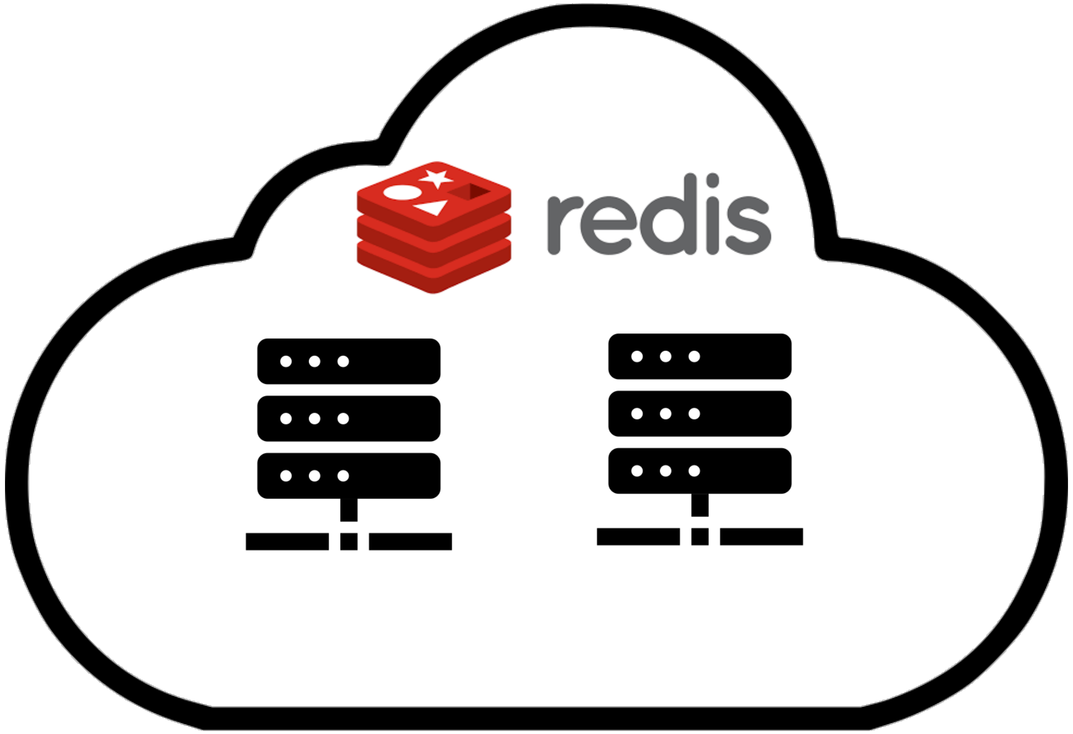
KVSEV:
A Secure In-Memory Key-Value Store with Secure Encrypted Virtualization

