TEE-Assisted Recovery and Upgrades for Long-Running BFT Services

Ines Messadi ✉, Markus Elias Gerber,
Tobias Distler, and Rüdiger Kapitza

Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany
{ines.messadi,tobias.distler,ruediger.kapitza}@fau.de

Abstract. Integrating Byzantine fault-tolerant (BFT) replication with trusted execution environments (TEEs) offers an unprecedented degree of resilience and confidentiality. Nevertheless, vulnerabilities in both the underlying infrastructure and the application software still exist, which means that for long-running services there typically is a substantial risk that eventually the number of faulty or compromised replicas exceeds a system's fault-tolerance threshold. We address this issue with Naboris, the first approach in the area of confidential computing to provide long-term resilience for critical BFT services. To achieve this, Naboris combines (1) proactive recovery of replicas to remove faults with (2) support for software upgrades to patch newly discovered vulnerabilities at runtime. For both of these procedures, Naboris afterwards provides remote entities with verifiable evidence that they actually took place. We implement Naboris using AMD SEV-SNP and show that its performance overhead is low compared to the state of the art in recovery procedures.

Keywords: Byzantine Fault Tolerance · Trusted Execution Environments · Long-Term Resilience · Recovery · Replica Evolution.

1 Introduction

Byzantine fault-tolerant (BFT) replication [29, 40] is an effective technique to build systems that provide their service even if a subset of its replicas (usually up to f out of $3f+1$ [15]) are subjected to arbitrary faults caused by hardware errors, software bugs, or intrusions. However, by itself Byzantine fault tolerance is not able to address additional aspects that are vital for practical deployments such as privacy or defense measures to reduce the probability of successful attacks.

In an effort to mitigate these issues, organizations are increasingly turning to hardware-based trusted execution environments (TEEs) to shield sensitive code and data, for example in the form of confidential virtual machines (CVMs). As a reaction to the growing demand, CVMs based on technologies such as AMD SEV-SNP and Intel TDX have already been integrated into major public-cloud platforms [3, 8], allowing customers to rent virtual machines while the provider cannot access their private data. Following this emergence of hardware-based protection, several BFT systems have adopted TEEs [10, 11, 14, 31, 39].

This is the authors' preprint version of an article to appear in the
Proceedings of the 20th International Conference on Availability, Reliability and Security (ARES '25), Ghent, 11–14 August 2025.

Sadly, TEEs are not a panacea but themselves susceptible to vulnerabilities [16,37,41,48], including software-based attacks that exploit application-level bugs (e.g., synchronization bugs [48]) and can cause the bypassing of the CVM protection [49]. Even worse, attacks are getting stealthier and detection techniques are limited, giving adversaries more time to gather information and learn from previous attacks on already compromised replicas. In such a stealthy attack on the Ronin Blockchain Network, for example, attackers recently gained undetected access for over a week, resulting in a loss of $615 million [47].

Taking these circumstances into account, for long-running services it is likely that more than f replicas become faulty or compromised during a system's lifetime. Thus, it is essential to enable such BFT systems to self-heal by automatically recovering replicas from faults [15, 38, 42]. In addition, they should offer mechanisms that allow replicas to evolve [32], meaning to dynamically modify the replica code during recovery in order to eliminate (known) vulnerabilities [26,36,43]. Unfortunately, despite some approaches specifically targeting virtualized environments [21,22,38,46], to our knowledge there are currently no solutions supporting both the recovery and the evolution of CVM-based replicas.

In this paper we address this problem with Naboris, an approach that offers resilience for long-running CVM-based services by proactively recovering replicas and enabling upgrades during execution. Since many faults and attacks are inherently difficult to identify, Naboris recovers replicas *proactively* [15] in configurable intervals. As part of the recovery process, Naboris creates an entirely new CVM instance for the affected replica containing the latest security patches, safely transfers the current agreement-protocol and application state into this instance, and refreshes the replica's cryptographic keys. This way, attackers have a reduced window of vulnerability and face a constantly changing attack surface.

To achieve flexibility with regard to deployment options, we designed Naboris to be as independent from the underlying infrastructure as possible. Ruling out the use of special-purpose hardware components [15, 42], this goal required us to develop a new methodology for ensuring that recoveries actually took place. More precisely, while traditional approaches rely on trusted subsystems to always execute recovery procedures in a reliable and timely manner [15, 21, 22, 38, 42], Naboris enables its replicas to present a *proof of recovery and upgrade* that is verifiable by both its peers as well as external parties (e.g., service customers). Only after a recovering replica provides such a proof, the other replicas in the system allow the replica to rejoin their group.

In summary, this work makes the following contributions: (1) It presents Naboris, the first confidential-computing-aware approach to offer both replica recovery and evolution. (2) It describes Naboris's multi-phase recovery protocol that performs replica rejuvenation with only negligible impact on overall system performance due to replacing the old CVM instance of a replica with a new CVM instance hosted on the same machine. (3) It offers details on the Naboris prototype implementation based on AMD SEV-SNP. (4) It outlines our vision of how to provide "Naboris as a service" in public-cloud environments. (5) It evaluates Naboris for multiple applications and in the presence of recovery procedures.

2 Background

In this section, we provide background on BFT state-machine replication and give an overview of how trusted execution environments are able to offer verifiable proofs of integrity in the context of confidential computing.

2.1 BFT State-Machine Replication

Our target systems achieve resilience against arbitrary faults by replicating the user application across multiple servers and executing a BFT agreement protocol (e.g., PBFT [15]) to keep the application state consistent (see Figure 1). To tolerate up to f faulty replicas, these systems require a total of $3f+1$ replicas. In the use-case scenarios we address, the fault threshold f is typically small (e.g., $f = 1$) which is why support for recovering replicas from faults is crucial.

In a nutshell, BFT state-machine replication works as follows. To invoke operations at the replicated service, clients submit requests to one of the replicas, the leader, which then initiates a multi-phase consensus protocol that enables all correct replicas in the system to agree on a common order in which to execute the requests in the application. With the application designed as a deterministic state machine [40], this approach ensures that all correct replicas produce the same replies to requests and also end up with the same application states. To garbage-collect information belonging to completed consensus-protocol runs, replicas periodically create checkpoints of the application state in predefined intervals [24]. If necessary, such checkpoints can later be used by trailing replicas to catch up with the rest of the group by performing a state transfer between replicas. In case the leader is suspected or confirmed to be faulty, the other replicas initiate a view-change mechanism that reassigns the leader role to a different replica and consequently allows the overall system to make progress again.

2.2 Trusted Execution Environments

Trusted execution environments (TEEs) promise a hardware-encrypted environment for computations beyond the observation and control of hosting entities, thereby laying the foundation for confidential computing. Depending on the technology, the CPU has the ability to protect the integrity and encrypt either specific application components [17] or entire virtual machines, which are then referred to as confidential virtual machines (CVMs) [3].

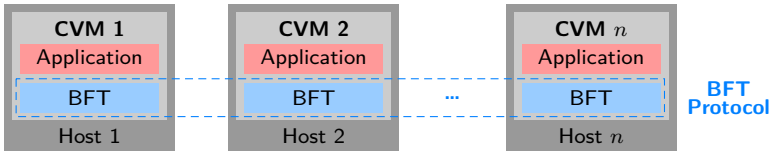

Fig. 1. BFT state-machine replication architecture for CVM-based replicas

In this paper, we focus on AMD SEV [2] as technology for CVMs because it has a lower adoption barrier than more fine-grained approaches. For the recent release of Secure Nested Paging (SNP), a CVM mitigates attacks of a malicious hypervisor. In the SEV-SNP architecture, the memory controller encrypts the initial set of VM pages using a key unique to the VM. This is monitored by a secure processor to provide essential security services for SEV-SNP. In addition, when a CVM is executed, its memory pages are decrypted by the memory controller and reencrypted when the data leaves the CPU. Any communication between the CVMs and the hypervisor occurs through shared memory pages.

