AtomicDisk: A Secure Virtual Disk for TEEs against Eviction Attacks

Hongliang Tian, *Ant Group;* Xinyi Yu, *NICE Lab, Xiamen University;* Shaowei Song and Qingsong Chen, *Ant Group;* Zhihao Zhang and Shiyu Wang, *NICE Lab, Xiamen University;* Weijie Liu, *Nankai University;* Erci Xu, *Shanghai Jiao Tong University;* Shoumeng Yan, *Ant Group;* Yiming Zhang, *NICE Lab, Xiamen University and Shanghai Jiao Tong University*

https://www.usenix.org/conference/fast25/presentation/tian-hongliang

This paper is included in the Proceedings of the 23rd USENIX Conference on File and Storage Technologies.

February 25–27, 2025 • Santa Clara, CA, USA

ISBN 978-1-939133-45-8

Open access to the Proceedings of the 23rd USENIX Conference on File and Storage Technologies is sponsored by

AtomicDisk: A Secure Virtual Disk for TEEs against Eviction Attacks

Hongliang Tian[1]*, Xinyi Yu[2]*, Shaowei Song[1], Qingsong Chen[1], Zhihao Zhang[2], Shiyu Wang[2], Weijie Liu[4], Erci Xu[3], Shoumeng Yan[1], Yiming Zhang[2,3]

[1]*Ant Group,* [2]*NICE Lab, XMU,* [3]*SJTU,* [4]*NKU*

Abstract

SGX-PFS is the state-of-the-art secure storage solution for Trusted Execution Environment (TEE). SGX-PFS uses Merkle Hash Trees (MHT) to achieve confidentiality, integrity, and freshness, and adopts a recovery journal to ensure consistency. Unfortunately, SGX-PFS is vulnerable to a new type of *eviction attacks*: a privileged adversary can capture transient on-disk states (referred to as *vulnerable snapshots*), which are generated by cache evictions of SGX-PFS inside the TEE (invisible and unanticipated to the user) and can potentially result in security loopholes.

Vulnerable snapshots are allowed mainly because neither the POSIX file system interface nor the block interface has constraints on the *ordering* and *timing* for the persistence of writes. To address this vulnerability, we propose a new security property called *sync atomicity*, which promises that all writes before a sync request are committed in an all-or-nothing manner. We further design a secure virtual disk (called *AtomicDisk*) by enhancing SGX-PFS. AtomicDisk achieves sync atomicity by introducing an internal `commit` operation, so that uncommitted evicted writes can be distinguished from committed synced ones, thus effectively preventing eviction attacks. We compare AtomicDisk to SGX-PFS with trace-driven workloads. SGX-PFS generates hundreds of thousands of vulnerable snapshots per trace, which can be used for eviction attacks. In contrast, AtomicDisk correctly generates exactly one valid state (caused by a sync), while achieving similar performance as SGX-PFS.

1 Introduction

Trusted Execution Environment (TEE) is a widely-adopted, hardware-based security technology, which enables users to run sensitive applications in private memory regions that cannot be snooped or tampered with by privileged adversaries. All major CPU architectures have TEE implementations [1, 2, 13, 14, 34, 40] like AMD SEV [1] and Intel SGX [13]. As TEE-enabled CPUs become mainstream, TEE is getting increasing attention from both industry and academia [35, 43, 54, 56].

While the in-memory data can be protected by TEE hardware, the on-disk data still needs to be protected by TEE software. Storage security becomes increasingly important to protect user data outside the TEE. The adversaries are *privileged*, *online*, and *active* [55], being able to monitor, tamper with, roll back, and crash the host machine. This requires the TEE software to provide a minimum security guarantees for the on-disk data of confidentiality, integrity, freshness, and consistency (CIFC).

For SEV-protected virtual machines (VMs), the go-to choice for guest OSes is the Linux kernel, but none of Linux's secure storage measures (including eCryptFs [32], fscrypt [9], dm-cyrpt [7] and dm-integrity [8]) is designed with TEEs in mind, thus falling short of the minimum security guarantees of CIFC. In contrast, SGX-protected TEE systems [3, 39, 58, 62] can adopt Intel SGX Protected File System (SGX-PFS) [21], the state-of-the-art secure storage solution, to satisfy the security requirements of CIFC. SGX-PFS uses Merkle Hash Tree (MHT) [42] to protect the in-place updated on-disk data with confidentiality, integrity, and freshness, and adopts a recovery journal to ensure consistency.

Unfortunately, we find that CIFC is not enough for protecting on-disk data against privileged adversaries. Specifically, SGX-PFS is vulnerable to the so-called *eviction attacks* [15, 23], a new attack type that is first identified by us and has been confirmed by Intel. The attacker can capture and replay transient on-disk states (referred to as *vulnerable snapshots*), which are generated by cache eviction of SGX-PFS inside the TEE and unknown to the user, to conduct malicious behaviors. To show a concrete attack example against SGX-PFS, we exploit vulnerable snapshots to gain complete access to an SGX-protected Redis server, as described in §3.

Eviction attacks exploit the inability that TEEs cannot distinguish vulnerable snapshots (which are unanticipated to the user) from *valid* on-disk states generated by sync requests (which are submitted by the user). We analyze the root cause

*Co-primary authors.

of this problem and find that vulnerable snapshots are allowed mainly because neither the POSIX file system interface nor the block interface has constraints on the *ordering* and *timing* for the persistence of writes.

To address this vulnerability at a reasonable overhead, we propose a new security property called *sync atomicity*, which promises that all writes before a sync request are committed in an all-or-nothing manner, i.e., being committed if and only if the sync is done. Sync atomicity enables the TEE to identify user-synced disk states no matter whether eviction occurs.

We further design a secure virtual disk (called *AtomicDisk*) by enhancing the MHT-protected SGX-PFS. AtomicDisk prevents eviction attacks by providing sync atomicity without changing upper-layer applications or file systems. The key idea is simple: we introduce an internal `commit` operation triggered by sync, which allows us to distinguish between cache-evicted writes (uncommitted) and user-synced writes (committed). During a crash recovery, AtomicDisk only restores the committed writes from the recovery journal, while ignoring the uncommitted ones.

