Units (RTUs) and Programmable Logic Controllers (PLCs) that interact with the power grid equipment, and Human Machine Interfaces (HMIs) that operators use to issue commands and view the system state.

The Spire 1.2 implementation already includes a SCADA master application and RTU/PLC proxies. Its system components communicate over the Spines intrusion-tolerant network [34], and updates are ordered using the Prime intrusion tolerant replication engine [38]. An intrusion-tolerant communication library (the Intrusion-Tolerant Reliable Channel, or ITRC) manages communications between client proxies and the control center servers, as well as between Prime and the SCADA Master application.

A. Confidentiality-Preserving Intrusion Tolerant Middleware

Confidential Spire adapts and extends Spire's intrusion-tolerant communication library into a Confidentiality-Preserving Intrusion-Tolerant Middleware (CP-ITM). While the CP-ITM serves the same basic functions as Spire's ITRC, it additionally supports encryption and decryption of client updates, the creation (and encryption) of periodic checkpoints, and a new checkpoint-based state transfer protocol. The CP-ITM is intended to be a generic middleware that can handle client communication and state management/transfer for any application.

B. Encryption and Decryption Details

The CP-ITM encrypts client requests before injecting them into Prime and decrypts them before delivering them to the SCADA Master application. For each client, the CP-ITM

maintains a shared symmetric encryption key and a pseudo-random function key (which can be periodically refreshed as described in Section V-D, though this is not yet implemented). To encrypt a request, the CP-ITM generates a hash-based message authentication code (HMAC) based on the update request itself and the shared pseudorandom function key for that client, following the approach of [30]. Then, the client update request is encrypted using AES-256 in CBC mode with this HMAC as the initialization vector (IV) and the client's shared encryption key.

Since the encryption key and pseudorandom function key for each client are shared across all control center CP-ITM instances, they all generate the same encrypted result for a client request by using the above method. We note that even if a client issues the same request multiple times, it will not result in the same encrypted output, as the client sequence number is included in the message content over which the HMAC is generated and in the content that is encrypted. The CP-ITM can decrypt encrypted content using the shared encryption key for that client and the IV (HMAC) which is included in the message header as cleartext.

C. Checkpointing and State Transfer Implementation

When the CP-ITM running in a control center replica determines that a new checkpoint is needed (i.e. that C updates have been ordered since the previous checkpoint), it requests the SCADA master to package and send back a snapshot of the current state of the system. Before the CP-ITM multicasts this checkpoint to other replicas, it encrypts the checkpoint and the associated ordered sequence number using the same method as described in Section VI-B. Every CP-ITM instance maintains an additional shared pseudorandom key and encryption key (in addition to the client key pairs) for encrypting and decrypting the checkpoints (which can be hardware-protected, as discussed in Section V-D). In this way, all control center CP-ITM instances can independently generate identical encrypted checkpoints.

When a replica requires a state transfer, its CP-ITM collects the correct encrypted checkpoint and the correct set of updates following the protocol in Section V-C. When the CP-ITM is done collecting, if it is running on a data center replica, then it simply stores the encrypted checkpoint and updates and continues operations in normal status. However, if the CP-ITM is running on a control center replica, it decrypts and sends the correct checkpoint to the SCADA Master to apply, and then decrypts and sends each collected update request in the order of their sequence numbers to the SCADA Master. Finally, it does the same with any new ordered encrypted client requests that were pending while waiting to collect state, and resumes normal operations.

VII. Evaluation

We first evaluate the overhead of providing confidentiality in our approach by comparing our Confidential Spire implementation to Spire 1.2 [9], and then evaluate our implementation's performance under particular types of attacks.

For all experiments, we emulate a power grid SCADA setup with control centers and data centers spanning about 250 miles of the US East Coast. Experiments are conducted in a local area network, but latencies between sites are emulated to reflect this geographic distribution. We emulate ten power grid substations each injecting updates via proxies at a rate of one per second per substation.

A. Performance Overhead of Confidentiality

To assess the performance overhead of our approach, we compare Confidential Spire to Spire 1.2 in two different configurations: one tolerating one compromised replica ($f = 1$) and one tolerating two compromised replicas ($f = 2$). Both configurations additionally tolerate a proactive recovery and disconnected site to support our full threat model.