SEV-SNP incorporates an attestation mechanism that allows communicating parties to prove the TEE's genuineness. Attestation starts with the CVM requesting an *attestation report* (including the initial measured pages) from the secure processor. The report is signed with the versioned chip endorsement key (VCEK) that is unique to the secure processor firmware and a unique chip ID to ensure that the report originated from a specific machine and that the system is properly patched. In addition, the verification process also involves a remote key distribution service (KDS) that supplies certificates to authenticate the VCEK's validity. Existing works have proposed a variety of options when it comes to leveraging TEEs for enhancing resilience and confidentiality in BFT systems. Some approaches like Engraft [46] and CCF [39] encapsulate consensus and transaction handling while maintaining untrusted storage and network layers. Others aim to strengthen resilience by reducing the trusted computing base, protecting only safety-critical components (e.g., message signing [11,31]).

2.3 System and Threat Model

As illustrated in Figure 1, with this work we target systems in which there is a clear separation between, on the one hand, the customer application running inside CVMs and, on the other hand, the underlying host infrastructure, which is typically provided by a different entity. For fault tolerance, the application is replicated using a BFT protocol that ensures safety in the presence of at most f concurrent Byzantine replica faults, and liveness under partial synchrony [15]. We assume that by exploiting an existing vulnerability in the implementation of an application replica, an adversary may cause the replica to behave in an arbitrary way. For this reason, despite running replicas inside TEEs, our target systems cannot employ TEE-based BFT protocols such as Hybster [11] or DAMYSUS [18], as those protocols require their TEE-based components to be fully trusted. Hence, we rely on full-fledged BFT agreement provided by approaches like PBFT [15] to ensure consistency among $n = 3f + 1$ replicas.

Besides direct attacks on the application, another potential strategy of an adversary involves gaining access to the host infrastructure. If the adversary takes control over a server, we assume the TEE to ensure the confidentiality of the local replica (i.e., our approach does not offer improvements regarding (rollback or side-channel) attacks against the TEE itself [33,44,45]). However, during these kinds of attacks the replica is no longer guaranteed to make progress; as discussed in Section 7, such scenarios commonly do not pose a major problem in practice.

3 Challenges

Next, we identify the main challenges associated with ensuring resilience for long-running CVM-based BFT services and highlight how NABORIS addresses them.

3.1 Long-Term Resilience for CVM-based Services

As discussed in Section 2.1, the replicated BFT systems we target in our work typically consist of a small number of replicas, therefore making it necessary to provide some form of automated recovery to support long-running applications [20]. Specifically, the recovery of replicas ensures that faults do not accumulate over time, and hence allows a system to tolerate more than f faulty replicas over the course of its lifetime.

In the past, several strategies have been proposed to address this issue, including reactive-recovery procedures that are initiated once a fault was either detected or at least suspected [27], proactive-recovery mechanisms that are time-triggered [15,21,22,38], as well as combinations of both [42]. To further increase the effectiveness of such measures, the recovery should be combined with the concept of replica evolution [32] which entails modifications to the replica code to remove existing vulnerabilities. Among other things, this may involve the use of randomization techniques [13], recompiling replica executables to enhance the diversity of implementations [36], and substitutions of port numbers and passwords [43]. Unfortunately, despite the large body of previous works on these topics, it is still an open question how to perform replica recovery and evolution in BFT systems consisting of a group of CVM-based replicas.

$\succ$ *Our Approach in a Nutshell.* NABORIS offers long-term resilience for replicated BFT services by supporting proactive recovery of CVM-based replicas, including the opportunity to dynamically eliminate vulnerabilities at runtime by installing upgrades as part of the recovery process.

3.2 Proof of Recovery

In a nutshell, recovery procedures remove faults by replacing an existing (potentially faulty) replica instance with a new instance that is created from a verified state [20]. For this approach to be effective it is essential that an adversary must not be able to prevent the initiation, execution, and completion of the recovery process. Traditionally, this problem is addressed by introducing a trusted component that cannot be compromised through attacks and takes care of handling critical recovery steps. Examples for such components include a hardware-based watchdog timer [15], a dedicated replica-local synchronous subsystem [42], or an entire virtualization layer that is considered to be trustworthy [22,38].

For our target use cases, and especially potential future scenarios in which NABORIS is deployed as a service in a cloud environment (see Section 7), none of these options represents a practical solution due to conflicting with our goal of minimizing infrastructure dependence. Specifically, relying on any type of low-level component to control the recovery process would mean to shift additional

responsibilities to the underlying infrastructure, and hence makes it more difficult to develop a generic design. Furthermore, to our knowledge no cloud provider currently offers this kind of reliable-recovery functionality to upper layers. Thus, there is no possibility to directly force faulty replicas to perform a timely recovery under all circumstances and to the same extent as traditional approaches do.

➢ *Our Approach in a Nutshell.* To circumvent the lack of recovery support from the underlying infrastructure, NABORIS pursues a novel strategy that focuses on reliably detecting scenarios in which faulty replicas try to avoid recovery. More precisely, once a recovery is due, the other replicas temporarily exclude the affected replica from participating in the BFT protocol and only allow the replica to rejoin after presenting a proof that it actually has recovered. To create such a *proof of recovery*, we extend a CVM's attestation report with a recovery counter whose value is dictated by the other replicas and changes on each recovery.

3.3 Proof of Upgrade

Besides trying to hinder the execution of recovery procedures, an adversary may also aim at preventing upgrades in an effort to keep the replica vulnerable, and consequently make it easier to compromise a replica even after a recovery. To solve this problem, replicas should not only prove that a recovery indeed took place, but also that they are currently running the latest firmware and software versions. For systems that do not support upgrades at runtime, the latter can be achieved by assuming that the replica code is obtained from a non-modifiable medium [15]. Otherwise, existing approaches typically resort to a trusted component to perform upgrades in a reliable and timely manner [36,43], which is why their designs do not incorporate mechanisms to dynamically validate the replica code during execution.

➢ *Our Approach in a Nutshell.* As it is the case for recovery, NABORIS does not simply rely on the assumption that a replica upgrade has actually been performed, but instead offers the other replicas in the system to verify such a scenario based on a *proof of upgrade*. In general, NABORIS replicas validate the integrity of their peers by comparing the attestation measurements of other replicas to their own, thereby ensuring that all participating replicas execute the identical (unaltered) code; if a replica on such occasion presents a diverging measurement, it is not allowed to (re)join the system. To support verifiable upgrades (which change the code of the affected replica and hence its measurement), NABORIS utilizes a special command that is sent through the agreement protocol to inform all replicas about the new measurement in a consistent manner.

4 NABORIS

NABORIS is a TEE-based framework that offers long-term resilience for critical BFT services by supporting both replica recovery as well as replica evolution. In this section, we first present an overview of our approach and then discuss the most important techniques and components in more detail.

