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

*Co-primary authors.
Host-based Computing
In-Storage Computing: A Promising Technique for I/O-Intensive Applications

Host-based Computing

Host

CPU

DRAM

SSD

I/O Bottleneck!
In-Storage Computing: A Promising Technique for I/O-Intensive Applications

Host-based Computing

- Host
  - CPU
  - DRAM
  - SSD

- Host-bottleneck!

In-Storage Computing

- Host
  - In-Storage Processor
  - DRAM Buffer
  - Flash Memory

Comparison between Host-based and In-Storage Computing.
In-Storage Computing: A Promising Technique for I/O-Intensive Applications

Host-based Computing

Host

CPU

DRAM

SSD

I/O Bottleneck!

In-Storage Computing

Host

In-Storage Processor

DRAM Buffer

Flash Memory
In-storage computing offers an effective solution to alleviate the I/O bottleneck.
SSD Architecture for In-Storage Computing

SSD Controller
SSD Architecture for In-Storage Computing

- SSD Controller
- PCIe Interface
- Host
- Embedded Cores
- Internal Bus
- Memory Controller
- Off-chip DRAM
SSD Architecture for In-Storage Computing

- SSD Controller
- Embedded Cores
- Internal Bus
- Memory Controller
- Off-chip DRAM
- PCIe Interface
- Host
- Flash Controller
SSD Architecture for In-Storage Computing
SSD Architecture for In-Storage Computing

Host

PCIe Bus

SSD

Block I/O

Flash Translation Layer

NAND Flash
SSD Architecture for In-Storage Computing

- Host
- PCIe Bus
- SSD
- Flash Translation Layer
  - Address Translation
  - NAND Flash
- Block I/O
SSD Architecture for In-Storage Computing

Host

PCle Bus

SSD

Block I/O

Flash Translation Layer

Address Translation

Garbage Collection

NAND Flash
SSD Architecture for In-Storage Computing

- Host
- PCIe Bus
- SSD
- Flash Translation Layer
  - Address Translation
  - Garbage Collection
  - Wear Leveling
- NAND Flash
- Block I/O
State-of-the-Art Frameworks for In-Storage Computing

- Flow-based Programming
  - SSDlet → SSDlet → ...
  - SSD Processor

- MapReduce-based Framework
State-of-the-Art Frameworks for In-Storage Computing

Flow-based Programming
SSDlet → SSDlet → ...

MapReduce-based Framework

RPC-based Offloading

Host → Lightweight OS

User App → SSD Processor

User App → SSD Processor

User App → SSD Processor
State-of-the-Art Frameworks for In-Storage Computing

Flow-based Programming

SSDlet -> SSDlet -> ...

SSD Processor

MapReduce-based Framework

RPC-based Offloading

Host -> Lightweight OS

User App

User App

User App

SSD Processor

RPC

FPGA Accelerator

SSD Controller

Flash Chips

Industry SmartSSD

RPC-based Offloading Industry SmartSSD

SSDlet

Processor

Flash

Chips
Most of the existing frameworks focus on performance and programmability.
State-of-the-Art Frameworks for In-Storage Computing

Most of the existing frameworks focus on performance and programmability.

Few of them consider security as the first-class citizen.
Why Should We Secure In-Storage Computing?

- In-Storage App 1
- In-Storage App 2
- In-Storage App 3
Why Should We Secure In-Storage Computing?

In-Storage App 1 → Flash Translation Layer → NAND Flash

In-Storage App 2

In-Storage App 3
Why Should We Secure In-Storage Computing?

- Attack co-located programs

```
In-Storage App 1  In-Storage App 2  In-Storage App 3

Flash Translation Layer

NAND Flash
```
Why Should We Secure In-Storage Computing?

Attacks on co-located programs can lead to data breaches. The figure shows a data breach affecting In-Storage App 1, which in turn affects the Flash Translation Layer. This layer then impacts NAND Flash.

- In-Storage App 1
- Data Breach
- In-Storage App 3
- Flash Translation Layer
- NAND Flash

Flash Translation Layer

NAND Flash
Why Should We Secure In-Storage Computing?