We consider configurations using two control center sites and two data center sites, as these were shown to be the most practical for Spire [4]. The 4-site configurations are also the most reasonable for Confidential Spire, as they allow us to use fewer total replicas compared to configurations using only one data center, but using additional data center sites beyond two does not provide further benefits, due to the requirements on the number of replicas per control center (see Table I).

Therefore, for the $f = 1$ configurations, we evaluate configuration "3+3+3+3" (three replicas in each of 2 control centers and 2 data centers) for Spire 1.2 and configuration "4+4+3+3" (four replicas in each of 2 control centers and 3 replicas in each of 2 data centers) for Confidential Spire. For tolerating 2 simultaneous intrusions, we use the "5+5+5+4" configuration for Spire 1.2, and the equivalent "6+6+5+4" configuration in Confidential Spire. We ran each configuration for 1 hour and report the resulting update latencies in Table II.

From the results for the $f = 1$ configurations, we can see that Confidential Spire adds a small constant latency overhead of about 2ms. This increase in overhead is small because it avoids adding any new wide-area communication on the critical path compared with Spire 1.2. While Confidential Spire requires control center replicas to cooperate to generate a threshold signature on each incoming client request, it is always possible for a replica to collect the needed $f + 1$ signature shares from replicas within its own site, since each on-premises site contains $2f + 2$ replicas. Hence, Confidential Spire only utilizes the local-area network for the added communications. The sum of computational overhead, to compute the signatures and to encrypt/decrypt the requests, and the local-area network communications overhead is small compared to the multiple rounds of wide-area network message exchanges needed for the agreement protocol. While Confidential Spire also adds computation and communication for checkpoint creation and exchange, this occurs off the critical path of request processing and thus does not have a significant effect on latency.

In the $f = 2$ case, we can see that Confidential Spire's "6+6+5+4" configuration adds somewhat more overhead, increasing average latency by about 6.8ms as compared to Spire's "5+5+5+4" configuration. This is about 3.5 times the

	f	Setup	Avg Latency	% <100ms	% <200ms	0.1 percentile	1 percentile	50 percentile	99 percentile	99.9 percentile
Spire	1	3+3+3+3	51.7 ms	100	100	39.7 ms	41.0 ms	51.7 ms	62.4 ms	63.9 ms
	2	5+5+5+4	54.4 ms	100	100	42.5 ms	43.6 ms	54.4 ms	65.6 ms	67.7 ms
Confidential Spire	1	4+4+3+3	53.6 ms	100	100	41.6 ms	42.8 ms	53.6 ms	64.2 ms	66.1 ms
	2	6+6+5+4	61.2 ms	100	100	46.0 ms	47.5 ms	61.1 ms	78.4 ms	86.2 ms

TABLE II
SPIRE AND CONFIDENTIAL SPIRE PERFORMANCE ON LAN WITH EMULATED LATENCIES BETWEEN SITES FOR 36000 UPDATES OVER 1 HOUR

average latency increase noted above in the $f = 1$ case. This can be explained by increasing communication overheads, due to the all-to-all communication patterns, as the number of replicas increases. However, this is still acceptable, as the results show that our Confidential Spire implementation still meets the timeliness requirements for power grid SCADA systems (processing updates within 100ms), even while tolerating 2 intrusions. The observed 99.9 percentile latency is 86.2ms, and no update crossed the 100ms threshold. We expect that these can even be further reduced through improved engineering of the communication protocols to reduce the rate at which traffic is sent, and of the cryptographic mechanisms to reduce processing overheads. As shown by [36] and noted in [4], the majority of the advantages of proactive recovery can be obtained by tolerating two intrusions, instead of only one, making "6+6+5+4" a useful configuration to support.