4.1 Overview

NABORIS provides Byzantine fault tolerance by running multiple replicas of the same application on different servers and connecting them through a BFT agreement protocol (PBFT [15] in our prototype). To protect the integrity of the replicated service, each replica is located inside its own CVM. As illustrated in Figure 2, in addition to the replica CVM every participating server also accommodates an auxiliary NABORIS system component, the *recovery agent*, which is responsible for assisting in proactive-recovery procedures for its local replica. Specifically, whenever the NABORIS protocol (which itself is running inside the replica CVM) decides that the recovery of a replica is due, the recovery agent starts a second CVM on the same host which at the end of the recovery process becomes the new replica and fully replaces the old CVM. Among other things, this approach has three main advantages: (1) By creating a new replica instance in a separate CVM, from the overall replica's perspective the NABORIS recovery process is able to remove any effects that a (potential) intrusion might have had on the old CVM. (2) The use of a second CVM enables NABORIS to not only eliminate existing faults, but also close vulnerabilities by installing security patches in the new CVM. (3) Hosting both the old and the new CVM on the same server allows NABORIS to bring the application state of the new replica instance up to date via a state transfer that primarily relies on local operations [21, 22], and therefore has only negligible impact on system availability and performance.

4.2 Proof of Recovery

Since Byzantine faults and successful attacks do not necessarily result in detectable faulty behavior, NABORIS recovers replicas in a periodic and proactive manner, thereby ensuring that faults are removed after a configurable window of time. Specifically, replicas are recovered individually using a round-robin strategy to minimize the impact of recovery on overall system performance. To be compatible with different underlying infrastructures, in contrast to other approaches (see Section 3.2) NABORIS does not rely on special-purpose hardware to trigger recovery procedures but instead applies a *coordinated timer* to decide when the next recovery is due. In a nutshell, NABORIS's coordinated timer works as follows.

To participate in the timer subprotocol, replicas maintain a *recovery counter* as part of the state that is replicated using NABORIS's BFT agreement protocol. Using a deterministic mapping to replica ids, the current value *recno* of this

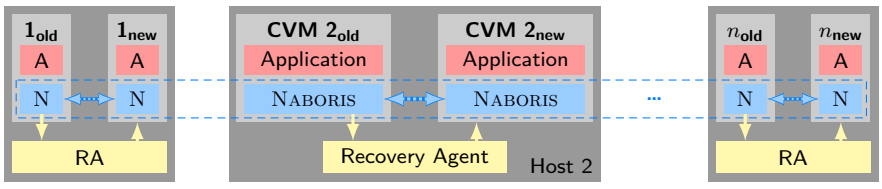

Fig. 2. NABORIS replication infrastructure

counter indicates which replica i to recover next (e.g., $recno \,\%\, n = i$, with n denoting the total number of replicas). At the beginning of every recovery period, each replica starts a local timer that is set to the end of the period (i.e., when the next recovery should take place). Once its local timer expires, a replica signals this event by distributing a message demanding the recovery counter to be incremented. Combining such notifications from $f+1$ different replicas (i.e., at least one correct replica), the current leader creates a special request and passes it through the agreement protocol. When this request is committed, the execution of the request causes the recovery counter to be updated, thereby informing all replicas (including the one to be recovered) about the fact that the recovery is due. At this point, the other replicas exclude the recovering replica from their group and only allow the replica (more precisely: its new CVM instance running on the same machine) to rejoin after supplying them with a proof of recovery.

To produce such a proof of recovery for Naboris, we extend a CVM's attestation report by adding the corresponding recovery-counter value. Specifically, we instruct the hypervisor to add the recovery-counter value as *host data* during the startup of the new CVM. Once such host data is set for a CVM, it is included in all of its attestation reports and cannot be changed by either the host or the CVM. Correct replicas only accept an attestation report if it contains the expected recovery-counter value. This way, and as a consequence of the report of a CVM being immutable at runtime, the old replica CVM is unable to rejoin the group by impersonating the new instance.

A replica's recovery agent (i.e., the Naboris infrastructure component running on each host server, see Section 4.1) learns about the need to perform a recovery by acting as a special BFT-protocol client that periodically retrieves the current recovery-counter value from the replica group. Based on the knowledge of the deterministic mapping of counter values to replica ids, a recovery agent is able to determine when the time has come to recover its local replica. In particular, this approach offers two key benefits: (1) Triggering a recovery by evaluating the recovery counter allows the recovery agent to directly obtain the counter value to use for the configuration of the new CVM instance. (2) Having the recovery agent fetch the recovery-counter value by sending a read request to the replica group ensures that faulty replicas are unable to prevent recovery procedures.

4.3 Proof of Upgrade

To enable Naboris replicas to evolve by applying upgrades at runtime, we first put each replica in the position to verify the integrity of other replicas, and then further extend this mechanism to support dynamic changes.

Verifiable Integrity Overall, system integrity in Naboris builds on three principles: (1) a verifiable boot chain ensures the integrity of each replica CVM right from the beginning, (2) a running code-hash validation enables each replica to verify the integrity of the code that is currently executed by another replica, and (3) a read-only filesystem prevents adversaries from retroactively modifying system files even in case of successful intrusions.

The SEV-SNP attestation process is limited to measuring the initial VM context before the VM begins execution. The context is typically used only to place and measure the initial firmware (Open Virtual Machine Firmware, OVMF), which is sufficient to continue booting a VM. This is a significant weakness because it allows adversaries, for example, to load a malicious kernel undetected because only the firmware is reflected in the attestation report. To mitigate this risk, the initialization and boot process was updated to include the kernel, initrd, and command-line hashes as part of the OVMF, referred to as Direct-Measured Boot. However, this still leaves the system vulnerable to runtime attacks (e.g., privilege escalation) where a privileged attacker could replace critical system binaries or files, persisting across reboots. For this reason, for NABORIS we extend the verified boot process to protect the entire CVM state against tampering [25]. Apart from the hashes incorporated during the directly measured boot, the root hash of a Merkle tree created from the root file system is also added. Upon the start of the CVM, the OVMF validates the hashes while loading the kernel and initrd. Furthermore, the initrd mounts the root file system and verifies its integrity using `dm-verity`. To secure and persist CVM runtime data across reboots, we provide support for encrypted file systems using `dm-crypt`. The required disk encryption key is wrapped and unwrapped using the sealing key provided by the AMD secure processor. The wrapped key is stored on the disk in an unencrypted partition and is therefore persisted and accessible across reboots. The sealing key is unique to the CVM measurement and machine identification and can only be retrieved by a running CVM with that particular measurement. Therefore only the CVM can access the encrypted file systems.

Supporting Upgrades During periods without upgrades, a NABORIS replica is able to validate the integrity of other (recovering) replicas by comparing their attestation measurements to its own. However, with NABORIS recovering replicas using a round-robin strategy, this is no longer possible once an administrator applies a patch to fix a discovered vulnerability, because changes to the code result in diverging attestation reports. To address this issue, when performing upgrades NABORIS administrators not only update the replica code but also submit a corresponding command to the replica group (via the BFT replication protocol) that informs replicas of the new attestation value to use for the measurement comparison. This way, NABORIS enables upgrades while at the same time allowing continuous integrity verification.

4.4 Recovery Protocol

We use the established PBFT-PR [15] protocol as basis for the NABORIS recovery mechanism and adapt it to fit our needs. In the following, we highlight the differences to PBFT-PR and describe the NABORIS recovery phases in detail.

Properties In order to be applicable to our target use cases, the NABORIS recovery protocol offers improvements over PBFT-PR with regard to several properties that are summarized in Table 1 and further discussed next.