We have implemented a prototype of AtomicDisk [4] , which is integrated with the Occlum library OS [58] for SGX. We compare AtomicDisk to SGX-PFS with trace-driven workloads. SGX-PFS generates hundreds of thousands of vulnerable snapshots per trace, which are vulnerable to eviction attacks. In contrast, AtomicDisk correctly generates exactly one valid state (caused by a sync) while achieving similar performance as SGX-PFS.

AtomicDisk, as well as the attack reproduction artifact, has been open-sourced [4, 18].

2 Background

2.1 Trusted Execution Environments

TEEs protect sensitive code and data from untrusted infrastructures. Modern TEEs can be classified into two categories: enclave-based TEEs (Intel SGX [13]) and VM-based TEEs (AMD SEV [1]). Enclave-based TEEs create isolated and encrypted memory regions in the address space of user-space processes, while VM-based TEEs apply isolation and memory encryption to protect a VM from a malicious hypervisor.

Enclave-based and VM-based TEEs have several common characteristics. First, they assume powerful adversaries that are privileged/online/active (§2.2). Second, they lack storage I/O protection, as storage devices are beyond the scope of TEE hardware protection and thus untrusted by TEEs. Third, TEEs support *remote attestation*: an instance of TEE can authenticate itself to a remote party by showing a hardware-endorsed certificate and subsequently establish a secure communication channel with the party. Note that the recent generations of SGX [12] can provide abundant trusted memory. Therefore, unlike prior work [27, 37, 38, 48, 65], we do not consider the capacity of SGX trusted memory as a constraint.

2.2 TEE Threat Model

In this paper, we consider a typical setting of TEE usage, where applications are ported into the TEE with no (or few) modifications thanks to a TEE-aware runtime. For SGX, a popular choice for such a runtime is library OSes [53, 58, 62]. For SEV, one can choose off-the-shelf OS kernels like Linux. Since SGX-PFS is the state-of-the-art storage solution for TEEs, in this paper we mainly focus on SGX-PFS in SGX enclaves, but our analysis and design also hold for secure storage in SEV-based secure VMs.

A secure virtual disk has the standard block I/O interface, supporting `read`, `write`, and `flush` operations. The standard `flush` causes a synchronization (sync) of all cached writes to the disk. It is desirable to protect all block I/O *transparently*, so that we can free the rest of the TEE from worrying about the security of storage I/O and enable the reuse of existing storage stacks.

All data written to or read from the secure virtual disk is in plaintext. Inside the TEE, the secure virtual disk is responsible for encrypting/decrypting the data transferred to/from the (untrusted) host block device properly. SGX-PFS performs in-place updates, allowing it to be implemented in a raw disk format. The logical block addresses (LBA) can be directly mapped to the host block addresses (HBA) by adding a simple offset without querying an index for translation.

Security assumptions. We assume that the TEE hardware is trustworthy and it protects the confidentiality and integrity of the memory along with the CPU states. We trust all software components within the TEE boundary, and assume that they do not voluntarily leak secrets and are not compromised. We assume that the software in the TEE can do remote attestation to fetch necessary secrets (e.g., the root key).

Adversary capabilities. We assume a strong adversary who is *privileged* (controlling any hardware and software outside the TEE on the host), *online* (conducting attacks at any timing throughout the lifecycle of the TEE), and *active* (having the capability of tampering with any I/O requests to and responses from the host block device).

Possible attacks include the following. (i) *Snooping attacks* attempt to learn the content of data blocks. (ii) *Tampering attacks* replace valid data blocks with invalid ones. (iii) *Rollback attacks* replace up-to-date data blocks with outdated ones. (iv) *Crashing attacks* kill TEEs abruptly to make on-disk states inconsistent. (v) The new type of *eviction attacks* will be described in §3.

There are some attacks that are out of the scope of this paper. We do not consider (i) denial-of-service (DoS) or wiper attacks [25] that affect the availability or accessibility of data; (ii) side-channel attacks [24] that infer sensitive information from side channels like disk access patterns; or (iii) *entire-disk* rollback attacks that can roll back the entire disk including the root key, which needs an $O(1)$ rollback-resistant trusted store [26, 45] for protection and is orthogonal to this paper.

Solutions	Security Attributes				
	Confid-entiality	Inte-grity	Fresh-ness	Consis-tency	Atom-icity
DM	✓	✓	✗	✗	✗
SGX-PFS	✓	✓	✓	✓	✗
AtomicDisk	✓	✓	✓	✓	✓

Table 1: A comparison between different solutions. The red ✗ represents unsatisfied security attributes. AtomicDisk is the *only* solution that satisfies all security attributes (CIFCA).

Security properties. To counter such a strong adversary, the secure virtual disk should provide a minimum security guarantees of confidentiality, integrity, freshness, and consistency (CIFC). *Confidentiality* guarantees that the user data submitted by any write is not leaked, thus preventing snooping attacks. *Integrity* promises that the user data returned from any read are genuinely generated by the user, thus preventing tampering attacks. *Freshness* ensures that the user data returned from any read are up-to-date, thus preventing rollback attacks. *Consistency* ensures that all data is consistent despite any accidental crashes or crashing attacks. As summarized in Table 1, on AMD SEV, the state-of-the-art device-mapper-based secure storage solution (dm-crypt [7] + dm-integrity [8]) can only provide confidentiality and integrity; while on Intel SGX, the state-of-the-art SGX-PFS can provide the CIFC attributes, which will be elaborated in §2.3.

More important, we propose a new security property (*sync atomicity*) to prevent the new type of eviction attacks, which will be introduced in §4.

2.3 MHT-based SGX-PFS

The state-of-the-art SGX-PFS, as well as other major TEE systems [3, 21, 39, 58, 62], adopt Merkle Hash Tree [42] to protect the on-disk data for TEE.

SGX-PFS. Although the name suggests a file system, SGX-PFS can only protect individual files (each being used as a secure virtual disk) rather than the file system. An open file of SGX-PFS consists of three components [19]: a variant of MHT, a cache, and a recovery journal. ❶ In the MHT, every node is persisted on disk with authenticated encryption. The MHT leaf nodes link to the data nodes that store the actual data, while the MHT non-leaf nodes maintain the encryption keys and message authentication codes (MAC) of their children. MHT ensures the confidentiality, integrity, and freshness of the entire file. ❷ To avoid performing disk I/O for every request, the file maintains a fixed-size cache within secure memory to store the most-recently-used nodes in the MHT. When the cache is full, dirty nodes will be demoted [64] from the cache to the storage. ❸ Before the eviction of the new-version nodes, the corresponding old-version nodes are saved in the recovery journal, which allows the file to be recovered to its previous consistent state once a crash happens during the eviction.