Junseung You, Kyeongryong Lee, Hyungon Moon,
Yeongpil Cho, Yunheung Paek

Santa Cruz, USA
October 31-November 1, 2023

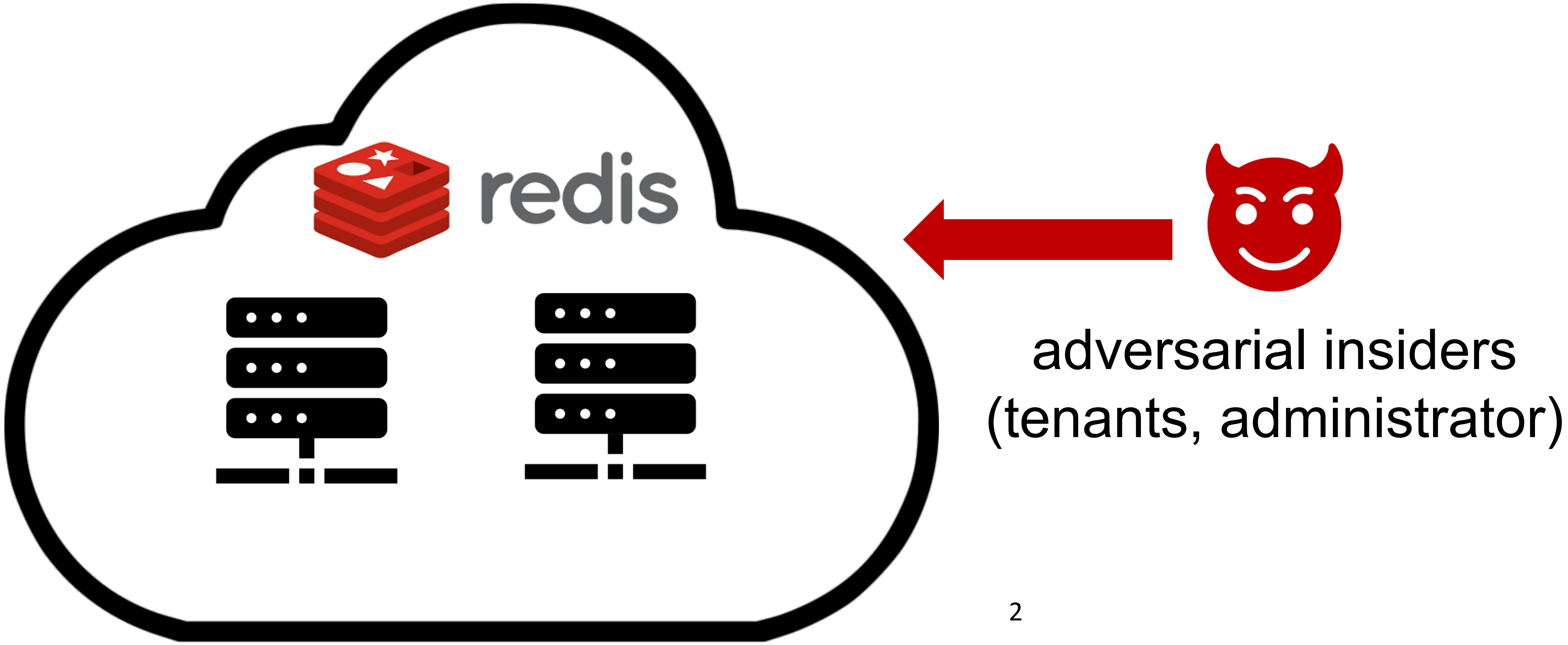
Trusted Key-Value Stores

- User data is exposed to adversarial insiders in cloud



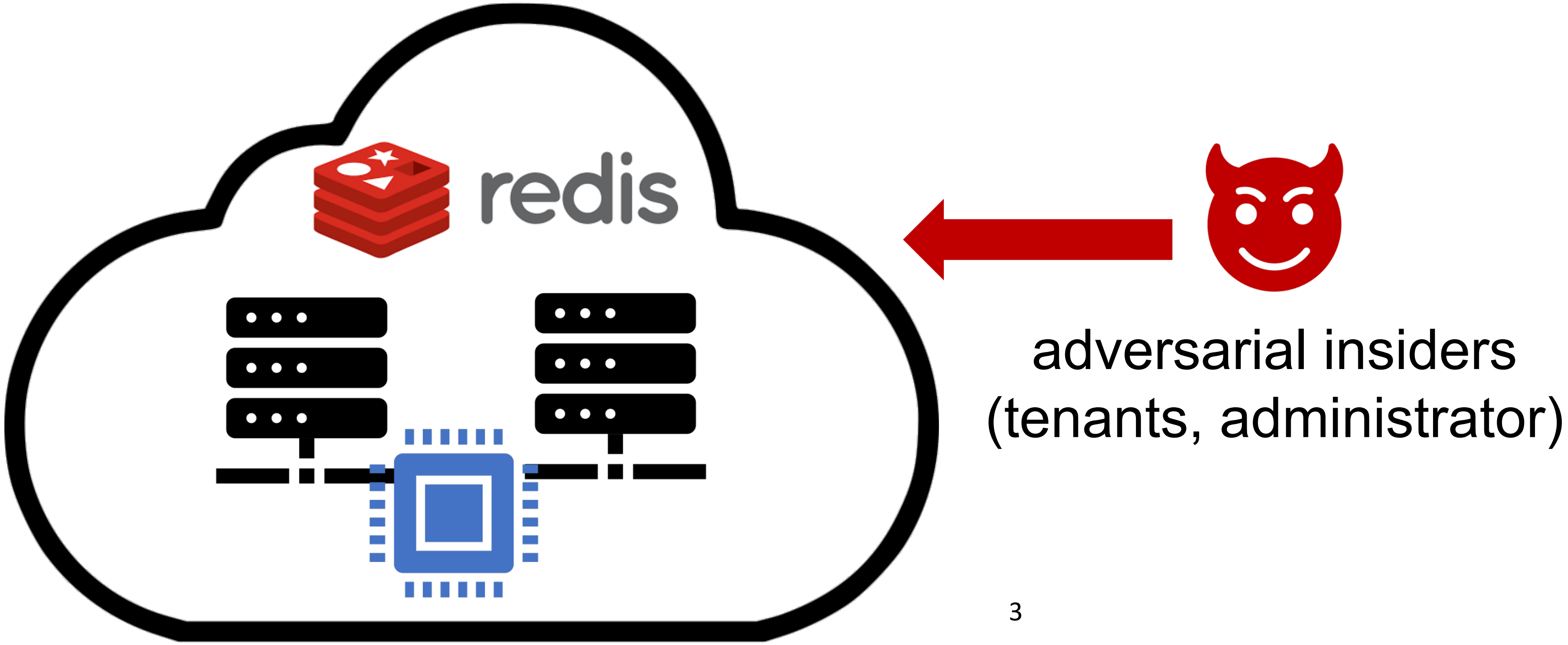
Trusted Key-Value Stores

- User data is exposed to adversarial insiders in cloud



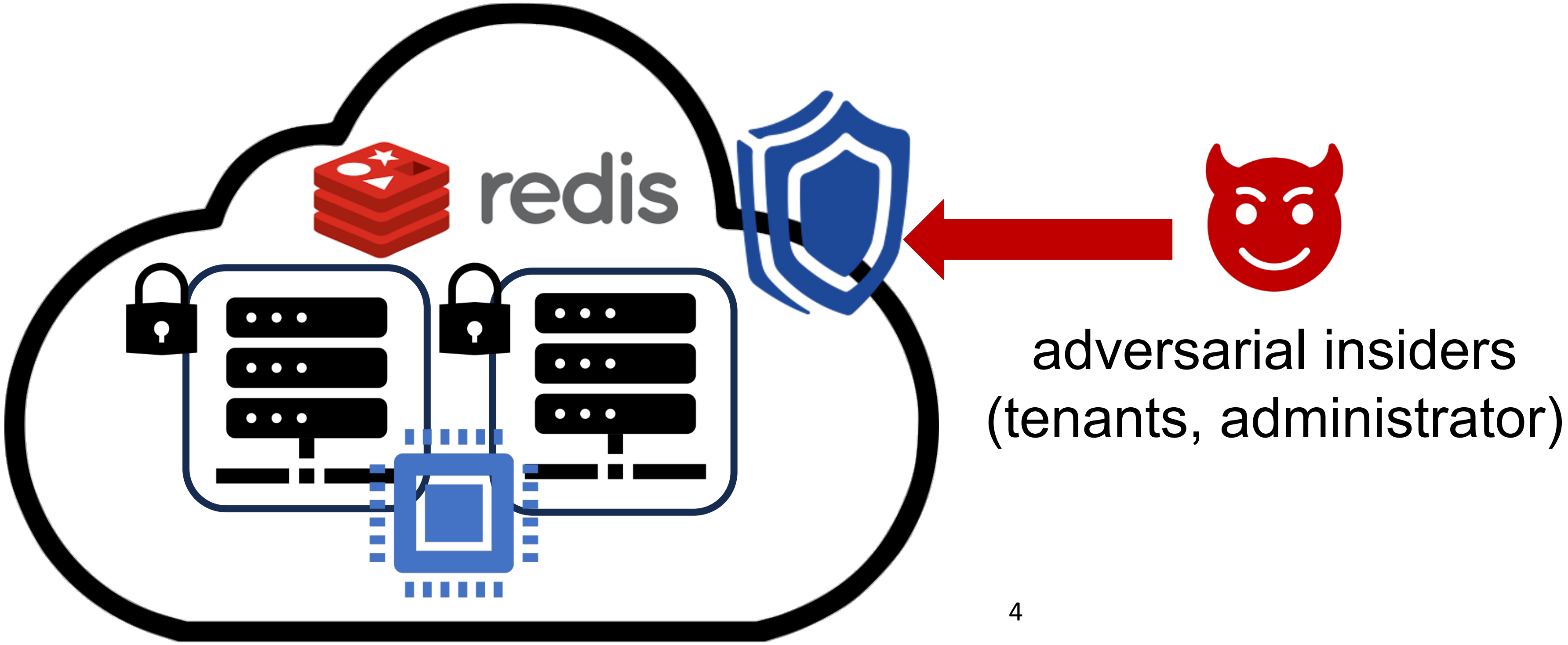
Trusted Key-Value Stores

- User data is exposed to adversarial insiders in cloud
- Hardware-based security supports



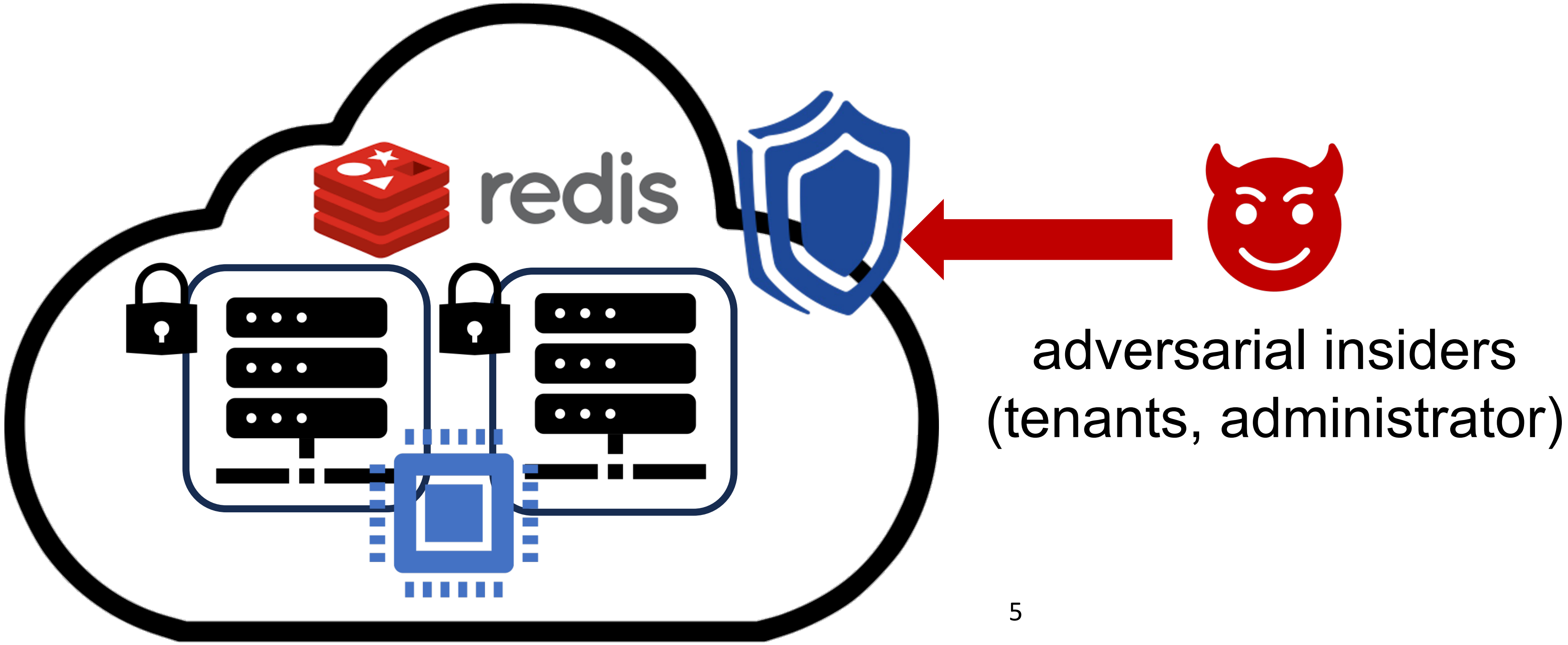
Trusted Key-Value Stores

- **User data is exposed to adversarial insiders in cloud**
- **Hardware-based security supports**
  - Provide *trusted execution environment* in remote server(s)



Trusted Key-Value Stores

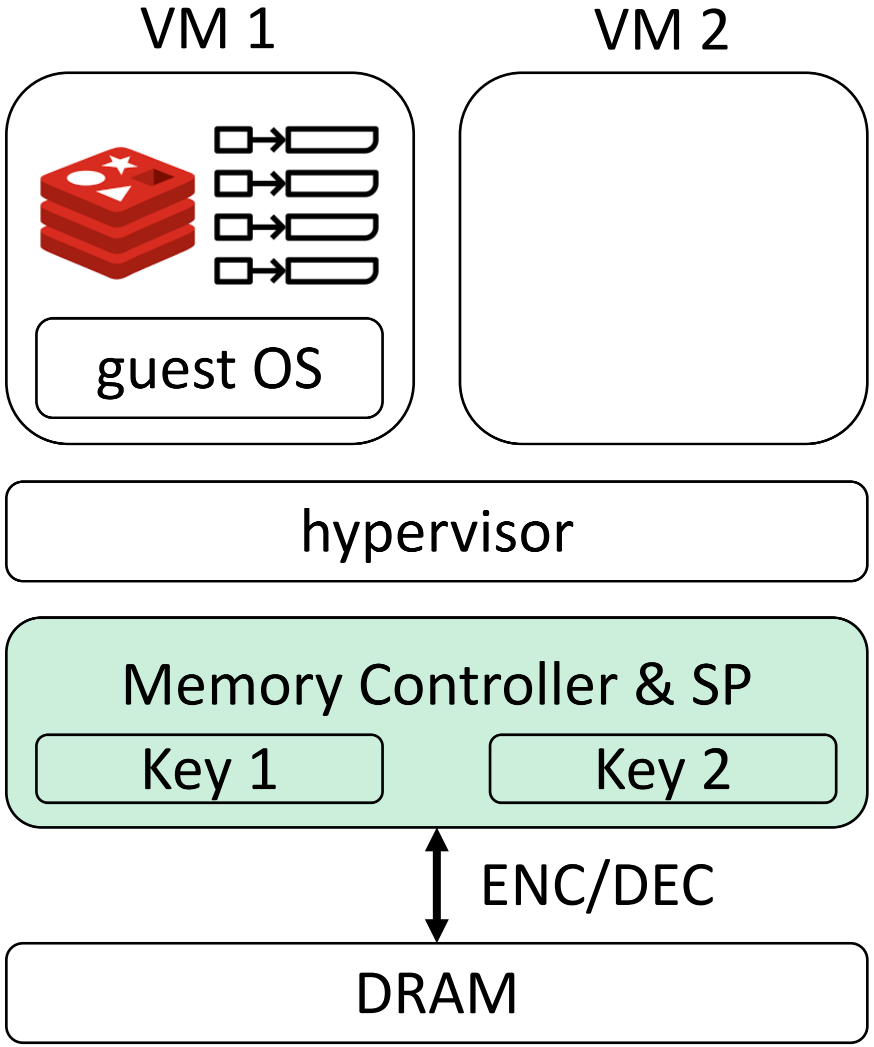
- **User data is exposed to adversarial insiders in cloud**
- **Hardware-based security supports**
  - Provide *trusted execution environment* in remote server(s)