Table 1. Comparison between PBFT-PR and Naboris

Aspect	Property	PBFT-PR [15]	Naboris
Recovery strategy		Proactive	Proactive
Code protection	Ⓤ	ROM	Read-only rootfs
Rejuvenation	Ⓐ	Same server	New CVM
Trusted component	Ⓒ	Secure co-processor	CVM protection
Recovery trigger	Ⓒ	HW watchdog	Coordinated timer

Ⓤ *Upgrade Support* As part of the recovery process, PBFT-PR ensures the integrity of the operating-system and application code by verifying the code based on a digest that was stored in read-only memory prior to the system start. Since this digest cannot be modified, PBFT-PR does not support dynamic code changes and hence makes it impossible to fix newly discovered vulnerabilities. In contrast, Naboris replicas validate the integrity of their peers by comparing attestation measurements and allow an administrator to consistently update the expected measurement, thereby enabling upgrades while the system is running.

Ⓐ *Availability* In PBFT-PR, a replica spans a whole server and is recovered by rebooting the physical machine. As a consequence, the recovery of a PBFT-PR replica is typically associated with a significant period of time during which the replica is not able to regularly participate in the system. For services requiring high availability, this only leaves two options: counting the recovering replica as a temporary fault with regard to the fault-tolerance threshold f (and hence being able to tolerate fewer actual faults during this period), or extending the system with additional replicas at the cost of an increased resource footprint [42]. In Naboris, we circumvent this problem by implementing the recovery as a switch from the old replica CVM to a new replica CVM hosted on the same physical machine. As a key benefit, this allows Naboris to keep the old replica CVM running while the new replica CVM is started, initialized, and brought up to date, thereby minimizing unavailability to a negligibly short period during the local switch.

Ⓒ *Cloud Readiness* Involving the reboot of an entire physical machine, the recovery process in PBFT-PR is triggered by a trusted hardware-based watchdog timer. For potential cloud-based use-case scenarios (see Section 7), neither the reboot of an entire physical machine nor the reliance on a special-purpose hardware component is feasible, especially for approaches that seek to offer provider independence. To enable the use of Naboris in these kinds of environments, we designed the Naboris recovery protocol in such a way that it can be operated within CVMs using standard cloud infrastructure services (e.g., starting and terminating CVMs) and without requiring additional specialized hardware.

Recovery Phases As illustrated in Figure 3, Naboris's protocol for recovering replicas consists of several phases. Once the replicas' local timers signal that the next recovery is due, the replicas ① execute the coordinated-timer subprotocol to trigger the recovery in a consistent manner (see Section 4.2). In response to this, ② the old replica CVM instance saves its volatile application state to disk;

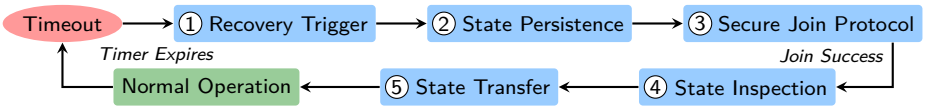

Fig. 3. NABORIS recovery protocol phases

this data serves as basis for the new replica CVM instance, which is subsequently started by the recovery. If the old replica instance is faulty and fails to comply in this phase, the recovery agent fetches the necessary data from another replica. Once the new replica instance is running, ③ it provides the other replicas with its proof of recovery (and upgrade) in order to rejoin the group. After that, with assistance from the group, ④ the new replica instance executes a subprotocol that allows the replica to clear faulty residue that the old replica instance potentially left on disk. Finally, in a last step ⑤ the new instance fetches recently updated application-state parts from other replicas to compensate for the state changes that occurred while the replica was recovering, before eventually returning to normal operation. In the following, we discuss each of these phases in detail.

Recovery Trigger The expiration of the timer in each replica marks the recovery start. At this point, NABORIS replicas execute a consensus round to establish the new recovery-counter value for the recovering replica (see Section 4.2). For this purpose, each replica sends an $\langle$INIT-REC, $recno_i + 1\rangle$ message to propose a value to the leader. The leader waits for matching INIT-REC messages from $f + 1$ different replicas (possibly including its own) to start the agreement. This prevents both accidental and malicious recovery triggers from individual replicas. Upon reaching an agreement, replicas notify clients to reject messages from the recovering replica, and then set up a timeout to wait for the rejuvenated replica to reconnect. If the replica fails to return within this interval, the protocol triggers failure handling (e.g., sending a blame signal to the administrator), as this situation implies a problem with the underlying infrastructure that is beyond NABORIS's control. Finally, the retreating replica instance sends a $\langle$SIGNAL, *recno*$\rangle$ message with the newly agreed counter value to the recovery agent to inform the agent about the value to use in the attestation report of the new replica instance.

State Persistence After the recovery is triggered, the old replica instance writes its state (including consensus and application data) to an encrypted disk. This step occurs for two reasons that are explained in the following.

Firstly, in order to ensure protocol safety, it is essential that the recovery process does not cause correct replicas to become faulty by losing their states [15]. More precisely, due to the correctness of BFT agreement protocols depending on the decisions made by a quorum of correct replicas, after recovery a correct replica must remember all statements it made to other replicas before the recovery took place. In NABORIS, we fulfill this condition by having the old replica instance flush all necessary state to disk and afterwards attaching this disk to the new replica instance. Notice that the aforementioned requirement does not apply

to faulty replicas. Specifically, if the state of a faulty replica is lost, or a malicious replica refuses to persist its current state, the new replica instance may be initialized with an older checkpoint (e.g., provided by another replica in the system).

Secondly, this approach allows to leverage the fact that in many use-case scenarios the replicated application already maintains large parts of its state on disk. As a consequence, forwarding the state via the same medium represents an efficient method to load the data into the new replica instance.

Secure Join Protocol Upon initialization by the recovery agent, the newly started replica instance generates a fresh pair of keys and broadcasts a JOIN-REQUEST message (Algorithm 1, Line 6). The other replicas validate the joining replica by comparing its measurement against their attestation values (Lines 21-27). As part of the process, this verification requires reproducible CVM builds to ensure consistent measurements across all replicas, as discussed in Section 4.3. Since the join request contains the versioned chip endorsement key, there is no need for the other replicas to perform a query to the key distribution service (see Section 2.2). A replica accepts a join request provided that: (1) the key digest in the join request report matches the computed digest of the public key pk_{irej}; (2) the recovery number $recno_i$ matches the agreed-upon number in the current view; (3) the report data in $attReport_{irej}$ matches the report data from at least $2f+1$ replicas in the current view; (4) the SNP certificate chain verifies correctly using the $vcek_{irej}$ included in the join-request message.

