IceClave: A Trusted Execution Environment for In-Storage Computing*

Luyi Kang†‡, Yuqi Xue†, Weiwei Jia†, Xiaohao Wang, Jongryool Kim§, Changhwan Youn§, Myeong Joon Kang§, Hyung Jin Lim§, Bruce Jacob‡, Jian Huang
UIUC, ‡University of Maryland, College Park, §SK Hynix

Abstract

In-storage computing with modern solid-state drives (SSDs) enables developers to offload programs from the host to the SSD. It has been proven to be an effective approach to alleviating the I/O bottleneck. To facilitate in-storage computing, many frameworks have been proposed. However, few of them treat the in-storage security as the first citizen. Specifically, since modern SSD controllers do not have a trusted execution environment, an offloaded (malicious) program could steal, modify, and even destroy the data stored in the SSD.

In this paper, we first investigate the attacks that could be conducted by offloaded in-storage programs. To defend against these attacks, we build a lightweight trusted execution environment, named IceClave for in-storage computing. IceClave enables security isolation between in-storage programs and flash management functions. IceClave also achieves security isolation between in-storage programs and enforces memory encryption and integrity verification of in-storage DRAM with low overhead. To protect data loaded from flash chips, IceClave develops a lightweight data encryption/decryption mechanism in flash controllers. We develop IceClave with a full system simulator. We evaluate IceClave with a variety of data-intensive applications such as databases. Compared to state-of-the-art in-storage computing approaches, IceClave introduces only 7.6% performance overhead, while enforcing security isolation in the SSD controller with minimal hardware cost. IceClave still keeps the performance benefit of in-storage computing by delivering up to 2.31× better performance than the conventional host-based trusted computing approach.

1 Background and Motivation

In-storage computing has been a promising technique for accelerating data-intensive applications, especially for large-scale data processing and analytics. It moves computation closer to the data stored in the storage devices like flash-based solid-state drives (SSDs), such that it can overcome the I/O bottleneck by reducing the amount of data transferred between the host machine and storage devices. As modern SSDs are employing multiple general-purpose embedded processors and large DRAM in their controllers, it becomes feasible to enable in-storage computing in reality today.

To facilitate the wide adoption of in-storage computing, a variety of frameworks have been proposed. All these prior works show the great potential of in-storage computing for accelerating data processing in data centers. However, most of them focus on the performance and programmability, but few of them treat the security as the first citizen in their design and implementation, which imposes great threat to the user data and SSD devices, and further hinders its widespread adoption.

As in-storage processors operate independently from the host machine, and modern SSD controllers do not provide a trusted execution environment (TEE) for programs running inside the SSDs, they pose severe security threats to user data and flash chips. To be specific, a piece of offloaded (malicious) code could (1) manipulate the mapping table in the flash translation layer (FTL) to mangle the data management of flash chips, (2) access and destroy data belonging to other applications, and (3) steal and modify the memory of co-located in-storage programs at runtime. Even worse, adversaries can steal and modify intermediate data and results generated by in-storage programs via physical attacks such as cold-boot attack, bus snooping attack, and replay attack.

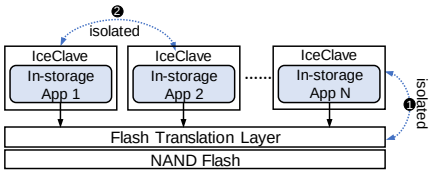

Figure 1: IceClave enables in-storage TEEs to achieve security isolation between in-storage programs, FTL, and flash chips.

To overcome these security challenges, state-of-the-art in-storage computing frameworks maintain a copy of the privilege information in the SSD DRAM and enforcing permission checks for in-storage programs. However, such a solution still suffers from many security vulnerabilities. An alternative approach is to adopt Intel SGX. Unfortunately, modern in-storage processors do not support SGX, and it also incurs significant performance overhead [1].