Intel SGX

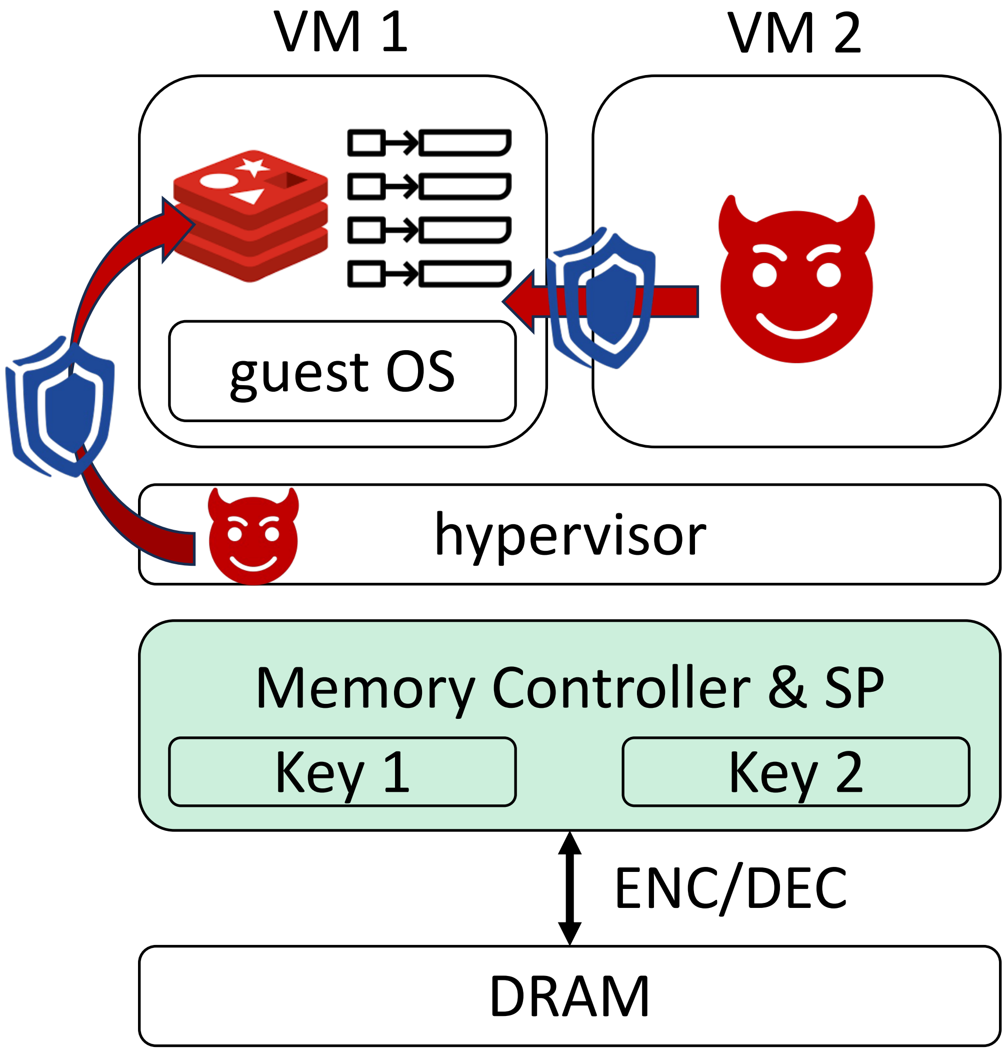
AMD SEV

Trusted Key-Value Stores with SEV



Secure Processor (SP) manages per-VM encryption keys

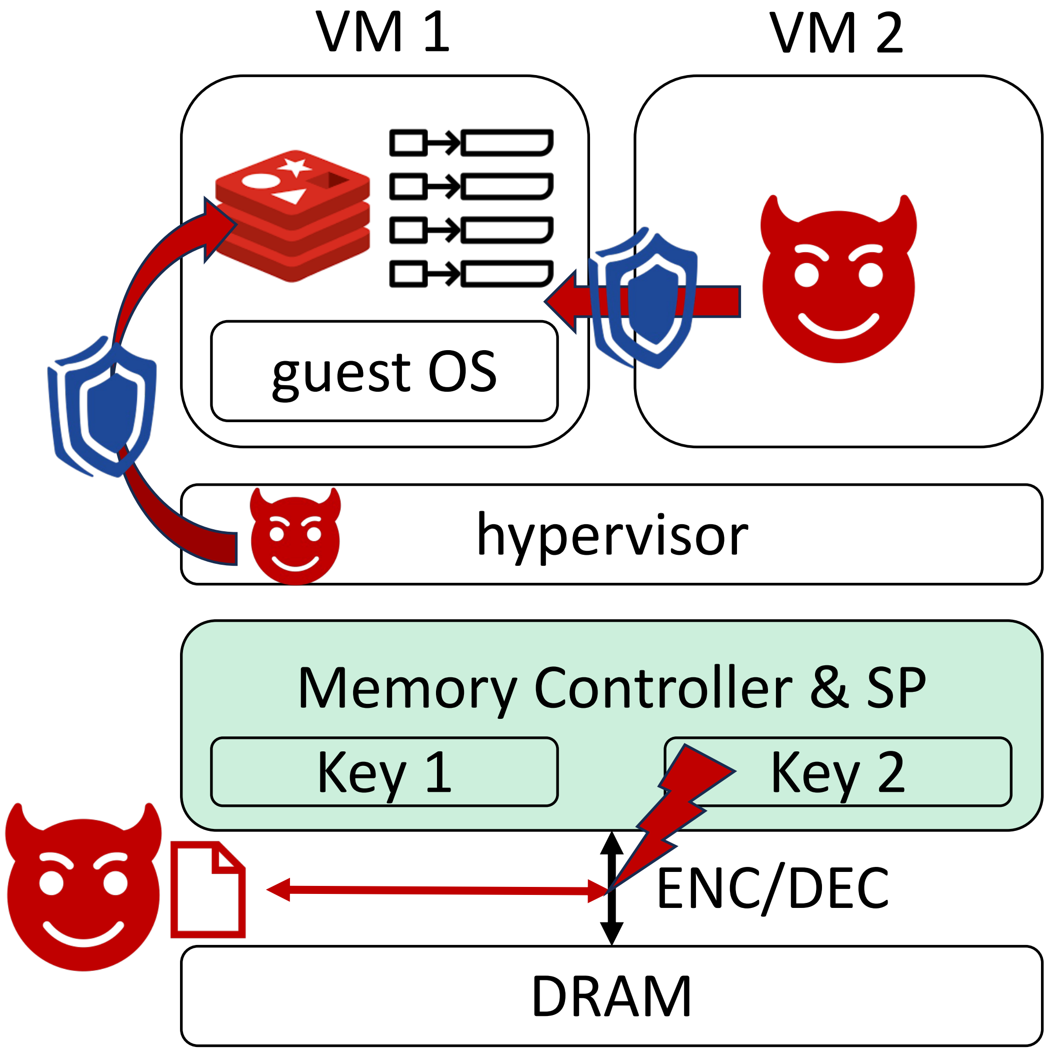
Trusted Key-Value Stores with SEV



Secure Processor (SP) manages per-VM encryption keys

Provides confidentiality and integrity from malicious VMs and hypervisor

Trusted Key-Value Stores with SEV

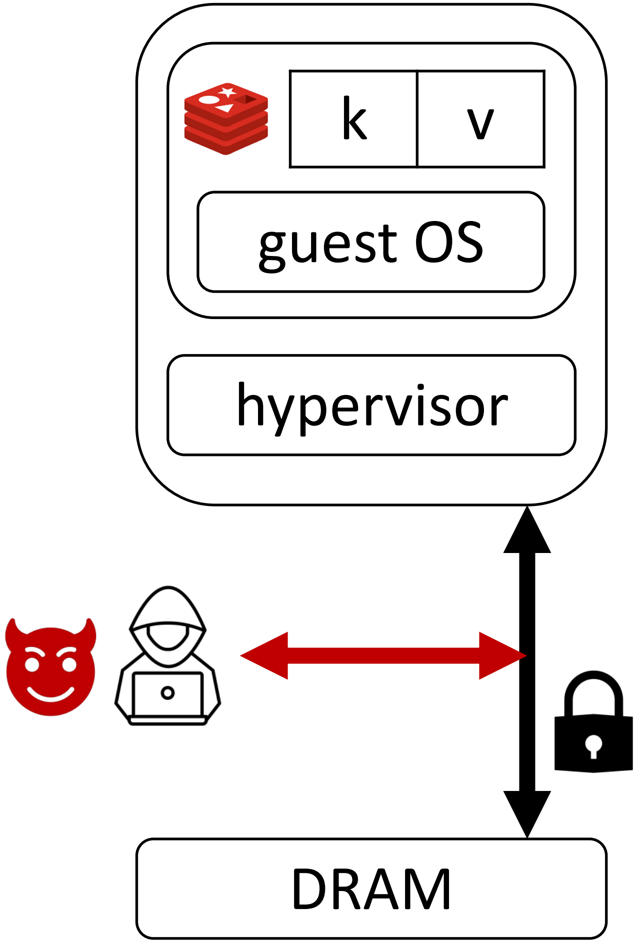


Secure Processor (SP) manages per-VM encryption keys

Provides confidentiality and integrity from malicious VMs and hypervisor

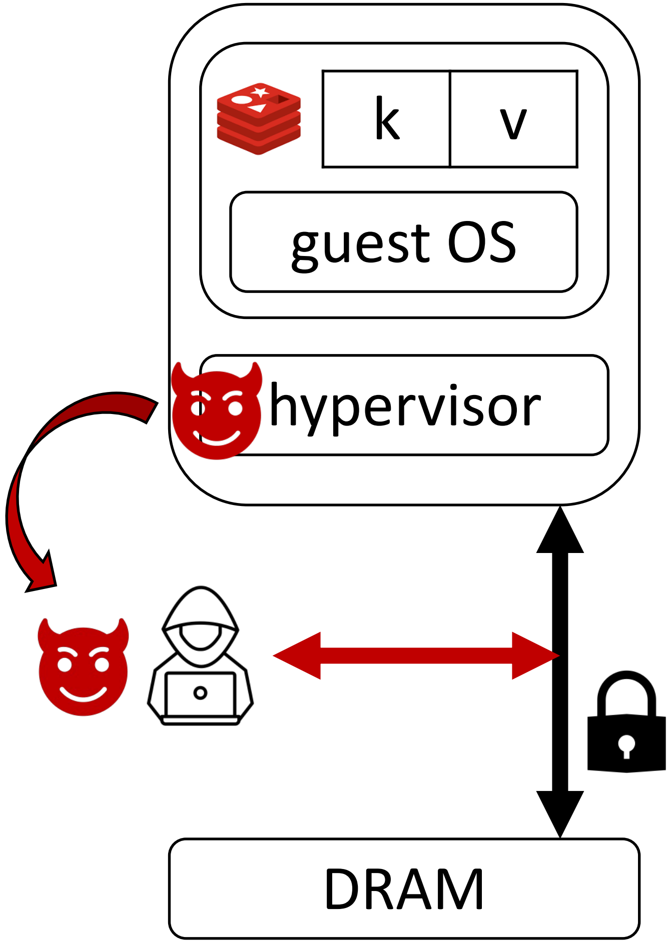