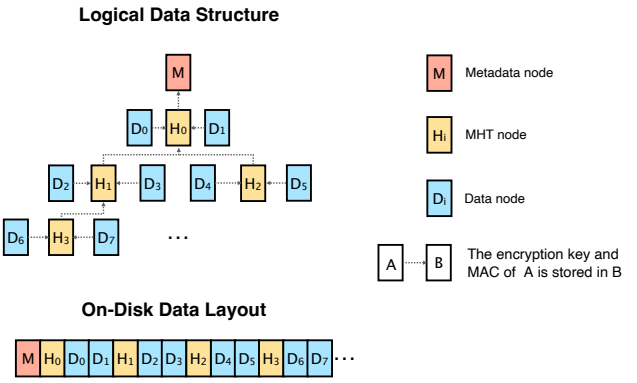


Figure 1: An improved MHT of SGX-PFS.

Merkle Hash Tree. The classic MHT is a tree structure for protecting data integrity, where every leaf node is labelled with the hash of a data block and every non-leaf node is labelled with the cryptographic hash of its child nodes. By organizing the hash values in an MHT tree, the integrity of any data block can be checked in $O(\log N)$ time where N is the number of nodes in the MHT tree. When writing a data block, the corresponding nodes from the leaf to the root of the MHT are in turn calculated (by hashing the child nodes) and updated; and when reading a data block, the corresponding nodes of the MHT are in turn read and verified.

SGX-PFS improves the standard MHT by using the AES-GCM scheme [11] (instead of simple hash) for authenticated encryption with keys and MACs, to provide not only *integrity* but also *confidentiality*. As shown in Fig. 1, the *data nodes* store the encrypted data blocks; the *MHT nodes* store the encryption keys and MACs of its child nodes; and the *metadata node* stores the file name as well as the encryption key and MAC of the root MHT node, protecting the *freshness* of the entire MHT and the data unless the metadata node is tampered by an entire-disk rollback.

Another important difference between the classic MHT and the improved MHT of SGX-PFS is the data node links. In the classic MHT, data nodes are only linked to the leaf MHT nodes; while in the improved MHT of SGX-PFS, data nodes are linked not only to the MHT leaf nodes but also to the MHT non-leaf nodes. The on-disk data layout of the improved MHT simplifies the modification to the MHT when writing new data blocks.

3 Eviction Attacks

We identify a new attack type named *eviction attacks* to TEEs. Before discussing the attack details, we first define a snapshot as the persistent state of a system at a point of time. If an enclave uses protected files of SGX-PFS to persist its states, then these protected files constitute the snapshot of the en-

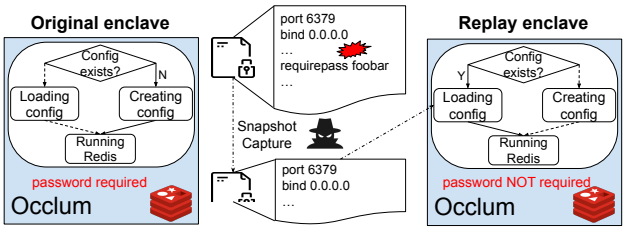


Figure 2: An example of eviction attacks on Redis running in an SGX enclave. All vulnerable snapshots are valid under the protection of the MHT.

clave. This attack is based on two key observations: First, the storage stack of a TEE may generate transient snapshots that are mostly invisible (and consequently *unexpected*) to users. This is because the POSIX standard puts little constraints on the ordering and atomicity guarantees [28], allowing the cache layer (e.g., the SGX-PFS cache) of a TEE to perform eviction at any time even without sync.

Second, in normal environments, the eviction-generated snapshots are harmless. But for a TEE, any snapshots of which the content is unanticipated to the user, can be captured by the adversary. Then, these vulnerable snapshots can be replayed to a TEE to conduct malicious behaviors and cause security loopholes.

Vulnerability in SGX-PFS. Consider SGX-PFS for an example as to why the storage stack of a TEE generates vulnerable snapshots. When the MHT cache is filled with dirty nodes and becomes full, it will evict the cache by persisting the content of all dirty nodes to the disk automatically. Each eviction results in a snapshot that is valid from the perspective of SGX-PFS yet *unanticipated* to the user.

Concrete attack. To understand the threats of the vulnerable snapshots for SGX-PFS, consider a scenario where a popular key-value store, Redis [17], is to be deployed on the cloud. For high security, the user ports Redis into an SGX enclave with the Occlum library OS [58], which features an encrypted file system that protects each file and directory with a protected file of SGX-PFS. The enclave starts in two steps: ❶ create and initialize a Redis configuration file called `redis.conf` (if it does not exist), and ❷ start the Redis server to serve user requests. The `.conf` file needs to be initialized on the fly because it contains the password of authenticated Redis clients, which need to be fetched from a trusted source on startup via remote attestation.

A concrete example of eviction attacks against the SGX-protected Redis is shown in Fig. 2. The first step of creating the `.conf` file jeopardizes the protection of SGX-PFS. This file is expected to have a configuration directive named `requirepass`, which specifies the password for logging to the server. If the directive is absent, any client can access the server *without restriction*. Unfortunately, because of the user-unaware cache eviction of SGX-PFS, the process of writing the `.conf` file may generate vulnerable snapshots where the `requirepass` directive has not yet been written. After capturing such snapshots, an adversary can restart the enclave or start a new one in an attempt to trick the victim enclave with the insecure configuration and gain complete access to the Redis server without authentication.

We have created a demo that manages to perform such attacks [23]. It deals with technical details including how to capture the vulnerable snapshots, how to apply them to another enclave, and most importantly, the exact conditions when SGX-PFS generates exploitable snapshots of the `.conf` file. In an effort of responsible disclosure, we have reported our finding to Intel, who has confirmed the vulnerability and assigned a CVE ID [15].