B. Attack Evaluation of Confidential Spire

We next evaluate Confidential Spire's ability to meet the timeliness requirements of power grid SCADA systems while under attack. Such systems require responses within 100ms in the normal case but may tolerate latencies up to 200ms in certain situations [10], [11]. We consider the effect of proactive recovery on performance, as well as network attacks that cause a site to be disconnected. While we do not explicitly evaluate malicious actions by protocol replicas, we note that many types of malicious actions closely resemble proactive recovery of the leader replica in terms of performance: once the leader takes a malicious action (e.g. sending conflicting messages), a view change is triggered to elect a new leader.[2] The Confidential Spire "4+4+3+3" configuration's performance under all combinations of recoveries and site disconnections is illustrated in Figure 2. We can see that proactive recovery of a leader replica, which occurs between 1:00 and 1:30, causes one client update to spike over 100ms, when the system must perform a view change. Recovery of a non-leader replica (the more common case), which occurs between 3:15 and 3:45, has almost no impact on performance. In this case, we only see one client update with higher than average latency, but it is still below the 100ms threshold.

Similarly, there is no latency spike when we disconnect a non-leader site, at 4:19, since there is no view change. However, we can see a few client updates spiking, with one rising above 100ms, though still under 200ms, when the leader-site is disconnected at 2:00, since this requires a view change. We also note a small (but still acceptable) increase in average latency for the duration of the time either the leader or non-leader site is disconnected in this experiment, as in both cases it renders the fastest quorum of replicas unavailable, and requires communication with a more distant site to occur on the critical path. However, when reconnecting a disconnected site, we can see significant latency spikes, crossing the 200ms threshold and reaching up to 450ms (e.g. at 2:30 and 5:00). This is due to the large number of checkpoint messages and update messages being sent over the network from the correct replicas to help catch up the lagging replicas in the site which just rejoined the network. While this is a limitation of our current implementation, we note that this is not an inherent limitation of the protocol, and should be fixable by engineering a better message flow control for the checkpoint messages and update messages that are being sent to catch up the other replicas. Overall, these results indicate that, even with the overhead of providing confidentiality, our system can provide the necessary performance, even while under attack.

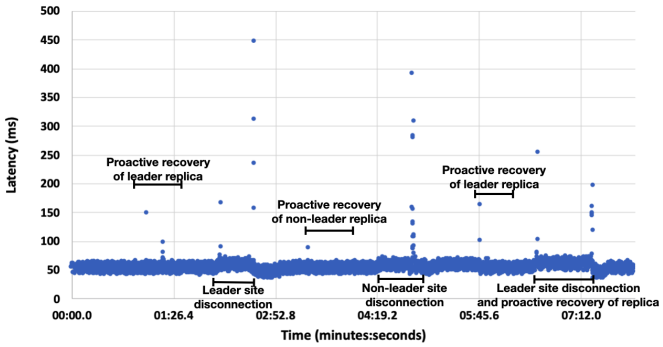

Fig. 2. Latency in the presence of proactive recoveries and site disconnections.

VIII. CONCLUSION

We presented a new partially cloud-based BFT architecture that can leverage offsite data centers to tolerate simultaneous network attacks and system compromises, without exposing confidential system state or proprietary algorithms to data center servers. In case on-premises servers are compromised, we also extended our basic protocol to include a key renewal mechanism that limits the amount of confidential information that can be disclosed. We implemented and evaluated our new model in Confidential Spire, a SCADA system for the power grid, and found that our new architecture meets SCADA timing requirements with only a small increase in overall latency to provide confidentiality guarantees.

[2]It is possible to reduce the performance impact of proactive recovery by preemptively changing the leader, but since our implementation does not do that, our experimental results accurately reflect the case where timeouts must expire before changing the leader.

References

[1] W. Zhao and F. E. Villaseca, "Byzantine fault tolerance for electric power grid monitoring and control," in *Int. Conf. Embedded Software and Systems*, July 2008, pp. 129–135.

[2] N. A. C. Medeiros, "A fault- and intrusion- tolerant architecture for EDP distribuicao SCADA system," Master's thesis, Univ. of Lisbon, 2011.

[3] J. Kirsch, S. Goose, Y. Amir, D. Wei, and P. Skare, "Survivable SCADA via intrusion-tolerant replication," *IEEE Trans. Smart Grid*, vol. 5, no. 1, pp. 60–70, Jan 2014.