Algorithm 1 Secure Join Protocol

```
1: as a rejuvenated replica
2:     upon connection to all do
3:         pk_i ← generatePublicKey()
4:         attReport_i ← requestReport(Hash(pk_i))
5:         vcek_i ← requestVcek(attReport_i)
6:         send ⟨JOIN-REQUEST, attReport_i, pk_i, vcek_i⟩ to all
7:         verify remote replicas' report
8:     upon receiving ⟨JOIN-RESPONSE, status, recno, c, p, v⟩ do
9:         if status = accept then
10:            (hm, view_i) ← estimateHm(c, p, v)      ▷ Get high water mark and view estimate
11:            joinProcessingRequests()
12:        else                                                    ▷ status = reject
13:            send ⟨SIGNAL, recno⟩ to agent                       ▷ Use correct recno
14:        end if
15: as a remote replica
16:    upon receiving ⟨JOIN-REQUEST, attReport_irej, pk_irej, vcek_irej⟩ do
17:        keyHash ← getHash(attReport_irej)
18:        recno ← getRecCounter(attReport_irej)
19:        c ← ∅
20:        p ← ∅
21:        if keyHash = Hash(pk_i) ∧ verifyCertifChain(vcek) ∧ recno = agreedRecno then
22:            c ← lastCheckpointSeqNum()
23:            p ← lastPreparedReq()
24:            status ← accept
25:        else
26:            status ← reject
27:        end if
28:        send⟨JOIN-RESPONSE, status, recno, c, p, view_i⟩ to rejuvenated replica
```

State Inspection Once accepted back into the group, the new replica instance examines the state that the old replica instance supplied via encrypted disk as part of the state-persistence phase. For this purpose, the new instance requests the corresponding disk encryption key (`dm-crypt` key) from the old instance, which, upon successful attestation verification, transmits this key and afterwards instructs the recovery agent to shut it down. Similar to the scenarios discussed above, if the old instance is faulty and does not cooperate, NABORIS resorts to a stable checkpoint provided by another replica to initialize the new instance.

Apart from being non-cooperative, a faulty old instance may also try to trick the new instance into using a corrupted state. To handle such cases, NABORIS executes a mechanism that closely resembles PBFT-PR's estimation subprotocol [15], which is why we omit specifics. In a nutshell, this subprotocol enables the new instance to determine an upper bound (in the form of a high water mark hm, see Algorithm 1, Line 10) up to which the old instance at most may have contributed to the consensus progress. Consequently, if the provided state contains information beyond this point, the old instance must have deliberately planted it there, meaning that the new instance can safely ignore it.

State Transfer If during the state-inspection phase the new instance determines that parts of its state are corrupted or missing, the new instance in a final step initiates a secure state transfer to repair/obtain the affected partitions. As part of this procedure, other replicas not only provide the requested state parts but also verifiable proof (in the form of quorum certificates that are backed up by $2f+1$ replicas [15]) of the transmitted data's correctness. Once the state transfer finishes, the recovery process of the replica is complete.

4.5 Discussion

As discussed in Section 3.2, our decision to forgo trusted recovery support from the underlying infrastructure in order to achieve a more flexible design may result in scenarios in which a timely recovery cannot always be guaranteed if an adversary managed to launch a successful intrusion of the host machine. Specifically, there may be situations where the recovery of a replica does not proceed as expected or is delayed due to a malicious actor at the host level, causing the recovering replica to become unavailable. If this happens to more than f hosts at the same time, the service is no longer live, however its safety is still ensured even under such circumstances. We argue that such cases of delayed recovery can be detected early by monitoring the NABORIS protocol's control messages using a special monitoring client. If a replica fails to recover within a specified time, this is detected by the monitoring client and the administrator is contacted to resolve the problem at the host-system level.

Having compromised a host, an adversary may try to exploit CVM reboots to launch forking attacks that violate state continuity, which can create divergent replica groups serving inconsistent responses to clients. To prevent such attacks, we can adopt Narrator's blockchain-based approach [35], where each CVM registers its unique identifier (provided by the AMD secure processor) with a trusted

tamper-proof blockchain during initialization, or let the leader dictate the CVM instance the group should communicate with. In AMD CVMs, each fork is a new instance for which the platform security processor generates a unique `REPORT_ID`. By registering this identifier, other replicas can detect forked identities.

5 Implementation

Our implementation builds on the PBFT components of the Themis codebase [5], reusing its core consensus functionality and extending it with our proactive recovery implementation. We realized the Naboris prototype in 2,802 lines of code (LOC), the PBFT-PR [15] protocol (which serves as baseline in our evaluation, see Section 6) in 2,297 LOC, and the agent logic using the Rust nightly 2023-04-01. To digitally sign protocol messages, we rely on the 256-bit ED25519 from the ring library (v0.16.20). The attestation functionality in our prototype is embedded as a Rust module (304 LOC) using the VirTEE API [6].

Naboris and PBFT-PR handle replica recoveries through different mechanisms. Naboris involves the recovery agent and starts new replica instances with the recovery-counter value determined by the replica group. Besides, SEV-SNP CVMs are architecturally restricted from performing internal reboots. In contrast, PBFT-PR replicas autonomously shut down and restart using `systemd` services on the host to maintain CPU pinning configurations across reboots.

Providing CVMs via AMD SEV-SNP Naboris's prototype runs inside a CVM built using a modified Revelio assembling scripts as a base [25]. The root filesystem is constructed through a Docker-based build process that incorporates an SEV-SNP-capable kernel and other essential packages. The build process eliminates nondeterministic elements to guarantee reproducible measurements across different instances. The resulting CVM integrates additional mechanisms: `dm-verity` for root filesystem integrity verification in the initramfs and `dm-crypt` for disk encryption. Furthermore, we added a service started at boot time to request the necessary VCEK from the KDS keyserver in order to enable the validation of attestation reports. Some configuration information (e.g., basic network settings) that is not critical to the integrity of the CVM but needed for a proper integration into the system is injected via a separate partition that is not part of the measurement of the CVM. During transition to the rejuvenated CVM, the retreating CVM's disk is mounted as removable storage, enabling state recovery by the rejuvenated CVM.

Can Naboris be Implemented Based on Other CVM Technologies Such as Intel TDX? The Naboris protocol is CVM-agnostic and can be realized using other TEE architectures that provide similar confidentiality and integrity properties, host-provided data (i.e., for the recovery counter), and trusted timers using accurate instruction timing (i.e., recovery triggers). In our case, the

AMD trusted-timer feature [2] queries timestamps from the AMD secure processor using encrypted communication, preventing time manipulation attacks. Furthermore, using a distributed consensus between CVMs, our design strengthens the timing guarantees. To enable verifiable recovery counters in attestation reports, we leverage `host_data` in AMD SEV-SNP. Intel TDX provides equivalent functionality via its measurement registers (`mrconfigid`, `mrowner`, and `mrownerconfig`) [30]. For cloud-based deployments (see Section 7), some features in Azure such as the configuration of `host_data` in the interface for creating SEV-SNP CVMs may be restricted, however this is not a fundamental problem as the `host_data` setting could be made available to customers.

6 Evaluation

In this section, we evaluate Naboris using PBFT-PR (see Section 4.4) as baseline. For this purpose, we conduct experiments with two different applications: (1) A counter (incremented on each client request) to measure the overhead of the recovery protocol in isolation. (2) An in-memory key-value store (KVS) that operates at a workload of 50%/50% read/write operations using 10,000 keys.

In our evaluation, we primarily focus on examining how Naboris performs compared to a system (PBFT-PR) which also supports proactive recovery, but offers lower security guarantees than Naboris. In this context, it is important to note that we do not claim Naboris to be a means for improving throughput or latency in TEE-based replicated systems, which is why our experiments neither include protocols that exploit TEEs for enhancing the performance of Byzantine agreement [11,18], nor approaches that speed up the recovery process by optimizing the state transfer between old and new replica instance [21,22]. As discussed in Section 3, our goal in this paper is to provide proofs of recovery and upgrade for CVM-based replicas, hence in the evaluation we are interested in studying whether Naboris is able to achieve this without notable performance overhead.