No integrity protection from physical adversaries

Physical Attacks on Key-Value Stores



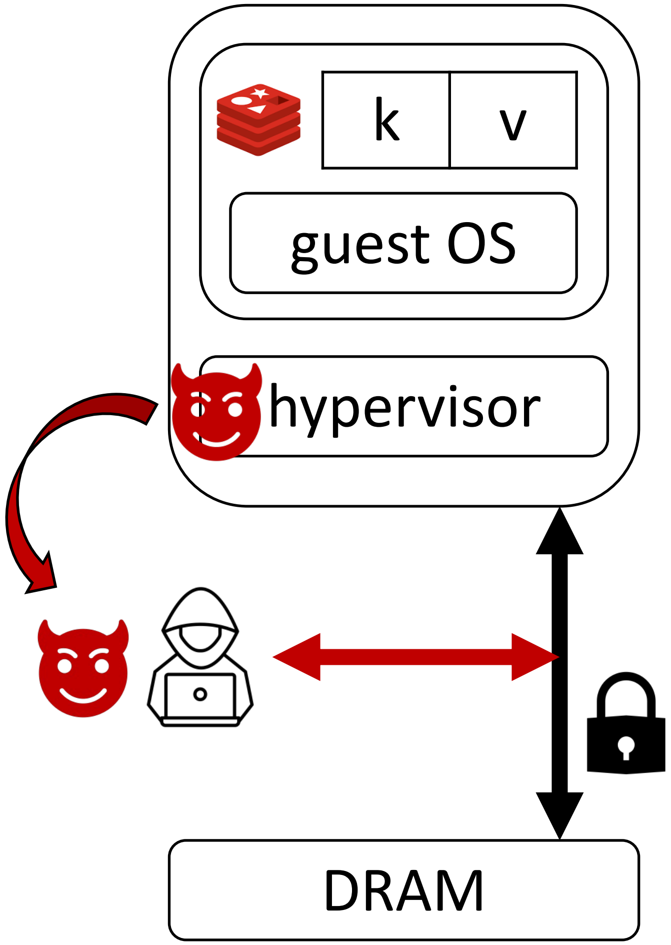
- **DRAM traffic is encrypted**

Physical Attacks on Key-Value Stores



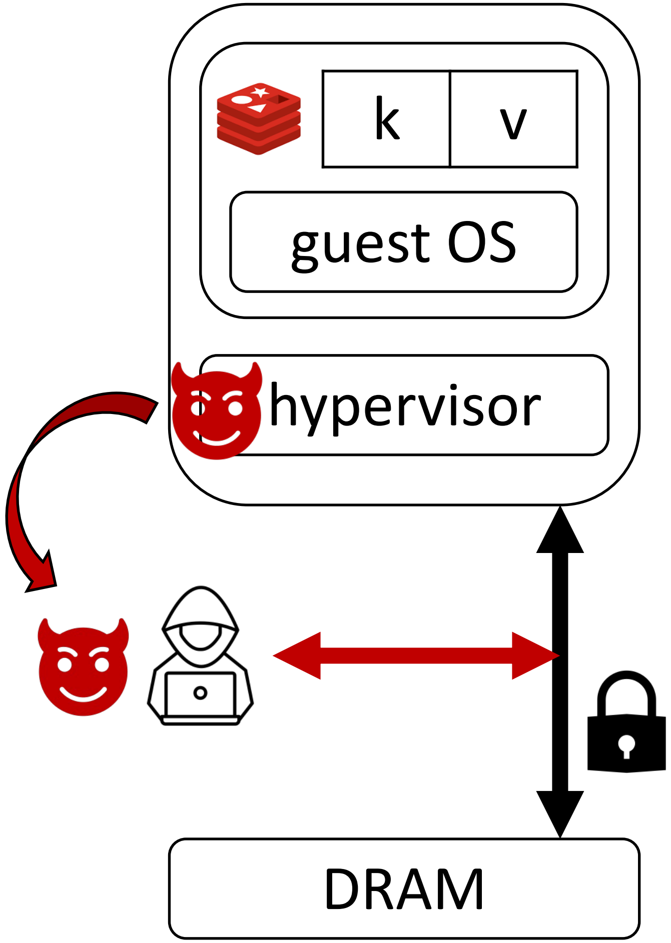
- **DRAM traffic is encrypted**
  - Need collaborator to control **when** and **what** to inject

Physical Attacks on Key-Value Stores



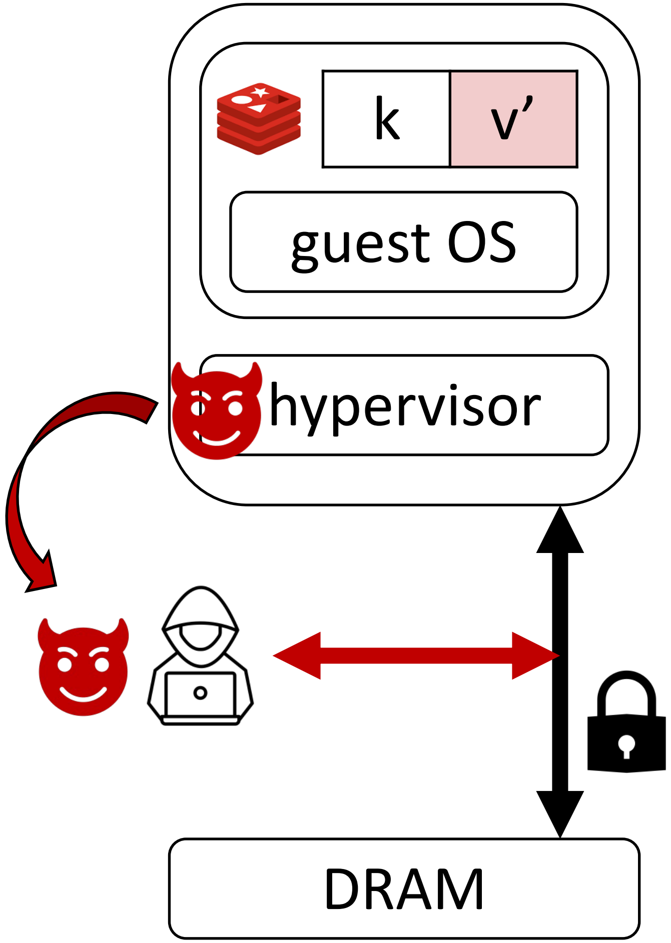
- **DRAM traffic is encrypted**
  - Need collaborator to control **when** and **what** to inject
- **Temporal (when) precision is limited**
  - Single instruction (single stepping)
  - Multiple instructions