It is practical to exploit the vulnerable snapshots to attack more applications. Popular databases like MongoDB [16] and Cassandra [6] use similar configuration-based authentication. An adversary can exploit snapshots where authentication directives are missing or set to defaults, and then use eviction attacks to bypass authentication. A more interesting direction, which we leave for future work, is how to exploit the vulnerable snapshots that involve multiple files.

4 AtomicDisk

4.1 Overview

Eviction attacks arise from the gap between the possible on-disk states expected by an application and those allowed by the storage stack. In theory, this gap can be addressed by manipulating the entire persistent state space of an application with a transactional database. But in practice, this solution hardly works, because it is difficult, if not impossible, for programmers to figure out all possible on-disk states of a complex system and the transitions between them.

A more practical solution is to differentiate eviction-generated, vulnerable disk images from sync-generated, valid ones. To achieve this, we propose AtomicDisk with the *sync atomicity* guarantee. AtomicDisk is designed by modifying SGX-PFS. Specifically, we introduce an internal `commit` operation to identify and reject the uncommitted evicted writes upon a reboot.

4.2 Sync Atomicity

To protect against eviction attacks, we propose to enhance the semantics of sync with *sync atomicity*, which promises that all evicted writes before a sync are committed in an all-or-nothing manner, i.e., being committed if and only if the sync is done. Accordingly, the total number of valid disk images will be equal to that of the sync operations.

To enforce sync atomicity, AtomicDisk allows a data block to have two statuses, committed and uncommitted. We introduce a new internal operation called `commit` for converting

from the uncommitted status to the committed one. We extend the SGX-PFS metadata node with a new flag, `committed`, which indicates whether the disk image is full of committed data blocks or not. By default, the flag is set to `false`, as most images are triggered by evictions and contain uncommitted data. The recovery journal may store committed/uncommitted data blocks.

Similar to SGX-PFS, the writes submitted by the user to AtomicDisk are first held in the cache. An eviction occurs once the cache is full, demoting each in-cache dirty MHT node (N_c) to the disk (Fig. 1). The demoted on-disk node (N_d) is uncommitted.

If the eviction of N_d is a new write to an empty block, i.e., no overwrite occurs, then we say that N_d has an old version of N'_d = NULL. Otherwise, the eviction of N_d overwrites an existing node N'_d. Therefore, N_d always has an old version of N'_d on the disk, no matter whether it is a new write. Similar to the original SGX-PFS, we save the old version N'_d into the journal before eviction.

The journal-saved version has the same (committed/uncommitted) status as N'_d. There are two cases for the journal-saved version being committed: a new write to an empty block, or the first overwrite to a committed block. When receiving a sync, AtomicDisk triggers a `commit` operation, which (i) converts all blocks of the disk image to be committed by setting the `committed` flag of the metadata node to `true`, and (ii) removes the entire journal as its content is no longer needed. When opening a disk image after a reboot, AtomicDisk checks the `committed` flag of the metadata node. If `true`, then the open operation proceeds; otherwise, it turns to the journal for disk rollback.

The key difference between the AtomicDisk journal and the SGX-PFS journal is that the SGX-PFS journal can remove the old-version data once the eviction is done, while the AtomicDisk journal needs to keep the old-version data until receiving a sync, so that AtomicDisk can recover data to the committed status. On the downside, the AtomicDisk journal needs to keep the old-version data longer, and thus consumes more disk space than SGX-PFS.

After a crash, AtomicDisk will roll back to the most recently committed image by restoring the committed blocks in the journal while ignoring the uncommitted ones. In principle, only committed old-version blocks need to be saved to the journal. But for implementation simplicity, AtomicDisk directly saves all old-version blocks without distinguishing whether they are committed or not before eviction. This results in a simplified journal storage pattern: for each logical block, its first appearance in the journal must be committed and subsequent appearances uncommitted. During recovery, AtomicDisk reads the journal from the beginning to the end, restoring each logical block it encounters for the first time (which is committed). An in-memory bitmap is used to record the restored blocks and identifying the subsequent uncommitted blocks.

4.3 Security Guarantees

Protection graph. We establish a directed graph of cryptographic protection relationship. Blocks are the units of encryption. All blocks are protected with authenticated encryption. AtomicDisk can guarantee the security of user data as it can build protection paths from the root key to the data blocks, and thus the security of the root key implies that of user data.

Root key. The root of trust of AtomicDisk is the root key, which is securely possessed by the TEE owner and given to an AtomicDisk instance upon startup. Typically, the root key is fetched securely by the TEE via remote attestation.

Key generation. Each block is protected with a unique encryption key generated by one of the two methods: random key generator or deterministic key derivation. The key of a data block is generated randomly and saved by its parent node for future retrievals. In contrast, the key of a journal block is determined via a deterministic key derivation function, whose input includes a key derivation key (KDK) and a sequence number. For example, the KDK of journal blocks is the root key of AtomicDisk and the sequence numbers are their ever-increasing logical IDs. This simplifies key management as only the KDK needs to be saved.

Key acquisition. To open a volume protected by AtomicDisk, one needs the root key, which is typically acquired by doing remote attestation in a user program. But user programs depend on the root file system, which itself should be mounted on AtomicDisk for I/O protection. This chicken-and-egg dilemma can be solved with *early user space* [22], which is supported by both Linux and Occlum via `initramfs`, a temporary, in-memory root file system. Its content is shipped with the OS kernel and its integrity can be verified through remote attestation. AtomicDisk mounts the root file system in the early user space.

MAC management. For a data block, its MAC is kept in its parent node just like its encryption key. In contrast, each journal block has two copies of MAC. The first copy is stored as plaintext beside the encrypted journal block on the disk. The second copy is stored by its *next* journal block. That is, journal blocks are *chained*, so that AtomicDisk can validate the integrity of each journal block, and consequently, the integrity of the entire journal. The inclusion of MAC values and node numbers extends each journal block beyond the physical block size. We implement a write buffer that aggregates multiple journal blocks to optimize I/O operations. The buffer is persisted to disk when full or upon sync requests.

Snooping/tampering/rollback/crashing. The adversary cannot effectively conduct snooping, tampering, rollback, or crashing attacks, as AtomicDisk inherits the security design of SGX-PFS. However, there is a catch: the root key does not prevent journal blocks from being rolled back, but the chain of journal blocks does. The chain is formed by keeping the MAC of each journal block in the next one. Therefore, no journal blocks, except the last one, can be rolled back.