- Attack co-located programs
- Attack FTL

Flash Translation Layer

NAND Flash

In-Storage App 1

Data Breach

In-Storage App 3
Why Should We Secure In-Storage Computing?

- Attack co-located programs
- Attack FTL

Diagram:

- In-Storage App 1
- Data Breach
- In-Storage App 3
- Data Leakage & Data Loss
- NAND Flash
Why Should We Secure In-Storage Computing?

- Attack co-located programs
- Attack FTL
- Physical Attacks

Diagram:
- In-Storage App 1
- In-Storage App 3
- Data Leakage & Data Loss
- NAND Flash

- Data Breach
Why Should We Secure In-Storage Computing?

- Attack co-located programs
- Attack FTL
- Physical Attacks

Diagram:

- In-Storage App 1
- Data Breach
- In-Storage App 3
- Data Leakage & Data Loss

ELECTRICAL & COMPUTER ENGINEERING
GRAINGER ENGINEERING
Why Should We Secure In-Storage Computing?

It is desirable to build a secure in-storage computing environment!
Existing TEEs Do Not Work For In-Storage Computing

Intel SGX is not available in storage processors
Existing TEEs Do Not Work For In-Storage Computing

- Intel SGX is not available in storage processors
- Unclear how to apply ARM TrustZone to in-storage computing
IceClave: A Trusted Execution Environment for In-Storage Computing

- In-Storage App 1
- In-Storage App 2
- In-Storage App 3

Flash Translation Layer

SSD Processors

Flash Chips

Off-chip DRAM
IceClave: A Trusted Execution Environment for In-Storage Computing

Protecting FTL from malicious in-storage apps

In-Storage App 1  In-Storage App 2  In-Storage App 3

Flash Translation Layer

SSD Processors

Flash Chips

Off-chip DRAM
IceClave: A Trusted Execution Environment for In-Storage Computing

Protecting FTL from malicious in-storage apps

Security isolation between in-storage apps
IceClave: A Trusted Execution Environment for In-Storage Computing

- Protecting FTL from malicious in-storage apps
- Security isolation between in-storage apps
- Securing data against physical attacks

Diagram:
- In-Storage App 1, In-Storage App 2, In-Storage App 3
- Flash Translation Layer
- SSD Processors
- Flash Chips
- Memory Encryption
- Encrypted Flash I/O
- Off-chip DRAM
IceClave Design Challenges

Bare-metal Environment
IceClave Design Challenges

Bare-metal Environment

Efficient Flash Access
IceClave Design Challenges

- Bare-metal Environment
- Efficient Flash Access
- Limited Resources in SSD Device
Threat Model

- In-Storage App 1
- In-Storage App 2
- In-Storage App 3

Flash Translation Layer

- Host
- SSD Processors
- Flash Chips

Off-chip DRAM
Threat Model

- In-storage applications can be malicious

Diagram:
- Host
- SSD Processors
- Flash Chips
- Off-chip DRAM
- Flash Translation Layer
- In-Storage App 1
- In-Storage App 2

Threat Model 9
Threat Model

In-storage applications can be malicious

Cloud platform operator may conduct physical attacks
Host runtime and host-SSD offloading channel is trusted

Cloud platform operator may conduct physical attacks

In-storage applications can be malicious
Protecting Flash Translation Layer

- Protecting FTL from malicious in-storage apps
- Security isolation between in-storage apps
- Securing data against physical attacks
Protecting Flash Translation Layer

Protecting FTL from malicious in-storage apps
Protecting Flash Translation Layer
Protecting Flash Translation Layer

Flash Translation Layer

Secure

In-Storage App 1

In-Storage App 2

Normal
Protecting Flash Translation Layer

Flash Translation Layer

Address Mapping Table

Secure

Normal

In-Storage App 1

In-Storage App 2
Protecting Flash Translation Layer

Naively applying TrustZone partitioning incurs significant performance penalty!
Protecting Flash Translation Layer

Naively applying TrustZone partitioning incurs significant performance penalty!
Naively applying TrustZone partitioning incurs significant performance penalty!
Protecting Flash Translation Layer