6.1 Experimental Setup

We evaluate Naboris using virtual machines with SEV-SNP-enabled kernels, comparing it against PBFT-PR running on virtual machines without SEV-SNP. Each virtual machine is configured with 4 vCPUs and 8 GiB memory, running Ubuntu 20.04. To optimize performance, each virtual machine is pinned to cores sharing the same physical CPU to prevent context switching. Both prototypes comprise 4 replicas and utilize a thread pool of 4 workers implementing the work stealing algorithm from the Tokio library, where networking and message authentication are parallelized while maintaining sequential protocol execution.

Client requests carry 100-byte payloads and the agreement on them is performed in batches of 50 requests. In all experiments, the checkpointing interval is set to 1,000 requests and the workload is produced by 100 client threads. Each client continuously issues synchronous requests and measures the time it takes

after the submission of a request to obtain a stable result. During the experiments, a new recovery procedure is triggered 40 seconds after the completion of the previous recovery procedure. Before a leader replica is recovered, the replica group initiates a view change to reassign the leader role to another replica.

6.2 Impact of Recoveries on Performance

The results of our experiments in Figures 4 and 5 show that the recovery process in NABORIS requires approximately 64 seconds on average from start to finish, while PBFT-PR completes recovery in 19 seconds. This difference is due to several factors: Firstly, CVM boot times are 220% higher compared to VM boot times in non-SEV environments due to the additional memory pre-allocation and initialization required for SEV-SNP CVMs. Secondly, the NABORIS recovery protocol (see Section 4.4) involves additional steps compared to PBFT-PR, including VM disk mounting and secure exchange of disk encryption keys. However, despite the longer recovery period NABORIS maintains stable throughput outside of view changes even during recoveries. For both NABORIS and PBFT-PR, the temporary drops in throughput (and the associated spikes in latency) that are observable in the graphs (e.g., at $t = 45$ seconds in Figure 4) are a result of the systems performing a view change prior to recovering the active leader replica. In a production setting, the windows between two recoveries are typically significantly longer than in our experiments (e.g., up to several hours), which is why the higher duration of NABORIS's recovery procedure does not pose a problem, especially due to the fact that over the course of this process NABORIS is able to provide normal-case performance.

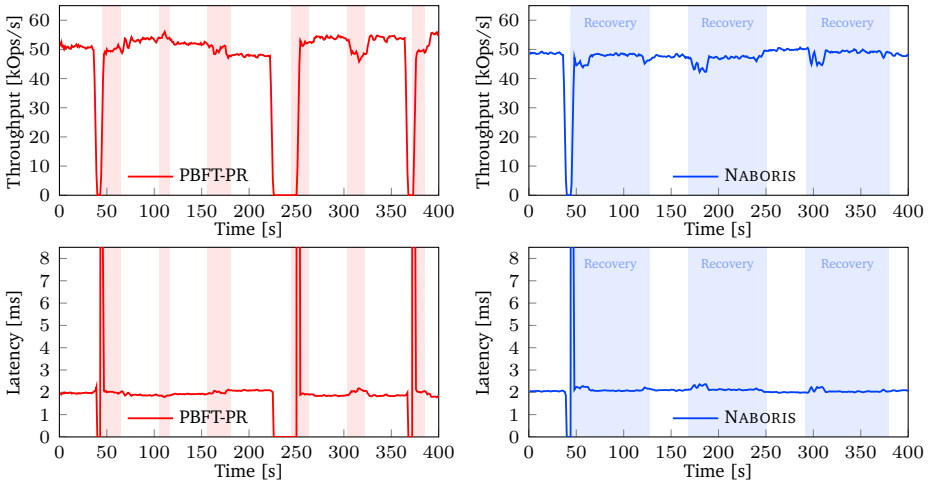

Fig. 4. Comparison of throughput and latency results for PBFT-PR (left) and NABORIS (right) during and outside of recovery procedures for the counter application.

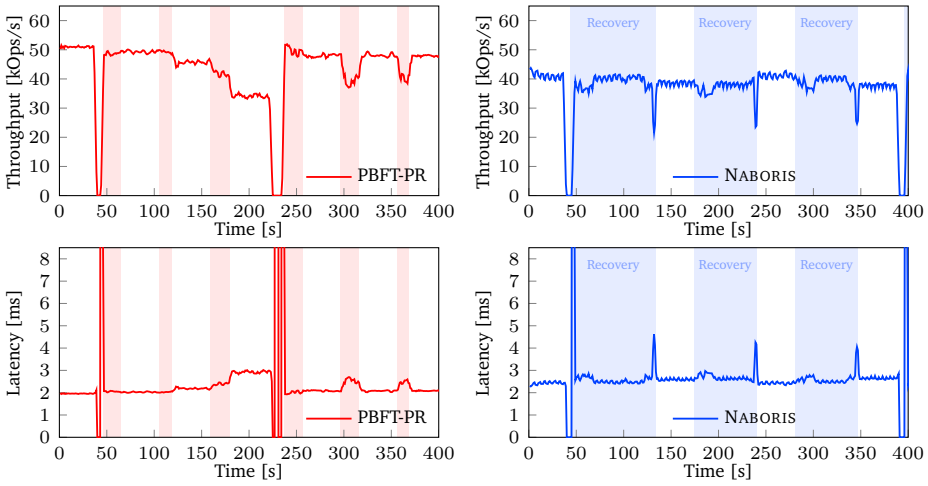

Fig. 5. Comparison of throughput and latency results for PBFT-PR (left) and NABORIS (right) during and outside of recovery procedures for the key-value store.

6.3 The Cost of SEV-SNP CVM Mechanisms

To better understand the difference between NABORIS and PBFT-PR, we measure the average latency of the protocols' main phases. Table 2 presents a detailed comparison. The total boot time shows that SEV-SNP CVMs have approximately 3.2× higher latency than non-SEV VMs, primarily due to the additional memory initialization and pre-allocation. The most significant performance difference appears in the key-exchange mechanism. NABORIS's attestation exchange requires 8 ms, compared to PBFT-PR's new-key message which takes about 53 µs (99% faster). The attestation message also carries a larger payload (876 bytes total: 748-byte report, 32-byte key, 96-byte VCEK) compared to PBFT-PR's simple 32-byte public-key message.

At boot time, a NABORIS replica must request a report from the AMD secure processor and in principle a VCEK from the key distribution service. The key distribution service introduces rate limiting when multiple similar CVMs request VCEKs running on the same host and it could be unavailable. However, the VCEK of a machine does not change unless the firmware of the SVE-SNP is upgraded, so the VCEK can be prefetched, cached and included in attestation messages. The duration of the estimation protocol varies based on replica load and message processing queues, particularly after performing a view change.

Updating the entire CVM requires about 5 minutes in which the VM is reassembled from scratch including rebuilding the base Docker image. However, this occurs outside the critical path by pre-preparing assembled VMs. Similarly, the one-time partition decryption and mounting cost of 1,217 ms (including a 1 s sleep to wait for decryption) is only necessary because NABORIS, in contrast to PBFT-PR, is designed for use in confidential-computing environments.