Physical Attacks on Key-Value Stores



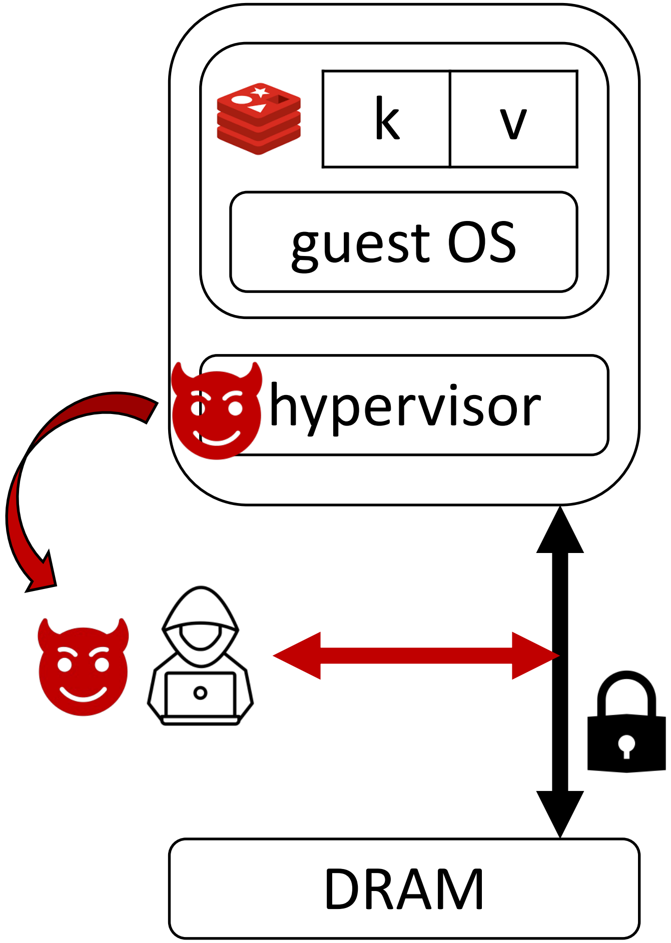
- **DRAM traffic is encrypted**
  - Need collaborator to control **when** and **what** to inject
- **Temporal (when) precision is limited**
  - Single instruction (single stepping)
  - Multiple instructions
- **Low temporal precision ➔ targets k-v pairs**
  - Arbitrary corruption

Physical Attacks on Key-Value Stores



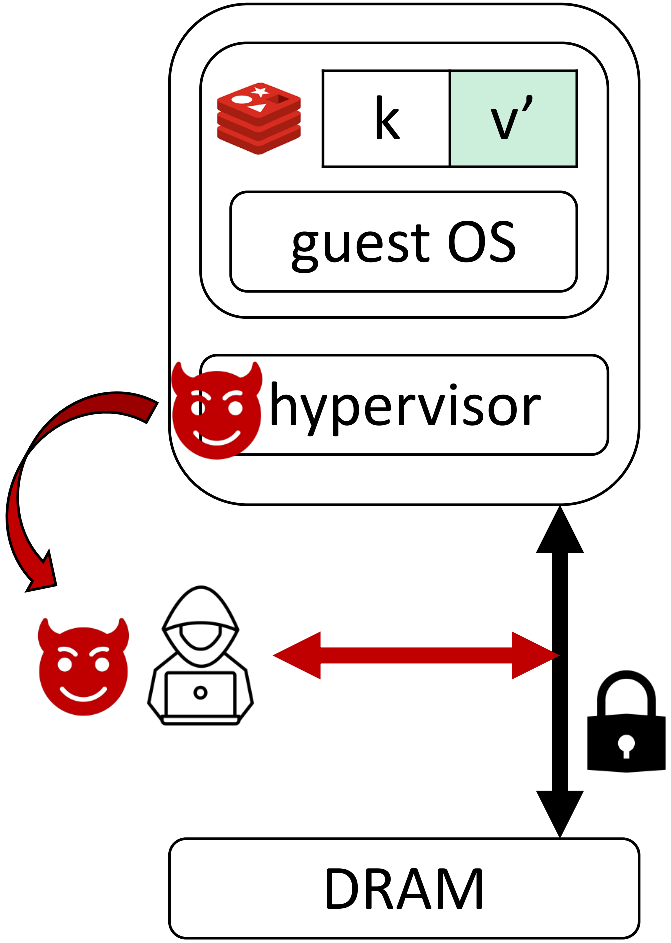
- **DRAM traffic is encrypted**
  - Need collaborator to control **when** and **what** to inject
- **Temporal (when) precision is limited**
  - Single instruction (single stepping)
  - Multiple instructions
- **Low temporal precision ➔ targets k-v pairs**
  - Arbitrary corruption

Physical Attacks on Key-Value Stores



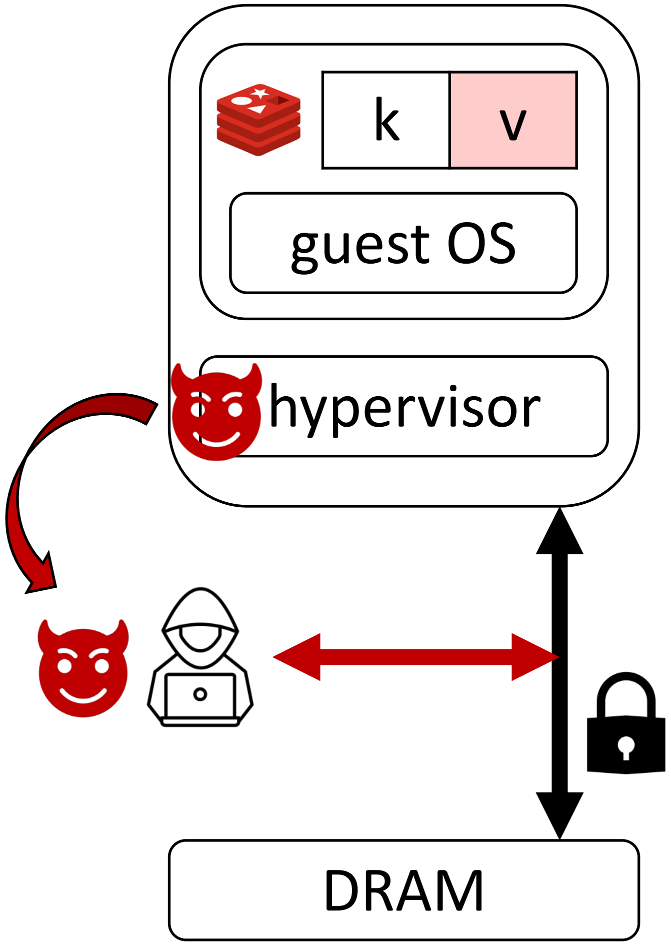
- **DRAM traffic is encrypted**
  - Need collaborator to control **when** and **what** to inject
- **Temporal (when) precision is limited**
  - Single instruction (single stepping)
  - Multiple instructions
- **Low temporal precision ➔ targets k-v pairs**
  - Arbitrary corruption
  - Rollback

Physical Attacks on Key-Value Stores



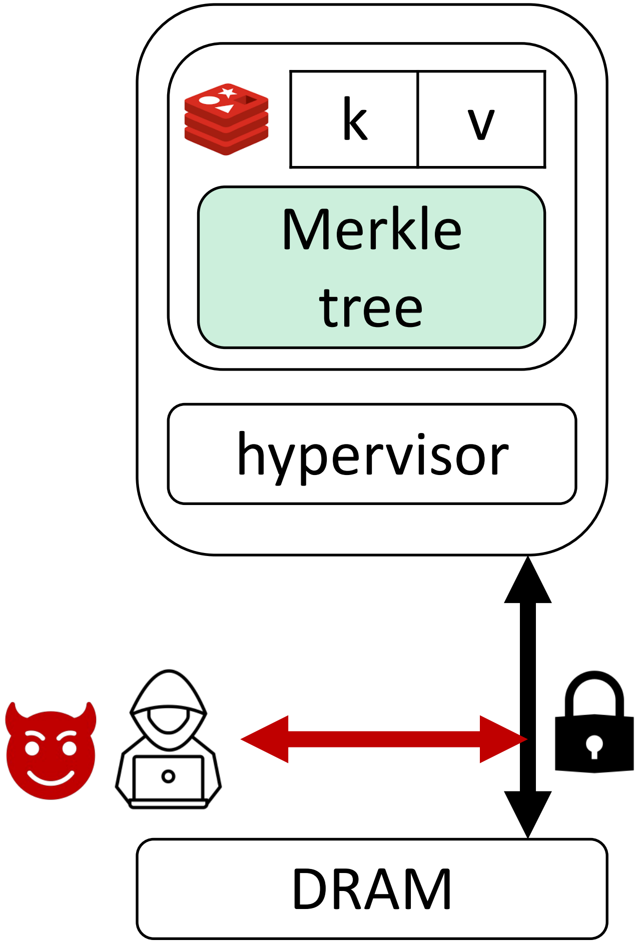
- **DRAM traffic is encrypted**
  - Need collaborator to control **when** and **what** to inject
- **Temporal (when) precision is limited**
  - Single instruction (single stepping)
  - Multiple instructions
- **Low temporal precision ➔ targets k-v pairs**
  - Arbitrary corruption
  - Rollback