Trace names	Total write sizes	Number of snapshots	
		PFSDisk	AtomicDisk
hm	22 GB	280K	1
mds	8 GB	276K	1
prn	49 GB	788K	1
wdev	8 GB	173K	1
web	13 GB	311K	1

Table 2: AtomicDisk vs. PFSDisk in snapshot numbers.

Rolling back the last journal blocks is equivalent to rolling back the state of the entire disk. This can be prevented with trusted monotonic counters [41, 49, 61], which can be readily integrated into AtomicDisk's secure journal.

Eviction attacks. By design, AtomicDisk can distinguish committed synced writes from uncommitted evicted ones in the journal. Therefore, the adversary cannot effectively conduct eviction attacks even if it is able to capture and replay any snapshots, as the recovery journal guarantees that AtomicDisk can recover to the latest state where all written blocks have been committed.

4.4 Implementation

We have implemented a prototype of AtomicDisk based on Rust [4], which aims to achieve memory safety and easy integration with Occlum. We modified the SGX-PFS's metadata node and journal to achieve sync atomicity. The structure of the MHT remains the same as in SGX-PFS. Subsequently, we develop it as a virtual disk and integrate it with Occlum. The implementation comprises approximately 5,000 Lines of Code (LoC).

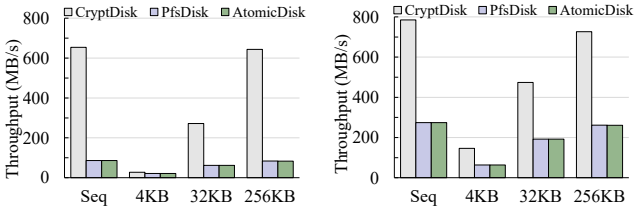
5 Evaluation

5.1 Experiment Setup

Baseline. We compare the performance of AtomicDisk with two baseline secure virtual disks, namely PFSDisk and CRYPTDISK. PFSDisk is a secure block device that simply redirects block I/O to an underlying SGX-PFS protected file. CRYPTDISK is a straightforard secure virtual disk that mimics Linux's dm-crypt and dm-integrity in combination. All AtomicDisk, PFSDisk and CRYPTDISK run on the Occlum [58] library OS for SGX.

Testbed. We set up one machine for tests on SGX. The SGX machine has a 64-core Intel Xeon Processor (Icelake) up to 3.10GHz, an Intel DC S3500 SATA SSD, and 256GB memory (out of which 64GB is preserved as SGX EPC). We use Linux kernel 5.17 and Intel SGX SDK 2.15.

Configuration. We configure PFSDisk, AtomicDisk and CRYPTDISK to have a 100GB user-visible disk capacity. The default block size is 4KB.



(a) Writes in SGX (b) Reads in SGX

Figure 3: The `FIO` benchmark. We test both random and sequential write/read workloads. For random workloads, block size = 4KB, 32KB, and 256KB.

5.2 Security Results

To quantify the extent to which the vulnerable snapshots can be eliminated by AtomicDisk, we compare PFSDisk and AtomicDisk in the numbers of snapshots generated during trace-driven, block-level I/O workloads. In this benchmark, Both PFSDisk and AtomicDisk use the default cache size (192KB). Each of the trace [20] only submits one sync request in the end. As expected, AtomicDisk only generates one valid snapshot (caused by the sync). In contrast, PFSDisk generates hundreds of thousands of snapshots per trace thus being vulnerable to eviction attacks.

5.3 Micro Benchmarks

We use `fio` [10] to generate sequential/random reads/writes in various I/O sizes to all three disks. `fio` is configured as follows: `ioengine=sync`, `direct=1`, `nunjobs=1`, and `fsync_on_close=1`. All three disks benefit from caching. For their optimal performance, we allow both of them to have a cache of 1GB of memory.

The results are shown in Fig. 3. First, AtomicDisk and PFSDisk have similar write performance, as AtomicDisk saves both committed and uncommitted blocks to the journal, thus having the same write amplifications as PFSDisk. Second, AtomicDisk and PFSDisk also have similar read performance, as they process read requests exactly the same. CRYPTDISK consistently outperforms AtomicDisk and PFSDisk, with speedups ranging from 1.2× to 7.5× for write and 2.2× to 2.8× for read, because it only performs data encryption without an MHT structure, avoiding write amplification.

5.4 Trace-Driven Benchmarks

Next, we evaluate the performance of AtomicDisk on more realistic workloads and compare it with PFSDisk and CRYPTDISK. We use block-level I/O traces gathered from data center servers providing different services [44]. For each of the traces, we load records from it and submit I/O requests accordingly to a target secure block device.

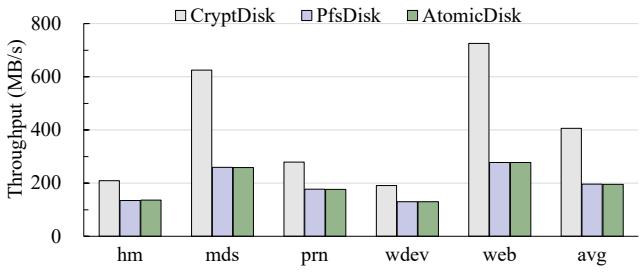
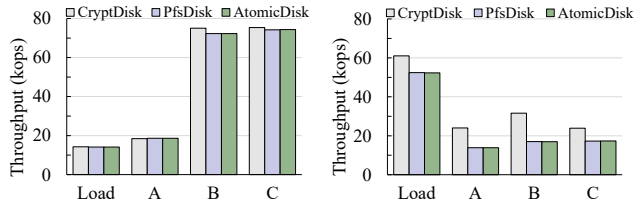


Figure 4: Trace-driven benchmark.



(a) Redis (b) BadgerDB

Figure 5: The YCSB benchmark. Record size: 1KB. # records: 5M. # operations: 1M. Threads: 64. Request dist.: uniform.

As illustrated in Fig. 4, AtomicDisk achieves comparable performance to PFSDisk, and the performance gap between AtomicDisk and CRYPTDISK narrows as these workloads mainly consist of small read/write operations.