- **Read-Only**
- **R/W**

- **Secure**
- **Normal**

- **Flash Translation Layer**
  - **In-Storage App 1**
  - **In-Storage App 2**

- **Address Mapping Table**
  - **Secure**
  - **Protected**
  - **Normal**
Protecting Flash Translation Layer

Flash Translation Layer

In-Storage App 1

In-Storage App 2

Address Mapping Table

Read-Only  →  Secure
Read-Write  →  Normal

Protected

Secure

Normal
Isolating In-Storage Applications

- Protecting FTL from malicious in-storage apps
- Security isolation between in-storage apps
- Securing data against physical attacks
Isolating In-Storage Applications

Security isolation between in-storage apps
Isolating In-Storage Applications

- **Secure**
- **Normal**

- **Read-Only**
- **R/W**

- **Flash Translation Layer**

- **In-Storage App 1**

- **In-Storage App 2**

- **Address Mapping Table**

- **Secure**
- **Protected**
- **Normal**
Isolating In-Storage Applications

- **Secure**
- **Normal**

- **Flash Translation Layer**
- **In-Storage App 1**
- **In-Storage App 2**
- **Address Mapping Table**
- **App 1 Allocated Memory**
- **App 2 Allocated Memory**

- **Read-Only**
- **R/W**
Isolating In-Storage Applications

- IceClave Runtime
- Flash Translation Layer
- In-Storage App 1
- In-Storage App 2
- App 1 Metadata
- App 2 Metadata
- Address Mapping Table
- App 1 Allocated Memory
- App 2 Allocated Memory

Read-Only ➔
R/W ➔

Secure
Normal

Secure
Protected
Normal
Isolating In-Storage Applications

- IceClave Runtime
- Flash Translation Layer
- In-Storage App 1
- In-Storage App 2
- App 1 Metadata
- App 2 Metadata
- Address Mapping Table
- Flash Access Control
- App 1 Allocated Memory
- App 2 Allocated Memory

Read-Only ➔
R/W ➔

Secure

Normal

Protected

Normal
Protecting Against Physical Attacks

Protecting FTL from malicious in-storage apps

Security isolation between in-storage apps

Securing data against physical attacks
Securing data against physical attacks
Protecting Against Physical Attacks

Securing data against physical attacks
Protecting Against Physical Attacks

Securing data against physical attacks

Physical Attacks

SSD Processors

Off-chip DRAM

Flash Chips

Physical Attacks

ELECTRICAL & COMPUTER ENGINEERING

GRAINGER ENGINEERING
Protecting Against Physical Attacks

Securing data against physical attacks

- SSD Processors
- Flash Chips
- Off-chip DRAM

Protecting Physical Memory
Protecting Physical Memory

- Secure Root
- Processor
- Merkle Tree
- Last-Level Counter Blocks
- Data Blocks
Protecting Physical Memory

- Processor
- Secure Root
- Merkle Tree
- Last-Level Counter Blocks
- Data Blocks
Protecting Physical Memory

Split Counter Mode (ISCA’06)
Protecting Physical Memory

Split Counter Mode (ISCA’06)
In-storage programs are read-intensive
In-storage programs are read-intensive
In-storage programs are read-intensive

State-of-the-art Split Counter Mode is not optimal for in-storage computing
Protecting Physical Memory

IceClave Hybrid Counter

Processor

Read-Only Root

Split Counter Root

Read-Only Memory

Writable Memory
Protecting Against Physical Attacks

Securing data against physical attacks

SSD Processors

Flash Chips

Off-chip DRAM

Protecting Physical Memory

Physical Attacks
Protecting Against Physical Attacks

- Securing data against physical attacks
- SSD Processors
- Flash Chips
- Protecting Physical Memory
- Protecting Data
- Access To Flash Chips
- Off-chip DRAM
Protecting Data Access To Flash Chips
Put It All Together

Host Machine

SSD Controller

Host SSD

SSD Controller

CPU Cores

Internal Bus

Memory Encryption Engine

Off-chip DRAM

PCIe Interface

Stream Cipher Engine

Flash Controller

Flash Chips
Put It All Together

Secure World
- IceClave Runtime
- Flash Translation Layer

Normal World
- In-Storage App 1
- In-Storage App 2

Secure Monitor

Host Machine → SSD Controller
- PCIe Interface
- CPU Cores
- Internal Bus
- Memory Encryption Engine
- Off-chip DRAM
- Stream Cipher Engine
- Flash Controller
- Flash Chips