Table 2. Comparison of system functions

Primitive	NABORIS	PBFT-PR
Booting Time	32 s	10 s
Pre-kernel	7 s	4 s
Integrity protection (`dm-verity`)	4 ms	-
Disk encryption (`dm-crypt`)	33 ms	33 ms
Attestation vs. new-key message	8 ms	53 µs
Verifying report	4 ms	-
Requesting report from AMD-SP	833 ms	-
Requesting VCEK from KDS	794 ms	-
Decryption and mount partition	1,217 ms	-
Saving state in files (KVS)	464 ms	123 ms
Estimation protocol	20 ms	10 ms
Re-assembling the VM (Upgrades)	5 m	-

7 NABORIS as a Service

With more and more critical applications being moved to the cloud, we envision that it can be beneficial to provide the NABORIS functionality as a cloud service to customers who need long-term resilience for their applications. For this purpose, it is important to avoid vendor lock-in by not relying on a specific provider so that the NABORIS service is able to run on different clouds. Fortunately, by designing NABORIS in such a way that it requires the underlying infrastructure to only offer basic operations like starting and terminating CVMs, from a technical perspective there are no additional barriers in this regard. Hence, in the following we focus our discussion on the question how provider independence can be achieved. The key to solve this is to take the special characteristics of cloud environments and its stakeholders into account, which we do by distinguishing between different roles that each are associated with a distinct set of privileges and responsibilities. Specifically, we separate the tasks that are related to the cloud infrastructure ($\rightarrow$ cloud provider) from the tasks necessary to perform replica recovery and evolution ($\rightarrow$ NABORIS operator).

Cloud Provider The cloud provider's central task is to maintain and operate the underlying infrastructure, offering the trusted execution environment that NABORIS relies on for confidentiality and integrity. For this purpose, the cloud provider does not require any knowledge about NABORIS in general, or of the recovery procedures and the replicated application in particular. In fact, as a result of the trusted execution environment ensuring confidentiality, the application state is completely hidden from the cloud provider.

If the cloud provider fails to preserve the availability of its infrastructure (e.g., by unilaterally shutting down CVMs), the liveness of the replicated service can no longer be guaranteed, however even in this case NABORIS's BFT agreement protocol ensures that safety is not at stake. Overall, we do not expect provider-induced liveness issues to become a major problem in practice for two reasons. On the one hand, cloud providers typically do not have an incentive to violate service level agreements, because doing so hurts their reputation and consequently their business. On the other hand, if the dependability of an in-

dividual cloud provider is of concern, NABORIS's provider-independent design makes it possible to distribute a NABORIS deployment across different clouds in order to reduce the reliance on a specific provider [12].

NABORIS Operator The NABORIS operator leverages the cloud infrastructure to offer NABORIS as a service to its customers. In this context, the operator is primarily responsible for running the NABORIS recovery agent associated with each replica and for performing timely upgrades to the replica CVMs to fix critical bugs. To ensure that a newly started CVM is co-located with the old CVM of the same replica, the operator may either rent a bare-metal instance [7] or rely on a dedicated host service [1,4]. As with the cloud provider, the trusted execution environment prevents the NABORIS operator from learning application secrets.

Using a CVM's attestation report measurement, customers (and other external entities) are able to validate the code by accessing a public repository that features a deterministic build process, and hence allows them to recompute the measurement offline. This way, they can verify whether upgrades were actually performed, and if necessary (i.e., when the replica code does not match the expected version) blame the NABORIS operator for not meeting its responsibilities.

8 Related Work

We consider works on recovery and rejuvenation of Byzantine fault-tolerant systems and approaches that build on TEEs to implement a hybrid fault model and support recovery as most relevant to NABORIS.

BFT Fault Recovery and Rejuvenation The problem of replica recovery with the goal of evicting adversaries has been studied in previous works [15, 22,38,42]. Castro and Liskov's work [15], PBFT-PR, inspired the development of several approaches in this area. PBFT-PR relies on trusted subsystems and watchdogs to proactively rejuvenate replicas one by one after a predefined time. As discussed in Section 4.4, the NABORIS approach shares similarities with the protocol when triggering a stateful recovery, but offers advantages with regard to properties such as upgrade support, availability, and cloud readiness.

VM-FIT [38] proposes a virtualization-based recovery initializing a new virtual machine replica in parallel to the normal execution to minimize the recovery overhead. SPARE [22] represents a design with $f + 1$ active replicas handling request processing and voting, while f passive replicas in a paused state receive periodic state updates; during proactive-recovery procedures, the passive replicas serve as foundation for the new replica instances. Sousa et al. [42] explored a reactive-recovery scheme introducing wormholes as trusted subsystems that coordinate rejuvenation, an approach that demands additional resources. Other systems, like Dynamic BFT [23], treat recovery as a reconfiguration problem, replacing faulty replicas rather than simply restarting them.

Various evolution techniques modify replica configurations across recoveries to prevent attackers from repeatedly compromising recovered replicas. These include code obfuscation [34] and parameter updates [43]. Building on this concept of evolution, our system generates a new CVM instance with distinct configurations after each recovery. The Eternal system [32] enables live software upgrades, where one replica can be upgraded while others maintain service availability. Compared to these works, NABORIS is the first to propose proactive recovery and upgrade support using commodity hardware support for TEEs such as CVMs.

Recovering TEE-based Replication Systems Several works have explored protecting consensus protocols using TEEs [10,28,46]. However, they commonly assume that code running inside a TEE cannot be exploited and focus on handling crash failures and their consequences, particularly state recovery and rollback attack prevention. Dinis et al. [19] propose a restart-rollback model for state replication during TEE restarts, with replicas verifying state freshness through digest comparisons. Engraft [46] encapsulates Raft consensus in TEEs and uses a two-phase protocol for state recovery after restarts. Nimble [9] prevents rollback attacks using trusted TEE endorsers that maintain signed ledger states and require majority consensus. While CCF [28] provides disaster recovery through ledger recovery and node restart protocols, its recovery model is purely reactive, initiating only after failures occur. NABORIS assumes that the hardware and firmware support implementing the TEE mechanisms can only fail by crashing, but imposes no further restrictions regarding Byzantine failures. Hence, it has a smaller trusted computing base than previous TEE-based replication systems.

9 Conclusion

NABORIS provides long-term resilience for CVM-based BFT services by supporting both proactive recovery and dynamic software upgrades. In contrast to existing recovery approaches, NABORIS does not rely on trusted special-purpose (hardware) components, but instead applies a novel concept that builds on the idea of only allowing recovering replicas to rejoin the system after having presented verifiable proof (in the form of CVM attestation reports) confirming that recoveries and upgrades indeed took place.

Acknowledgements This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 446811880, 541017677.

References

1. Amazon EC2 dedicated hosts, https://aws.amazon.com/ec2/dedicated-hosts/
2. AMD secure encrypted virtualization (SEV). https://developer.amd.com/sev/
3. AMD SEV-SNP on Amazon EC2 instances, https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/sev-snp.html