Therefore, providing a secure, lightweight, and trusted execution environment for in-storage computing is an essential step towards its widespread adoption. Ideally, we wish to enjoy the performance benefits of in-storage computing, while enforcing the security isolation between in-storage programs, the core FTL functions, and physical flash chips, as demonstrated in Figure 1.

2 Threat Model

We target the multitenancy where multiple application instances operate in the shared SSD. Following the threat models for cloud computing today, we assume the cloud computing platform has provided a secure channel for end users to offload their programs to the shared SSD. The related code-offloading techniques, such as secure RPC and libraries, have already been deployed in cloud platforms. However, a program potentially offloaded by a malicious user can still include (hidden) malicious code.

We assume hardware vendors do not intentionally implant backdoor or malicious programs in their devices. However, as we deploy those computational SSDs in shared platforms (e.g., public cloud), we do not trust the platform operators who could initiate board-level physical attacks such as bus-snooping and man-in-the-middle attacks, or exploit the host machine to steal or destroy data stored in SSDs. Similar to the threat model for SGX, we assume that the processor chip is safe against physical attacks, and we exclude software side-channel attacks [1].

3 Design and Implementation

In this paper, we present IceClave, a trusted execution environment for in-storage computing. IceClave is designed specifically for modern SSD controllers and in-storage programs, with considering the unique flash properties and in-storage workload characteristics. With ensuring the security isolation, IceClave includes (1) a new

*This work has been published at MICRO'21 [2] and is publicly available at https://arxiv.org/pdf/2109.03373.pdf.
†Co-primary authors.

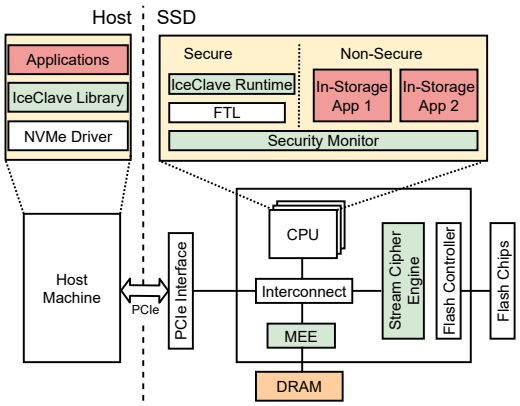

Figure 2: Overview of IceClave architecture.

memory protection scheme to protect the FTL and reduce the context switch overhead incurred by flash address translations; (2) a technique for securing in-storage DRAM for in-storage programs by taking advantage of the fact that most in-storage applications are read intensive; (3) a stream cipher engine for securing data transfers between storage processors and flash chips, with low performance overhead and energy consumption; and (4) a runtime system for managing the life cycle of in-storage TEEs. IceClave aims to defend against three attacks: (1) the attack against the FTL; (2) the attack against co-located in-storage programs; and (3) the potential physical attack against the data loaded from flash chips and intermediate data written in the in-storage DRAM.

Protecting Flash Translation Layer. As FTL manages flash blocks and controls how user data is mapped to each flash page, its protection is crucial. If any malicious in-storage programs gain control over it, they can read, erase, or overwrite data from other users, causing severe consequences such as data loss and leakage.

To protect FTL from malicious in-storage programs, we have to guarantee offloaded applications cannot access memory regions used by FTL. We can use ARM TrustZone to create secure and normal worlds, and then place FTL functions in the secure world, and place all in-storage applications in the normal world. However, this will cause significant performance overhead for in-storage applications. This is because when an application accesses a flash page each time, it needs to context switch to the secure world which hosts the FTL and its address mapping table.

To address this challenge, we partition the entire physical main memory space into three memory regions: normal, protected, and secure by extending TrustZone. We allow FTL to execute in the secure world, and place in-storage applications in the normal world; therefore, they cannot access any code or data regions that belong to the FTL. We use the protected memory region in the normal world to host the shared address mapping table, such that in-storage applications can only read the mapping table entries for address translation, without paying the context-switch overhead.

Access Control for In-Storage Programs. Although each in-storage program only has the read access permission when accessing the mapping table of the FTL, a malicious in-storage program could probe the mapping table entries (e.g., by brute-force) and easily access the data belonging to other in-storage programs.