5.5 YCSB Benchmarks

We measure the performance of applications running on three disks using YCSB [30]. We select Redis and BadgerDB [5](a persistent key-value store based on LSM-tree) as the target applications. We use four workloads: Load (insert only), YCSB-A (read:update=50:50), YCSB-B (read:update=95:5), YCSB-C (read only). For Redis, `appendonly=yes` and `appendfsync=always` are set.

As shown in Fig. 5, for Redis, all three disks achieve comparable performance due to Redis's lightweight I/O patterns, even with (`appendfsync=always`) set. For BadgerDB, AtomicDisk and PFSDisk achieve comparable performance, reaching 50%-85% of CRYPTDISK's throughput. This gap is due to BadgerDB's sequential writes not saturating disk bandwidth, and its uniform read distribution causing frequent small random reads that bottleneck performance.

6 Related Work

Secure file systems. To secure the file I/O of TEE, prior work from both academia [39, 62] and industry [3, 21, 58] build from scratch secure file systems that are based on in-place updates and MHTs. Despite the great efforts, these new file systems fall short of the features and quality offered by mature file systems. In contrast, AtomicDisk does not require modifications to existing file systems. More important, it covers a broader range of security attributes, guaranteeing sync atomicity thus preventing eviction attacks, while delivering improved performance.

Transactional file systems. Amino [63] uses Berkeley DB [47] as the backing store to support transactional operations. Transactional NTFS [36], Valor [60], TxFS [33] and exF2FS [46] provide transactional interface to file system users. TxOS [51] provides transactions as a first-class abstraction in the OS and converts ext3 to be transactional. They target general FS semantics with high performance penalty.

Transactional block storage. The concept of transactional atomicity for multi-block writes was first introduced in Mime [29]. Over the years, this idea has been developed further by projects such as Logical Disk [31], Stasis [57], and Tx Flash [52], which proposed solutions for achieving atomicity at the block or page level. Isotope [59] provides transaction isolation for block storage. However, this proposed design only targets atomicity or transaction isolation.

Crash consistency. It is known that the file I/O stack writes user data out of program order, generating transient and maybe unexpected on-disk states [28, 50]. Traditionally, these transient states are only visible to users in case of rare events of crashes. Therefore, they are mostly relevant to crash consistency. To the best of our knowledge, this work is the first to recognize that the transient states may be exploited by a TEE adversary to launch eviction attacks.

7 Conclusion and Discussion

The top level contribution of this paper is the discovery of a new type of *eviction attacks*, where a privileged adversary can capture transient vulnerable snapshots (generated by cache eviction) and replay the captured snapshots to conduct malicious behaviors. To address this vulnerability, we propose a new security property called *sync atomicity*, which promises that all (uncommitted) writes before a sync are committed in an all-or-nothing manner. We then present AtomicDisk, a secure virtual disk that achieves sync atomicity by introducing an internal `commit` operation and enhancing the recovery journal of SGX-PFS. Compared to the state-of-the-art SGX-PFS, AtomicDisk can eliminate the eviction-generated vulnerable snapshots while achieving similar I/O performance.

Acknowledgement

We thank our shepherd, Prof. Ming Zhao, and the anonymous reviewers for their valuable suggestions. The work is supported by the National Key Research and Development Program of China (no. 2022YFB4500302) and the National Natural Science Foundation of China (no. 62441220). Yiming Zhang and Shoumeng Yan are the corresponding authors.

References

[1] AMD Secure Encrypted Virtualization (SEV). https://developer.amd.com/sev/. Accessed: 2024-09-10.

[2] Arm Confidential Compute Architecture (Arm CCA). https://developer.arm.com/architectures/architecture-security-features/confidential-computing. Accessed: 2024-09-10.

[3] Asylo project. https://github.com/google/asylo. Accessed: 2024-09-10.

[4] AtomicDisk codebase. https://github.com/nicexlab/AtomicDisk. Accessed: 2024-9-10.

[5] BadgerDB. https://dgraph.io/docs/badger/. Accessed: 2024-9-10.

[6] Cassandra: Open Source NoSQL Database. https://cassandra.apache.org/. Accessed: 2024-09-10.

[7] Device-mapper's "crypt" target. https://www.kernel.org/doc/html/latest/admin-guide/device-mapper/dm-crypt.html. Accessed: 2024-09-10.

[8] Dm-integrity. https://docs.kernel.org/admin-guide/device-mapper/dm-integrity.html. Accessed: 2024-09-10.

[9] Filesystem-level encryption (fscrypt). https://www.kernel.org/doc/html/latest/filesystems/fscrypt.html. Accessed: 2024-09-10.

[10] fio - Flexible I/O tester. https://fio.readthedocs.io/en/latest/fio_doc.html. Accessed: 2024-09-10.

[11] Galois/counter mode. https://en.wikipedia.org/wiki/Galois/Counter_Mode. Accessed: 2024-09-10.

[12] How 3rd Generation Intel® Xeon® Scalable Processor platforms support 1 TB EPCs. https://www.intel.sg/content/www/xa/en/support/articles/000059614/software/intel-security-products.html. Accessed: 2024-09-10.

[13] Intel Software Guard Extensions (SGX). https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/overview.html. Accessed: 2024-09-10.

[14] Intel Trust Domain Extensions (Intel TDX). https://www.intel.com/content/www/us/en/developer/articles/technical/intel-trust-domain-extensions.html. Accessed: 2024-09-10.

[15] Intel® sgx sdk advisory: Intel-sa-00691. https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00691.html.

[16] MongoDB. https://www.mongodb.com/. Accessed: 2024-09-10.

[17] Redis is an in-memory database that persists on disk. https://github.com/redis/redis. Accessed: 2024-09-10.

[18] Reproduction Artifact. https://github.com/nicexlab/EvictionAttack. Accessed: 2024-9-10.

[19] Understanding SGX Protected File System. https://www.tatetian.io/2017/01/15/understanding-sgx-protected-file-system/. Accessed: 2024-09-10.

[20] Msr cambridge traces. http://iotta.snia.org/traces/388, 2020. Accessed: 2024-09-10.

[21] Intel protected file system library. https://www.intel.com/content/dam/develop/external/us/en/documents/overviewofintelprotectedfilesystemlibrary.pdf, 2022. Accessed: 2024-09-10.

[22] Linux early user spacec. https://www.kernel.org/doc/Documentation/early-userspace/README, 2022. Accessed: 2024-09-10.