Physical Attacks on Key-Value Stores



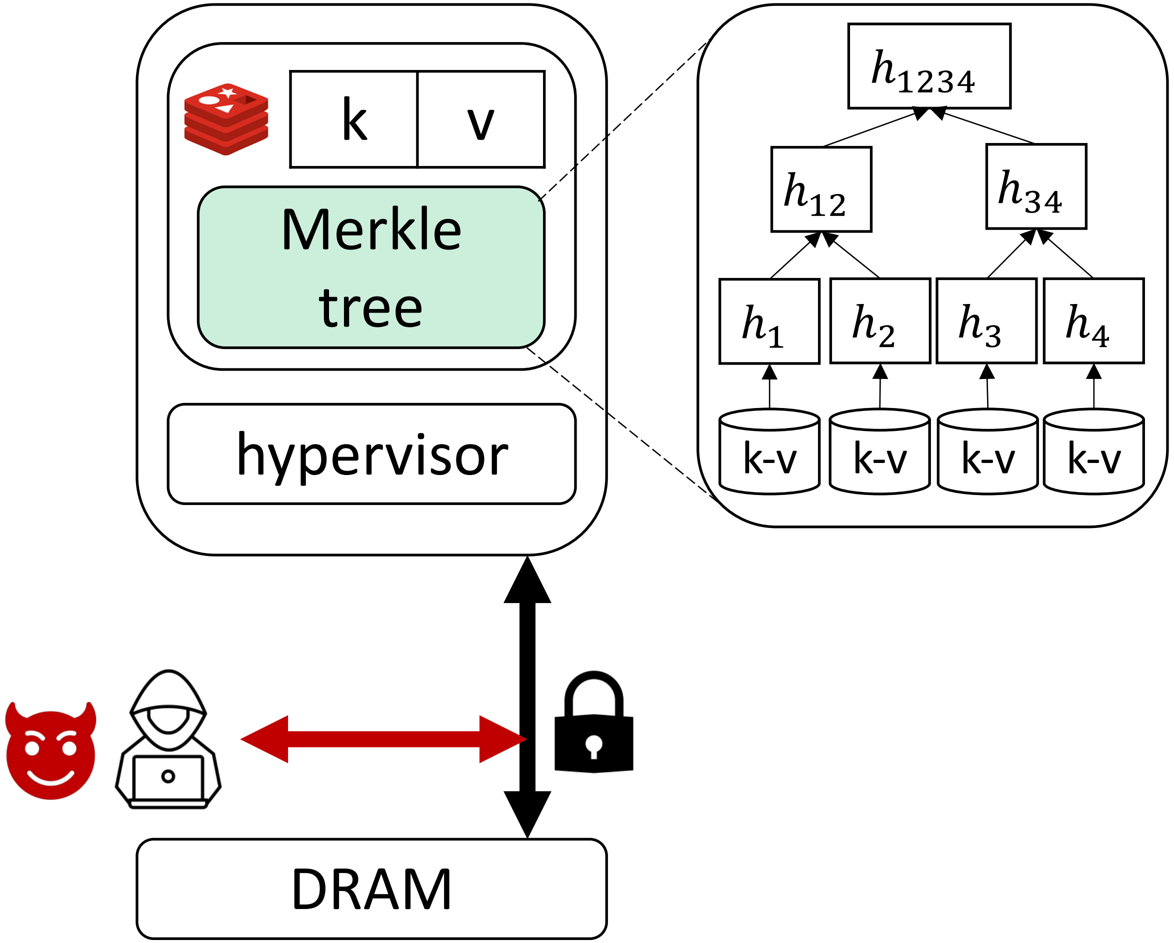
- **DRAM traffic is encrypted**
  - Need collaborator to control **when** and **what** to inject
- **Temporal (when) precision is limited**
  - Single instruction (single stepping)
  - Multiple instructions
- **Low temporal precision ➔ targets k-v pairs**
  - Arbitrary corruption
  - Rollback

Strawman Solution



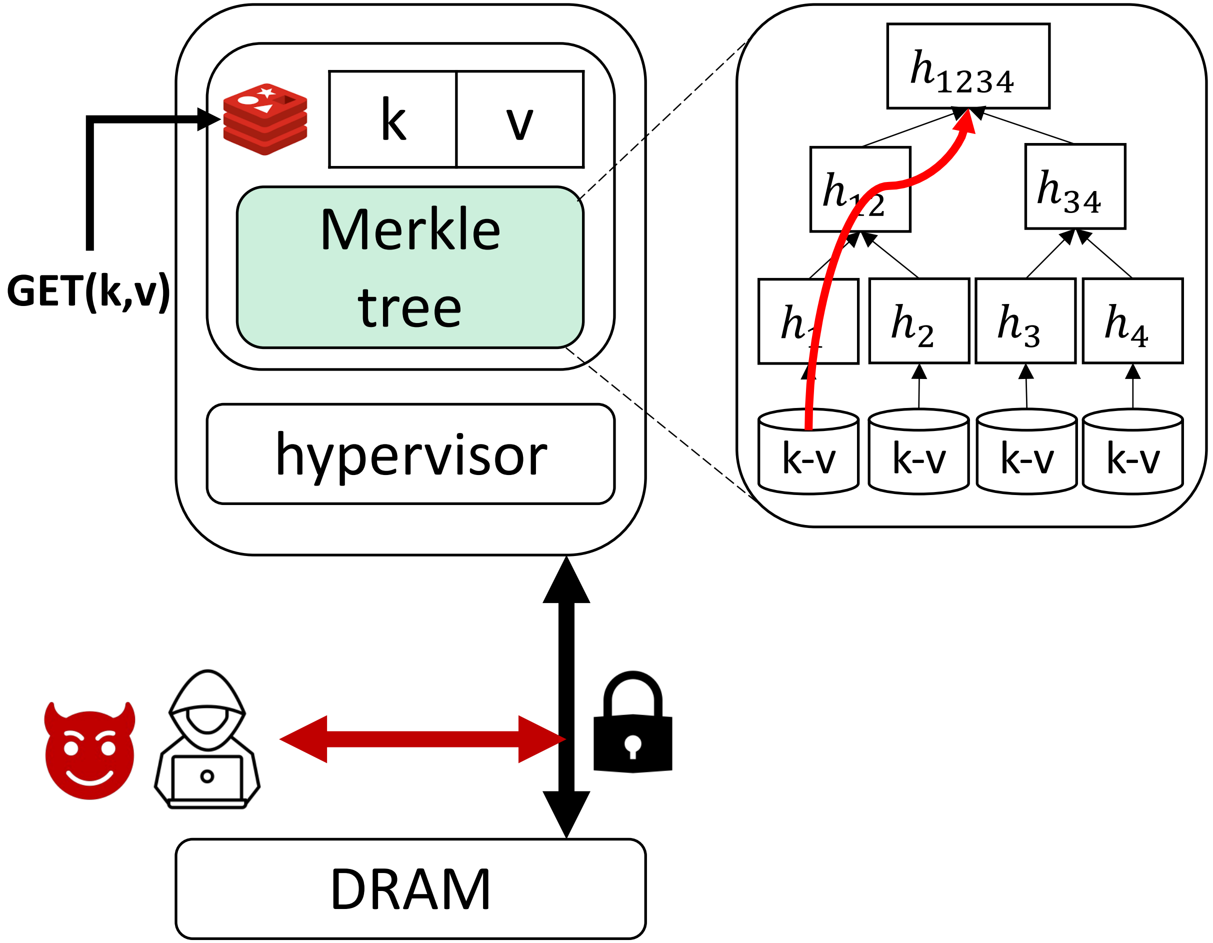
- **Merkle tree (MT)**
  - SW-only data structure for data integrity

Strawman Solution



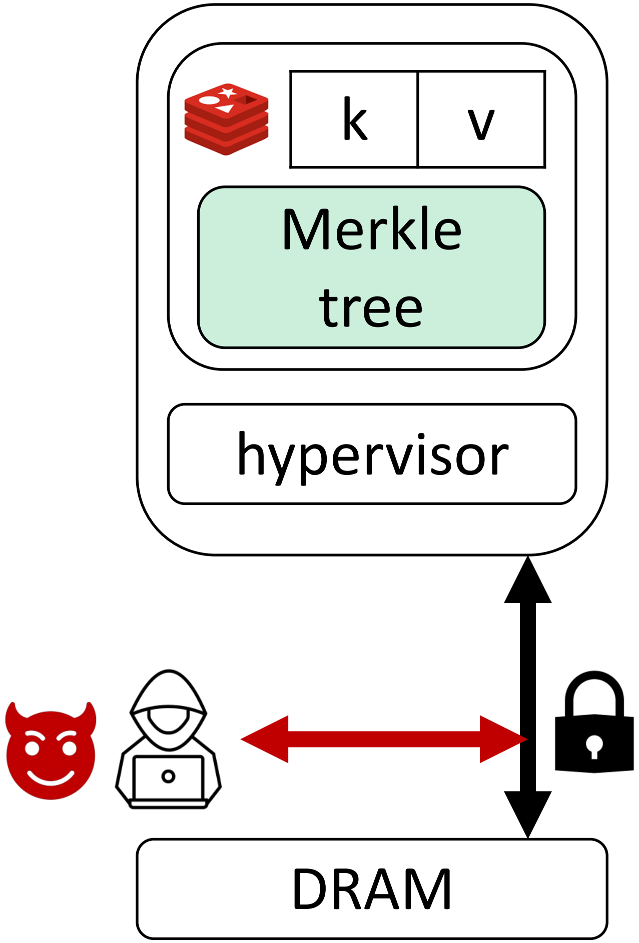
- **Merkle tree (MT)**
  - SW-only data structure for data integrity