[4] A. Babay, T. Tantillo, T. Aron, M. Platania, and Y. Amir, "Network-attack-resilient intrusion-tolerant SCADA for the power grid," in *48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, Luxembourg City, Luxembourg, June 2018, pp. 255–266.

[5] A. Nogueira, M. Garcia, A. Bessani, and N. Neves, "On the challenges of building a BFT SCADA," in *48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2018, pp. 163–170.

[6] A. Bessani, J. Sousa, and E. E. P. Alchieri, "State machine replication for the masses with BFT-SMART," in *44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2014, pp. 355–362.

[7] "bft-smart," https://github.com/bft-smart, retrieved 2020-11-23.

[8] A. Babay, J. Schultz, T. Tantillo, and Y. Amir, "Toward an intrusion-tolerant power grid: Challenges and opportunities," in *IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, Vienna, Austria, July 2018, pp. 1321–1326.

[9] "Spire: Intrusion-tolerant SCADA for the power grid," http://www.dsn.jhu.edu/spire/, retrieved 2020-11-24.

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

[11] J. Deshpande, A. Locke, and M. Madden, "Smart choices for the smart grid," *Alcatel-Lucent Technolgy White Paper*, 2011.

[12] M. Castro and B. Liskov, "Practical Byzantine fault tolerance and proactive recovery," *ACM Trans. Comput. Syst.*, vol. 20, no. 4, pp. 398–461, Nov. 2002.

[13] M. Correia, N. F. Neves, and P. Verissimo, "How to tolerate half less one byzantine nodes in practical distributed systems," in *IEEE Int. Symp. Reliable Distributed Systems (SRDS)*, Oct 2004, pp. 174–183.

[14] B.-G. Chun, P. Maniatis, S. Shenker, and J. Kubiatowicz, "Attested append-only memory: Making adversaries stick to their word," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 6, pp. 189–204, Oct. 2007.

[15] M. Correia, N. Neves, and P. Verissimo, "BFT-TO: Intrusion tolerance with less replicas," *The Computer Journal*, vol. 56, no. 6, pp. 693–715, June 2013.

[16] G. S. Veronese, M. Correia, A. N. Bessani, L. C. Lung, and P. Verissimo, "Efficient byzantine fault-tolerance," *IEEE Transactions on Computers*, vol. 62, no. 1, pp. 16–30, Jan 2013.

[17] Y. Amir, B. Coan, J. Kirsch, and J. Lane, "Byzantine replication under attack," in *IEEE Int. Conf. Dependable Systems and Networks (DSN)*, June 2008, pp. 197–206.

[18] A. Clement, E. Wong, L. Alvisi, M. Dahlin, and M. Marchetti, "Making byzantine fault tolerant systems tolerate byzantine faults," in *USENIX Symp. Networked Syst. Design and Implem. (NSDI)*, 2009, pp. 153–168.

[19] G. S. Veronese, M. Correia, A. N. Bessani, and L. C. Lung, "Spin one's wheels? byzantine fault tolerance with a spinning primary," in *IEEE Int. Symp. Reliable Distributed Systems (SRDS)*, Sept 2009, pp. 135–144.

[20] Z. Milosevic, M. Biely, and A. Schiper, "Bounded delay in byzantine-tolerant state machine replication," in *IEEE Int. Symp. Reliable Distributed Systems (SRDS)*, Sept 2013, pp. 61–70.

[21] T. Roeder and F. B. Schneider, "Proactive obfuscation," *ACM Trans. Comput. Syst.*, vol. 28, no. 2, pp. 4:1–4:54, Jul. 2010. [Online]. Available: http://doi.acm.org/10.1145/1813654.1813655

[22] P. Sousa, A. Bessani, M. Correia, N. F. Neves, and P. Verissimo, "Highly available intrusion-tolerant services with proactive-reactive recovery," *IEEE Trans. Parallel Distrib. Syst.*, vol. 21, no. 4, pp. 452–465, 2010.