4. Azure dedicated host, https://azure.microsoft.com/en-us/products/virtual-machines/dedicated-host
5. Themis: BFT framework in Rust, https://github.com/ibr-ds/themis
6. VirTEE: Virtualized Trusted Execution Environments, https://github.com/virtee
7. What is BareMetal infrastructure on Azure? https://learn.microsoft.com/en-us/azure/baremetal-infrastructure/concepts-baremetal-infrastructure-overview
8. Announcing the public preview of Azure confidential VMs with Intel TDX (2023), https://azure.microsoft.com/en-us/updates/confidential-vms-with-intel-tdx-dcesv5-ecesv5-public-preview/
9. Angel, S., Basu, A., Cui, W., Jaeger, T., Lau, S., Setty, S., Singanamalla, S.: Nimble: Rollback protection for confidential cloud services. In: OSDI '23 (2023)
10. Bailleu, M., Giantsidi, D., Gavrielatos, V., Quoc, D.L., Nagarajan, V., Bhatotia, P.: Avocado: A secure in-memory distributed storage system. In: ATC '21 (2021)
11. Behl, J., Distler, T., Kapitza, R.: Hybrids on steroids: SGX-based high performance BFT. In: EuroSys '17 (2017)
12. Bessani, A., Correia, M., Quaresma, B., André, F., Sousa, P.: DepSky: Dependable and secure storage in a cloud-of-clouds. ACM Transactions on Storage **9**(4) (2013)
13. Bhatkar, S., DuVarney, D.C., Sekar, R.: Address obfuscation: An efficient approach to combat a board range of memory error exploits. In: USENIX Security '03 (2003)
14. Brandenburger, M., Cachin, C., Kapitza, R., Sorniotti, A.: Trusted computing meets blockchain: Rollback attacks and a solution for Hyperledger Fabric. In: SRDS '19 (2019)
15. Castro, M., Liskov, B.: Practical byzantine fault tolerance and proactive recovery. ACM Transactions on Computer Systems **20**(4) (2002)
16. Chen, Z., Vasilakis, G., Murdock, K., Dean, E., Oswald, D., Garcia, F.D.: VoltPillager: Hardware-based fault injection attacks against Intel SGX enclaves using the SVID voltage scaling interface. In: USENIX Security '21 (2021)
17. Costan, V., Devadas, S.: Intel SGX explained. IACR Cryptology ePrint Archive **2016**(86) (2016)
18. Decouchant, J., Kozhaya, D., Rahli, V., Yu, J.: DAMYSUS: Streamlined BFT consensus leveraging trusted components. In: EuroSys '22 (2022)
19. Dinis, B., Druschel, P., Rodrigues, R.: RR: A fault model for efficient TEE replication. In: NDSS '23 (2023)
20. Distler, T.: Byzantine fault-tolerant state-machine replication from a systems perspective. ACM Computing Surveys **54**(1) (2021)
21. Distler, T., Kapitza, R., Reiser, H.P.: State transfer for hypervisor-based proactive recovery of heterogeneous replicated services. In: SICHERHEIT '10 (2010)
22. Distler, T., Popov, I., Schröder-Preikschat, W., Reiser, H.P., Kapitza, R.: SPARE: Replicas on hold. In: NDSS '11 (2011)
23. Duan, S., Zhang, H.: Foundations of dynamic BFT. In: SP '22 (2022)
24. Eischer, M., Büttner, M., Distler, T.: Deterministic fuzzy checkpoints. In: SRDS '19
25. Galanou, A., Bindlish, K., Preibsch, L., Pignolet, Y.A., Fetzer, C., Kapitza, R.: Trustworthy confidential virtual machines for the masses. In: Middleware '23 (2023)
26. Garcia, M., Bessani, A., Neves, N.: Lazarus: Automatic management of diversity in BFT systems. In: Middleware '19 (2019)
27. Haeberlen, A., Kouznetsov, P., Druschel, P.: PeerReview: Practical accountability for distributed systems. In: SOSP '07 (2007)
28. Howard, H., Alder, F., Ashton, E., Chamayou, A., Clebsch, S., Costa, M., Delignat-Lavaud, A., Fournet, C., Jeffery, A., Kerner, M., Kounelis, F., Kuppe, M.A., et al.: Confidential consortium framework: Secure multiparty applications with confidentiality, integrity, and high availability. arXiv preprint arXiv:2310.11559 (2023)

29. Lamport, L., Shostak, R., Pease, M.: The Byzantine generals problem. In: Concurrency: The Works of Leslie Lamport (2019)
30. Li, X.: QEMU patch submission (2024), https://patchew.org/QEMU/20241105062408.3533704-1-xiaoyao.li@intel.com/20241105062408.3533704-15-xiaoyao.li@intel.com/
31. Messadi, I., Becker, M.H., Bleeke, K., Jehl, L., Mokhtar, S.B., Kapitza, R.: SplitBFT: Improving Byzantine fault tolerance safety using trusted compartments. In: Middleware '22 (2022)
32. Moser, L.E., Melliar-Smith, P.M., Narasimhan, P., Tewksbury, L.A., Kalogeraki, V.: Eternal: Fault tolerance and live upgrades for distributed object systems. In: DISCEX '00 (2000)
33. Murdock, K., Oswald, D., Garcia, F.D., Van Bulck, J., Gruss, D., Piessens, F.: Plundervolt: Software-based fault injection attacks against Intel SGX. In: IEEE S&P '20 (2020)
34. Padilha, R., Pedone, F.: Belisarius: BFT storage with confidentiality. In: NCA '11
35. Peng, W., Li, X., Niu, J., Zhang, X., Zhang, Y.: Ensuring state continuity for confidential computing: A blockchain-based approach. IEEE Transactions on Dependable and Secure Computing (2024)
36. Platania, M., Obenshain, D., Tantillo, T., Sharma, R., Amir, Y.: Towards a practical survivable intrusion tolerant replication system. In: SRDS '14 (2014)
37. Puddu, I., Schneider, M., Haller, M., Capkun, S.: Frontal attack: Leaking control-flow in SGX via the CPU frontend. In: USENIX Security '21 (2021)
38. Reiser, H.P., Kapitza, R.: Hypervisor-based efficient proactive recovery. In: SRDS '07 (2007)
39. Russinovich, M., Ashton, E., Avanessians, C., Castro, M., Chamayou, A., Clebsch, S., Costa, M., Fournet, C., Kerner, M., et al.: CCF: A framework for building confidential verifiable replicated services. Technical Report MSR-TR-2019-16 (2019)
40. Schneider, F.B.: Implementing fault-tolerant services using the state machine approach: A tutorial. ACM Computing Surveys **22**(4) (1990)
41. Shih, M.W., Lee, S., Kim, T., Peinado, M.: T-SGX: Eradicating controlled-channel attacks against enclave programs. In: NDSS '17 (2017)
42. Sousa, P., Bessani, A.N., Correia, M., Neves, N.F., Verissimo, P.: Highly available intrusion-tolerant services with proactive-reactive recovery. IEEE Transactions on Parallel and Distributed Systems **21**(4) (2010)
43. Sousa, P., Bessani, A.N., Obelheiro, R.R.: The FOREVER service for fault/intrusion removal. In: WRAITS '08 (2008)
44. Van Bulck, J., Minkin, M., Weisse, O., Genkin, D., Kasikci, B., Piessens, F., Silberstein, M., Wenisch, T.F., et al.: Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution. In: USENIX Security '18 (2018)
45. Van Bulck, J., Piessens, F., Strackx, R.: SGX-Step: A practical attack framework for precise enclave execution control. In: SysTEX '17 (2017)
46. Wang, W., Deng, S., Niu, J., Reiter, M.K., Zhang, Y.: Engraft: Enclave-guarded Raft on Byzantine faulty nodes. In: CCS '22 (2022)
47. Warren-Kachelein, D.: Crypto hackers exploit Ronin network for $615 million (2022), https://www.bankinfosecurity.com/crypto-hackers-exploit-ronin-network-for-615-million-a-18810
48. Weichbrodt, N., Kurmus, A., Pietzuch, P., Kapitza, R.: AsyncShock: Exploiting synchronisation bugs in Intel SGX enclaves. In: ESORICS '16 (2016)
49. Zhang, R., Center, C.H., Gerlach, L., Weber, D., Hetterich, L., Lü, Y., Kogler, A., Schwarz, M.: CacheWarp: Software-based fault injection using selective state reset. In: USENIX Security '24 (2024)