[23] Eviction attack demonstration. https://sites.google.com/view/secure-virtual-disks/attack-demo, 2024. Accessed: 2024-09-10.

[24] Adil Ahmad, Kyungtae Kim, Muhammad Ihsanulhaq Sarfaraz, and Byoungyoung Lee. OBLIVIATE: A data oblivious filesystem for intel SGX. In *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*. The Internet Society, 2018.

[25] Jinwoo Ahn, Junghee Lee, Yungwoo Ko, Donghyun Min, Jiyun Park, Sungyong Park, and Youngjae Kim. Diskshield: A data tamper-resistant storage for intel sgx. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, ASIA CCS '20, page 799–812, New York, NY, USA, 2020. Association for Computing Machinery.

[26] Sebastian Angel, Aditya Basu, Weidong Cui, Trent Jaeger, Stella Lau, Srinath Setty, and Sudheesh Singanamalla. Nimble: Rollback protection for confidential cloud services. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pages 193–208, Boston, MA, July 2023. USENIX Association.

[27] Maurice Bailleu, Jörg Thalheim, Pramod Bhatotia, Christof Fetzer, Michio Honda, and Kapil Vaswani. SPEICHER: securing lsm-based key-value stores using

shielded execution. In Arif Merchant and Hakim Weatherspoon, editors, *17th USENIX Conference on File and Storage Technologies, FAST 2019, Boston, MA, February 25-28, 2019*, pages 173–190. USENIX Association, 2019.

[28] James Bornholt, Antoine Kaufmann, Jialin Li, Arvind Krishnamurthy, Emina Torlak, and Xi Wang. Specifying and checking file system crash-consistency models. In Tom Conte and Yuanyuan Zhou, editors, *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2016, Atlanta, GA, USA, April 2-6, 2016*, pages 83–98. ACM, 2016.

[29] Chia Chao, Robert English, David Jacobson, Er Stepanov, and John Wilkes. Mime: A high performance parallel storage device with strong recovery guarantees. 07 1995.

[30] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, page 143–154, New York, NY, USA, 2010. Association for Computing Machinery.

[31] Wiebren De Jonge, M Frans Kaashoek, and Wilson C Hsieh. The logical disk: A new approach to improving file systems. In *Proceedings of the fourteenth ACM symposium on Operating systems principles*, pages 15–28, 1993.

[32] Michael Austin Halcrow. eCryptfs: An enterprise-class encrypted filesystem for Linux. In *Proceedings of the 2005 Linux Symposium*, volume 1, pages 201–218, 2005.

[33] Yige Hu, Zhiting Zhu, Ian Neal, Youngjin Kwon, Tianyu Cheng, Vijay Chidambaram, and Emmett Witchel. Txfs: Leveraging file-system crash consistency to provide acid transactions. *ACM Transactions on Storage (TOS)*, 15(2):1–20, 2019.

[34] Guerney D. H. Hunt, Ramachandra Pai, Michael V. Le, Hani Jamjoom, Sukadev Bhattiprolu, Rick Boivie, Laurent Dufour, Brad Frey, Mohit Kapur, Kenneth A. Goldman, Ryan Grimm, Janani Janakirman, John M. Ludden, Paul Mackerras, Cathy May, Elaine R. Palmer, Bharata Bhasker Rao, Lawrence Roy, William A. Starke, Jeff Stuecheli, Enriquillo Valdez, and Wendel Voigt. Confidential Computing for OpenPOWER. In *Proceedings of the Sixteenth European Conference on Computer Systems*, EuroSys '21, page 294–310, New York, NY, USA, 2021. Association for Computing Machinery.

[35] Tyler Hunt, Congzheng Song, Reza Shokri, Vitaly Shmatikov, and Emmett Witchel. Chiron: Privacy-preserving machine learning as a service. *CoRR*, abs/1803.05961, 2018.

[36] John Kennedy and Michael Satran. About transactional ntfs. https://docs.microsoft.com/en-us/windows/desktop/fileio/about-transactional-ntfs. Accessed: 2024-10-11.

[37] Igjae Kim, J Hyun Kim, Minu Chung, Hyungon Moon, and Sam H Noh. A {Log-Structured} merge tree-aware message authentication scheme for persistent {Key-Value} stores. In *20th USENIX Conference on File and Storage Technologies (FAST 22)*, pages 363–380, 2022.

[38] Taehoon Kim, Joongun Park, Jaewook Woo, Seungheun Jeon, and Jaehyuk Huh. Shieldstore: Shielded in-memory key-value storage with sgx. In *Proceedings of the Fourteenth EuroSys Conference 2019*, EuroSys '19, New York, NY, USA, 2019. Association for Computing Machinery.

[39] Sandeep Kumar and Smruti R. Sarangi. Securefs: A secure file system for intel sgx. In *24th International Symposium on Research in Attacks, Intrusions and Defenses*, RAID '21, page 91–102, New York, NY, USA, 2021. Association for Computing Machinery.

[40] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Krste Asanović, and Dawn Song. Keystone: An open framework for architecting trusted execution environments. In *Proceedings of the Fifteenth European Conference on Computer Systems*, EuroSys '20, New York, NY, USA, 2020. Association for Computing Machinery.

[41] Sinisa Matetic, Mansoor Ahmed, Kari Kostiainen, Aritra Dhar, David M. Sommer, Arthur Gervais, Ari Juels, and Srdjan Capkun. ROTE: rollback protection for trusted execution. In Engin Kirda and Thomas Ristenpart, editors, *26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017*, pages 1289–1306. USENIX Association, 2017.

[42] Ralph C. Merkle. A digital signature based on a conventional encryption function. In Carl Pomerance, editor, *Advances in Cryptology — CRYPTO '87*, pages 369–378, Berlin, Heidelberg, 1988. Springer Berlin Heidelberg.

[43] Mitar Milutinovic, Warren He, Howard Wu, and Maxinder Kanwal. Proof of luck: an efficient blockchain consensus protocol. In *Proceedings of the 1st Workshop on System Software for Trusted Execution, SysTEX@Middleware 2016, Trento, Italy, December 12, 2016*, pages 2:1–2:6. ACM, 2016.