Strawman Solution



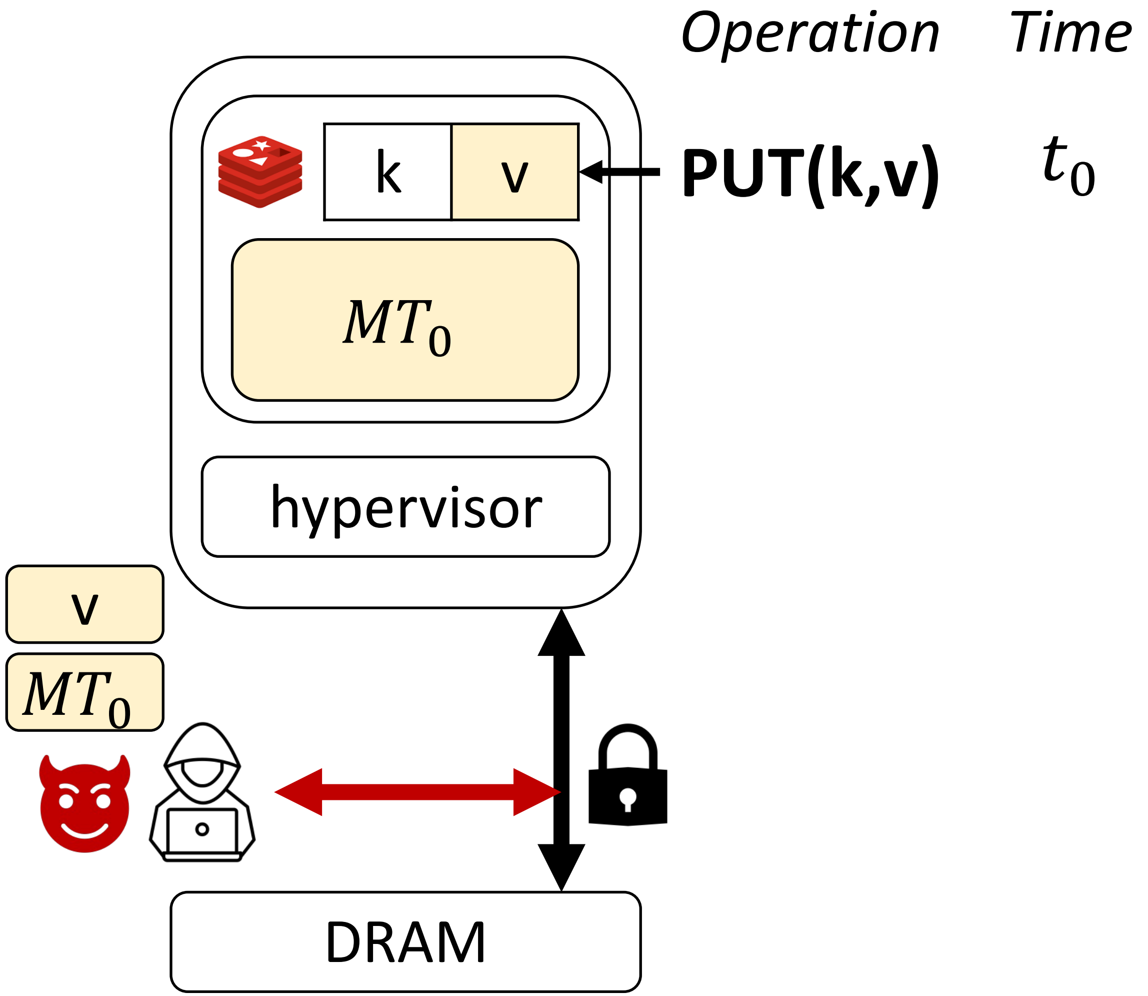
- **Merkle tree (MT)**
  - SW-only data structure for data integrity
  - Update on PUT, verify on GET

Strawman Solution



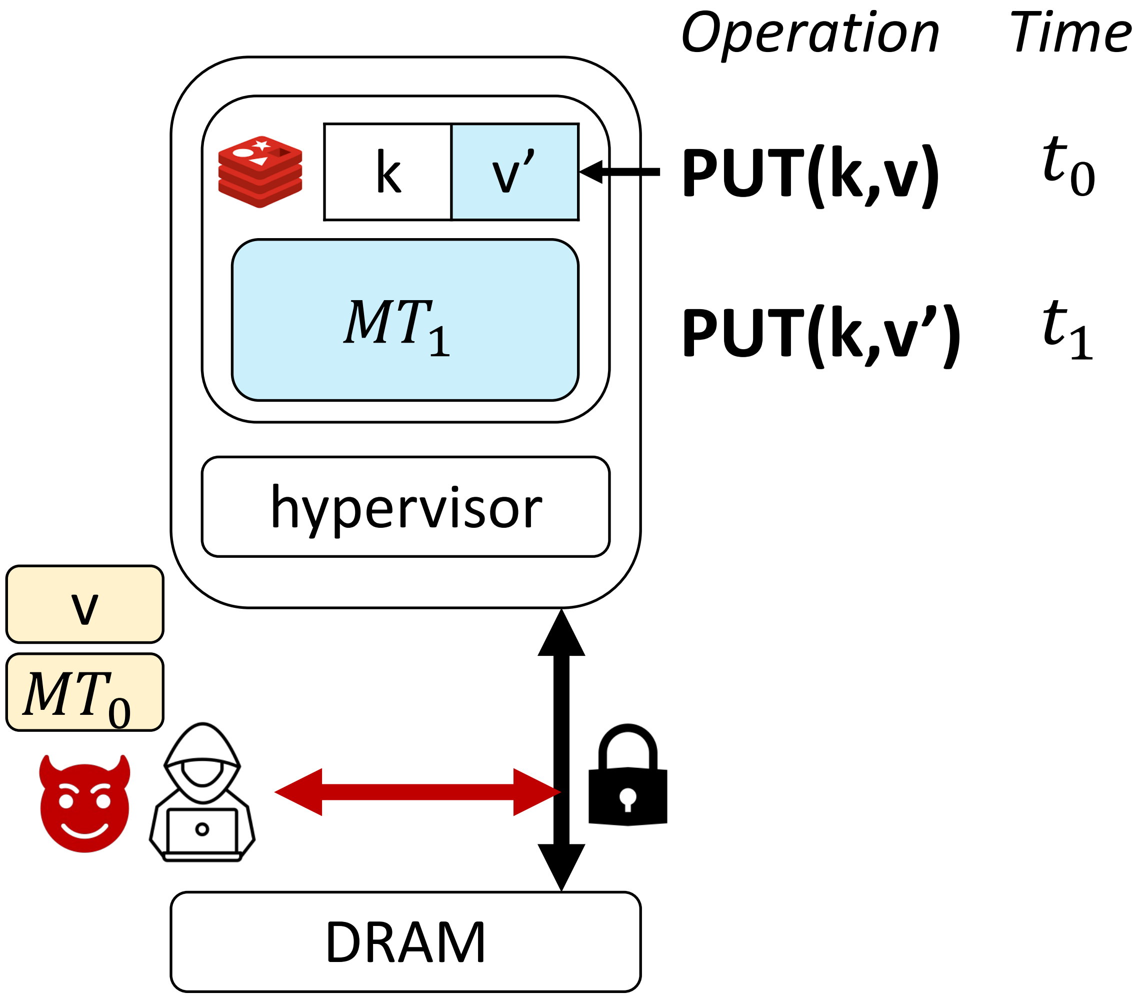
- **Merkle tree (MT)**
  - SW-only data structure for data integrity
  - Update on PUT, verify on GET
- **Problem**
  - Replay MT with k-v pair

Strawman Solution



- **Merkle tree (MT)**
  - SW-only data structure for data integrity
  - Update on PUT, verify on GET
- **Problem**
  - Replay MT with k-v pair

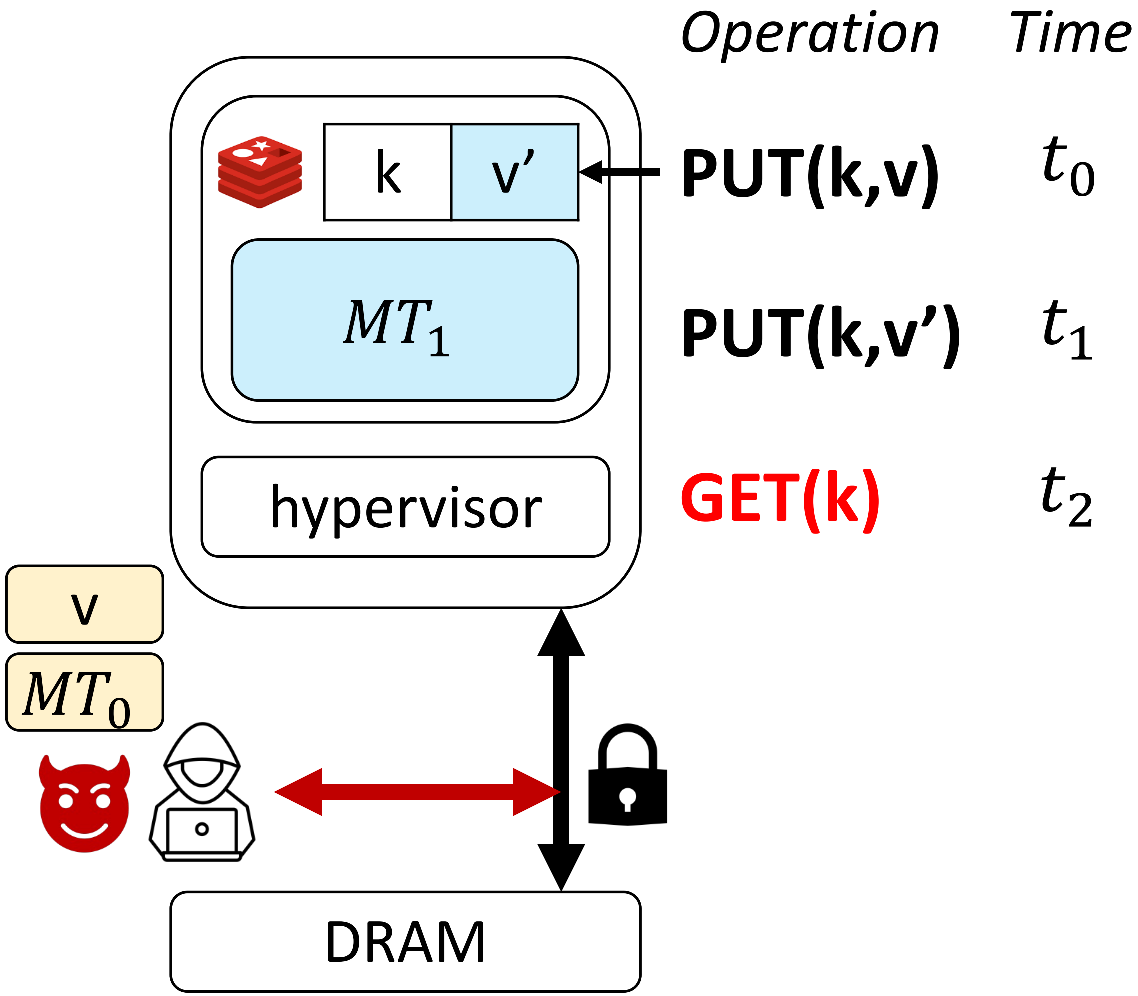
Strawman Solution



- **Merkle tree (MT)**
  - SW-only data structure for data integrity
  - Update on PUT, verify on GET