Host SSD
Put It All Together

- Applications
  - IceClave Library
  - NVMe Driver

- Secure World
  - IceClave Runtime
  - Flash Translation Layer

- Normal World
  - In-Storage App 1
  - In-Storage App 2

- Secure Monitor

Host Machine

- Host SSD
  - SSD Controller
    - CPU Cores
    - Stream Cipher Engine
    - Flash Controller
    - Flash Chips
  - PCIe Interface
    - Internal Bus
    - Memory Encryption Engine
    - Off-chip DRAM
IceClave Workflow

IceClave Library

Secure
- IceClave Runtime
- Flash Translation Layer

Protected
- Mapping Table

TEE

Stream Cipher Engine
- Flash Controller
- Flash

Secure
- IceClave Runtime
- Flash Translation Layer
IceClave Workflow

1. Create TEE
2. Secure
   - IceClave Runtime
   - Flash Translation Layer

Protected
- Mapping Table

TEE
- Stream Cipher Engine
- Flash Controller
- Flash
IceClave Workflow

- IceClave Library
- Secure
- IceClave Runtime
- Flash Translation Layer
- Reading Mapping Table
- Protected
- Mapping Table
- TEE
- Stream Cipher Engine
- Flash Controller
- Flash
IceClave Workflow

- IceClave Library
  - Secure
  - IceClave Runtime
  - Flash Translation Layer
  - TEE
  - Stream Cipher Engine
  - Flash Controller

- Protected
  - Mapping Table
  - Flash
IceClave Workflow

IceClave Library

Secure

IceClave Runtime

Flash Translation Layer

Protected

Mapping Table

Flash Mgmt

TEE

Stream Cipher Engine

Flash Controller

Flash
IceClave Workflow

IceClave Library

Secure

IceClave Runtime

Flash Translation Layer

Protected

Mapping Table

TEE

6 Load Data

Stream Cipher Engine

Flash Controller

Flash
IceClave Workflow

IceClave Library

Secure

Flash Translation Layer

Get Result

Protected

Mapping Table

IceClave Runtime

Stream Cipher Engine

Flash Controller

TEE

Flash
IceClave Workflow

- IceClave Library
- Secure
- Flash Translation Layer
- TEE
- Terminating TEE
- IceClave Runtime
- Protected
- Flash Translation Layer
- Mapping Table
- Stream Cipher Engine
- Flash Controller
- Flash
IceClave Workflow

1. Secure Offload App
2. Create TEE
3. Reading Mapping Table
4. Missing Mapping Entry
5. Flash Mgmt
6. Load Data
7. Get Result
8. Terminate TEE

IceClave Library
Flash Translation Layer
IceClave Runtime
Stream Cipher Engine
Flash Controller
Flash
Mapping Table
 Protected
IceClave Implementation

Experimental Setup

**Simulator**
- gem5 + USIMM + SimpleSSD

**Prototype**
- OpenSSD Cosmos+ FPGA

**Synthetic Workloads**
- Arithmetic, Aggregate, Filter, Wordcount

**Real-world Workloads**
- TPC-H, TPC-B, TPC-C
IceClave Overall Performance

- Host Load Time
- SSD Load Time
- Host Compute Time
- SSD Compute Time
- Mempry Encrypt

Normalized Execution Time

Left to Right: Host Host+SGX ISC IceClave
IceClave Overall Performance

IceClave introduces minimal overhead while providing strong security.
IceClave Overall Performance

More evaluations in the paper!
IceClave Summary

First Trusted Execution Environment for In-Storage Computing

2.3× Faster Than Host-based Computing
Thank you!

Luyi Kang, **Yuqi Xue**†, Weiwei Jia, Xiaohao Wang, Jongryool Kim, Changhwan Youn, Myeong Joon Kang, Hyung Jin Lim, Bruce Jacob, Jian Huang

† yuqixue2@illinois.edu

Systems Platform Research Group

This presentation and recording belong to the authors. No distribution is allowed without the authors’ permission.