[44] Dushyanth Narayanan, Austin Donnelly, and Antony I. T. Rowstron. Write off-loading: Practical power management for enterprise storage. In Mary Baker and Erik Riedel, editors, *6th USENIX Conference on File and Storage Technologies, FAST 2008, February 26-29, 2008, San Jose, CA, USA*, pages 253–267. USENIX, 2008.

[45] Jianyu Niu, Wei Peng, Xiaokuan Zhang, and Yinqian Zhang. Narrator: Secure and practical state continuity for trusted execution in the cloud. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 2385–2399, New York, NY, USA, 2022. Association for Computing Machinery.

[46] Joontaek Oh, Sion Ji, Yongjin Kim, and Youjip Won. exF2FS: Transaction support in Log-Structured filesystem. In *20th USENIX Conference on File and Storage Technologies (FAST 22)*, pages 345–362, Santa Clara, CA, February 2022. USENIX Association.

[47] Michael A Olson, Keith Bostic, and Margo I Seltzer. Berkeley db. In *USENIX Annual Technical Conference, FREENIX Track*, pages 183–191, 1999.

[48] Meni Orenbach, Pavel Lifshits, Marina Minkin, and Mark Silberstein. Eleos: Exitless os services for sgx enclaves. In *Proceedings of the Twelfth European Conference on Computer Systems*, EuroSys '17, page 238–253, New York, NY, USA, 2017. Association for Computing Machinery.

[49] Bryan Parno, Jacob R. Lorch, John R. Douceur, James W. Mickens, and Jonathan M. McCune. Memoir: Practical state continuity for protected modules. In *32nd IEEE Symposium on Security and Privacy, S&P 2011, 22-25 May 2011, Berkeley, California, USA*, pages 379–394. IEEE Computer Society, 2011.

[50] Thanumalayan Sankaranarayana Pillai, Vijay Chidambaram, Ramnatthan Alagappan, Samer Al-Kiswany, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. All file systems are not created equal: On the complexity of crafting crash-consistent applications. In Jason Flinn and Hank Levy, editors, *11th USENIX Symposium on Operating Systems Design and Implementation, OSDI '14, Broomfield, CO, USA, October 6-8, 2014*, pages 433–448. USENIX Association, 2014.

[51] Donald E Porter, Owen S Hofmann, Christopher J Rossbach, Alexander Benn, and Emmett Witchel. Operating system transactions. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 161–176, 2009.

[52] Vijayan Prabhakaran, Thomas L. Rodeheffer, and Lidong Zhou. Transactional flash. In *8th USENIX Symposium on Operating Systems Design and Implementation (OSDI 08)*, San Diego, CA, December 2008. USENIX Association.

[53] Christian Priebe, Divya Muthukumaran, Joshua Lind, Huanzhou Zhu, Shujie Cui, Vasily A. Sartakov, and Peter Pietzuch. SGX-LKL: Securing the host os interface for trusted execution, 2020.

[54] Christian Priebe, Kapil Vaswani, and Manuel Costa. Enclavedb: A secure database using SGX. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, pages 264–278. IEEE Computer Society, 2018.

[55] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. Trusted execution environment: what it is, and what it is not. In *2015 IEEE Trustcom/BigDataSE/ISPA*, volume 1, pages 57–64. IEEE, 2015.

[56] Felix Schuster, Manuel Costa, Cédric Fournet, Christos Gkantsidis, Marcus Peinado, Gloria Mainar-Ruiz, and Mark Russinovich. VC3: trustworthy data analytics in the cloud using SGX. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 38–54. IEEE Computer Society, 2015.

[57] Russell Sears and Eric Brewer. Stasis: Flexible transactional storage. In *Proceedings of the 7th symposium on Operating systems design and implementation*, pages 29–44, 2006.

[58] Youren Shen, Hongliang Tian, Yu Chen, Kang Chen, Runji Wang, Yi Xu, Yubin Xia, and Shoumeng Yan. Occlum: Secure and efficient multitasking inside a single enclave of intel SGX. In James R. Larus, Luis Ceze, and Karin Strauss, editors, *ASPLOS '20: Architectural Support for Programming Languages and Operating Systems, Lausanne, Switzerland, March 16-20, 2020*, pages 955–970. ACM, 2020.

[59] Ji-Yong Shin, Mahesh Balakrishnan, Tudor Marian, and Hakim Weatherspoon. Isotope: Transactional isolation for block storage. In *14th USENIX Conference on File and Storage Technologies (FAST 16)*, pages 23–37, Santa Clara, CA, February 2016. USENIX Association.

[60] Richard P Spillane, Sachin Gaikwad, Manjunath Chinni, Erez Zadok, and Charles P Wright. Enabling transactional file access via lightweight kernel extensions. In *FAST*, volume 9, pages 29–42, 2009.

[61] Raoul Strackx and Frank Piessens. Ariadne: A minimal approach to state continuity. In Thorsten Holz and Stefan Savage, editors, *25th USENIX Security Symposium,*

USENIX Security 16, Austin, TX, USA, August 10-12, 2016, pages 875–892. USENIX Association, 2016.

[62] Chia-che Tsai, Donald E. Porter, and Mona Vij. Graphene-sgx: A practical library OS for unmodified applications on SGX. In Dilma Da Silva and Bryan Ford, editors, *2017 USENIX Annual Technical Conference, USENIX ATC 2017, Santa Clara, CA, USA, July 12-14, 2017*, pages 645–658. USENIX Association, 2017.

[63] Charles P Wright, Richard Spillane, Gopalan Sivathanu, and Erez Zadok. Extending acid semantics to the file system. *ACM Transactions on Storage (TOS)*, 3(2):4–es, 2007.

[64] Lujia Yin, Li Wang, Yiming Zhang, and Yuxing Peng. Mapperx: Adaptive metadata maintenance for fast crash recovery of dm-cache based hybrid storage devices. In Irina Calciu and Geoff Kuenning, editors, *Proceedings of the 2021 USENIX Annual Technical Conference, USENIX ATC 2021, July 14-16, 2021*, pages 705–713. USENIX Association, 2021.

[65] Wenchao Zhou, Yifan Cai, Yanqing Peng, Sheng Wang, Ke Ma, and Feifei Li. Veridb: An sgx-based verifiable database. In *Proceedings of the 2021 International Conference on Management of Data*, SIGMOD/PODS '21, page 2182–2194, New York, NY, USA, 2021. Association for Computing Machinery.