To address this challenge, we extend the address mapping table of FTL. We use the ID bits in each entry (8 bytes per entry) to track the identification of each in-storage TEE, and use them to verify whether an in-storage TEE has the permission to access the mapping table entry or not. Each in-storage program only has accesses to the address mapping table of the FTL and allocated memory space. Accesses to other memory locations will result in a fault in the memory management unit.

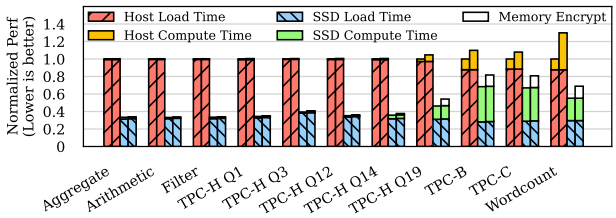

Figure 3: Performance comparison of Host, Host+SGX, ISC, and IceClave (from left to right).

Securing In-Storage DRAM. In-storage programs load data from flash chips to the SSD DRAM for data processing. The user data that includes raw data, intermediate data, and produced results in the DRAM could be leaked or tampered with at runtime due to physical attacks. To address this challenge, IceClave enables both memory encryption and integrity verification.

For memory encryption, the state-of-the-art work usually uses split-counter encryption [3], which has significant performance overhead. However, this is less of a concern for in-storage computing because in-storage workloads are mostly read intensive. Based on this observation, we design the hybrid-counter scheme.

The key idea of hybrid-counter is that we only use major counters for read-only pages. For writable pages, we apply the traditional split-counter scheme. As minor counters will not change as long as the pages are read-only, we do not need minor counters for read-only pages. In this case, we can improve the counter fetching performance by packing more counters per cache line.

To ensure the processor receives exactly the same content as it wrote in the memory most recently, we also enable memory integrity verification by employing Bonsai Merkle Tree (BMT) [3]. Due to the hybrid-counter scheme, IceClave maintains two Merkle trees, but the extra memory cost is negligible.

IceClave Implementation. We show the overview of IceClave architecture in Figure 2. We extend ARM TrustZone to create secure and normal world for security isolation and protection of different entities in FTL, while enabling memory encryption and verification with memory encryption engine (MEE). We implement IceClave with a computational SSD simulator developed based on the SimpleSSD, Gem5, and USIMM simulator. To verify the core functions of IceClave, we also implement a real-system prototype with an OpenSSD Cosmos+ FPGA board.

Performance of IceClave. We evaluate IceClave with a set of synthetic and real-world workloads that are typical for in-storage computing. We compare IceClave with the following state-of-the-art solutions: (1) host-based computing without security (`Host`), (2) host with Intel SGX (`Host+SGX`), and (3) in-storage computing without security (`ISC`). As shown in Figure 3, IceClave outperforms Host and Host+SGX by more than 2.3×, respectively. Compared to the ISC baseline, IceClave introduces 7.6% performance overhead, due to the security techniques used in the in-storage TEE. We presented more sensitivity analyses in the paper [2].

References

[1] Victor Costan and Srinivas Devadas. [n. d.]. Intel SGX Explained. https://eprint.iacr.org/2016/086.pdf.

[2] Luyi Kang, Yuqi Xue, Weiwei Jia, Xiaohao Wang, Jongryool Kim, Changhwan Youn, Myeong Joon Kang, Hyung Jin Lim, Bruce Jacob, and Jian Huang. 2021. IceClave: A Trusted Execution Environment for In-Storage Computing. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '21)*. Association for Computing Machinery, New York, NY, USA, 199–211. https://doi.org/10.1145/3466752.3480109

[3] Brian Rogers, Siddhartha Chhabra, Milos Prvulovic, and Yan Solihin. 2007. Using address independent seed encryption and bonsai merkle trees to make secure processors os-and performance-friendly. In *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*. IEEE, 183–196.