[23] A. N. Bessani, E. P. Alchieri, M. Correia, and J. S. Fraga, "DepSpace: A byzantine fault-tolerant coordination service," in *Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008*, ser. Eurosys '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 163–176. [Online]. Available: https://doi.org/10.1145/1352592.1352610

[24] R. Padilha and F. Pedone, "Belisarius: BFT storage with confidentiality," in *IEEE 10th International Symposium on Network Computing and Applications*, 2011, pp. 9–16.

[25] R. Vassantlal, "Confidential BFT state machine replication," Master's thesis, Universidade de Lisboa, 2019. [Online]. Available: http://hdl.handle.net/10451/40304

[26] A. Bessani, M. Correia, B. Quaresma, F. André, and P. Sousa, "Depsky: dependable and secure storage in a cloud-of-clouds," *Acm transactions on storage (tos)*, vol. 9, no. 4, pp. 1–33, 2013.

[27] A. N. Bessani, R. Mendes, T. Oliveira, N. F. Neves, M. Correia, M. Pasin, and P. Verissimo, "Scfs: A shared cloud-backed file system." in *USENIX Annual Technical Conference*. Citeseer, 2014, pp. 169–180.

[28] D. R. Matos, M. L. Pardal, G. Carle, and M. Correia, "Rockfs: Cloud-backed file system resilience to client-side attacks," in *Proceedings of the 19th International Middleware Conference*, 2018, pp. 107–119.

[29] J. Yin, J.-P. Martin, A. Venkataramani, L. Alvisi, and M. Dahlin, "Separating agreement from execution for byzantine fault tolerant services," in *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, ser. SOSP '03. New York, NY, USA: Association for Computing Machinery, 2003, p. 253–267. [Online]. Available: https://doi.org/10.1145/945445.945470

[30] S. Duan and H. Zhang, "Practical state machine replication with confidentiality," in *IEEE 35th Symposium on Reliable Distributed Systems (SRDS)*, 2016, pp. 187–196.

[31] M. Kapritsos, Y. Wang, V. Quema, A. Clement, L. Alvisi, and M. Dahlin, "All about eve: Execute-verify replication for multi-core servers," in *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*. Hollywood, CA: USENIX Association, Oct. 2012, pp. 237–250. [Online]. Available: https://www.usenix.org/conference/osdi12/technical-sessions/presentation/kapritsos

[32] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolić, S. W. Cocco, and J. Yellick, "Hyperledger fabric: A distributed operating system for permissioned blockchains," in *Proceedings of the Thirteenth EuroSys Conference*, ser. EuroSys '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3190508.3190538

[33] D. Obenshain, T. Tantillo, A. Babay, J. Schultz, A. Newell, M. E. Hoque, Y. Amir, and C. Nita-Rotaru, "Practical intrusion-tolerant networks," in *IEEE Int. Conf. Distrib. Comput. Syst. (ICDCS)*, June 2016, pp. 45–56.

[34] "The Spines Messaging System," www.spines.org, retrieved 2020-12-09.

[35] P. Sousa, N. F. Neves, and P. Verissimo, "Hidden problems of asynchronous proactive recovery," in *Proceedings of the Workshop on Hot Topics in System Dependability*, 2007.

[36] M. Platania, D. Obenshain, T. Tantillo, R. Sharma, and Y. Amir, "Towards a practical survivable intrusion tolerant replication system," in *2014 IEEE 33rd International Symposium on Reliable Distributed Systems*. IEEE, 2014, pp. 242–252.

[37] Y. Amir, B. Coan, J. Kirsch, and J. Lane, "Prime: Byzantine replication under attack," *IEEE Trans. Dependable and Secure Computing*, vol. 8, no. 4, pp. 564–577, July 2011.

[38] "Prime: Byzantine replication under attack," www.dsn.jhu.edu/prime, retrieved 2020-12-09.

[39] T. Tantillo, "Intrusion-tolerant SCADA for the power grid," Ph.D. dissertation, Johns Hopkins University, 2018.

[40] V. Costan and S. Devadas, "Intel sgx explained." *IACR Cryptol. ePrint Arch.*, vol. 2016, no. 86, pp. 1–118, 2016.