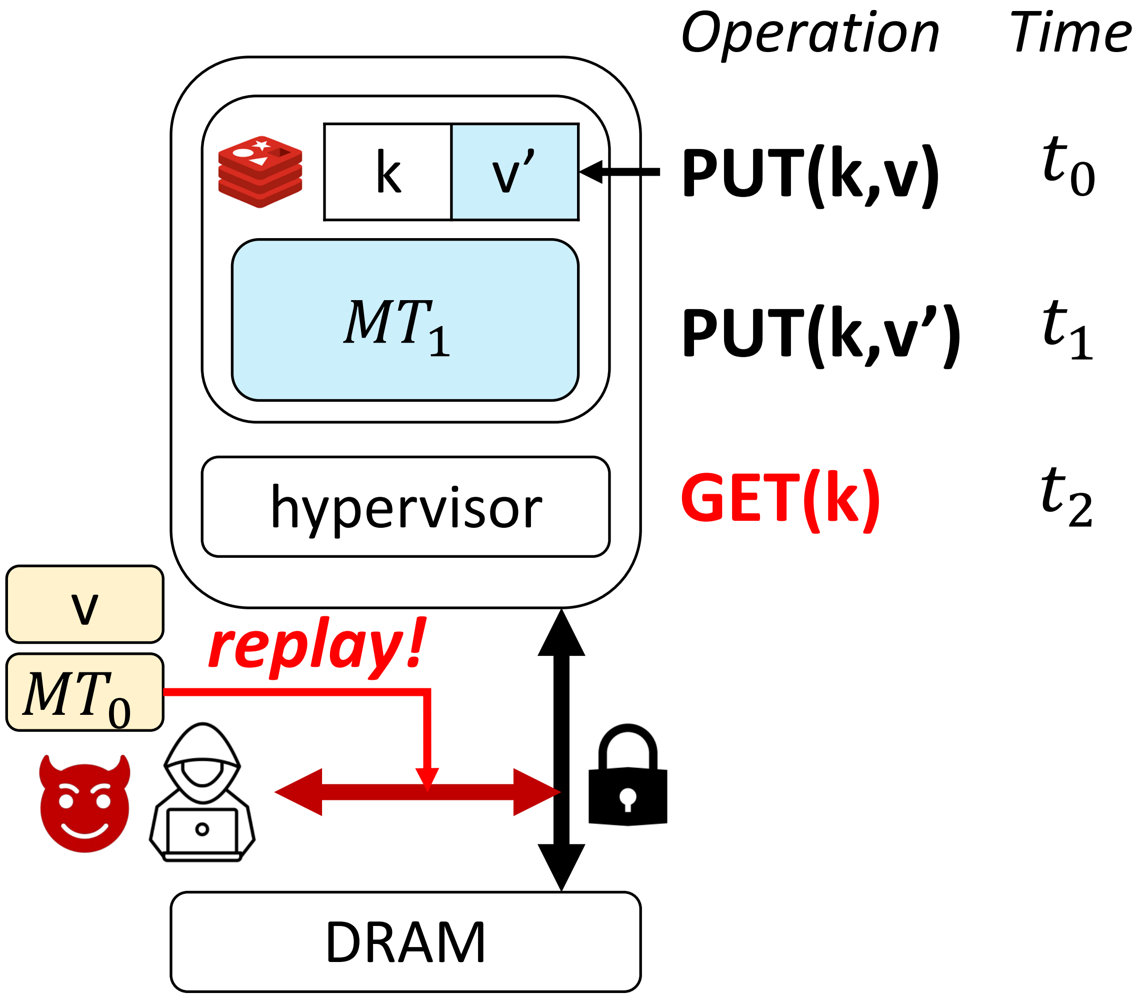
- **Problem**
  - Replay MT with k-v pair

Strawman Solution



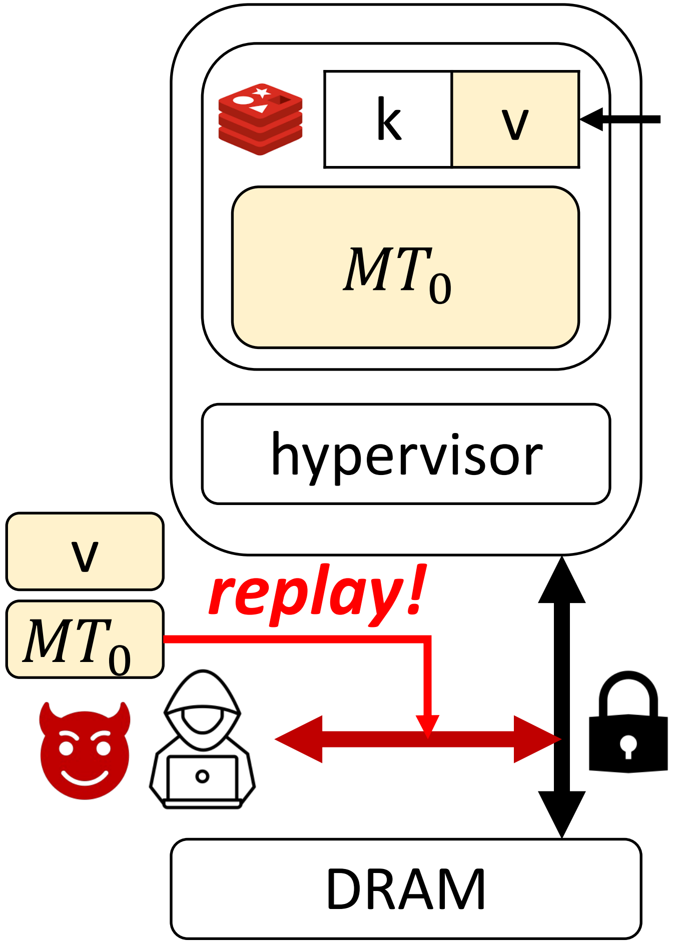
- **Merkle tree (MT)**
  - SW-only data structure for data integrity
  - Update on PUT, verify on GET
- **Problem**
  - Replay MT with k-v pair

Strawman Solution



- **Merkle tree (MT)**
  - SW-only data structure for data integrity
  - Update on PUT, verify on GET
- **Problem**
  - Replay MT with k-v pair

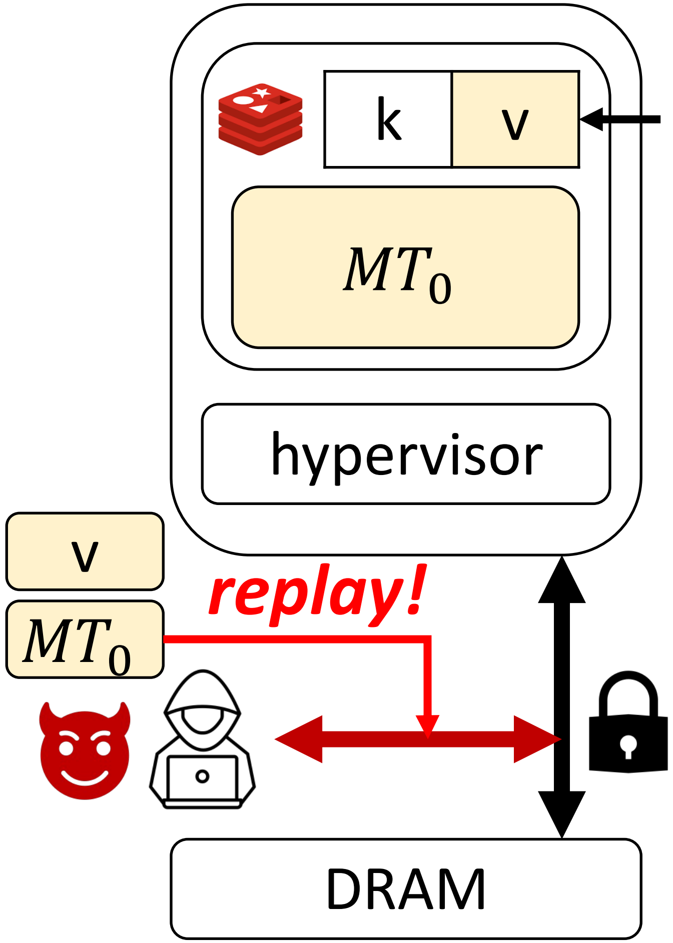
Strawman Solution



Operation	*Time*
PUT(k,v)	t_0
PUT(k,v’)	t_1
GET(k)	t_2

- **Merkle tree (MT)**
  - SW-only data structure for data integrity
  - Update on PUT, verify on GET
- **Problem**
  - Replay MT with k-v pair

Strawman Solution

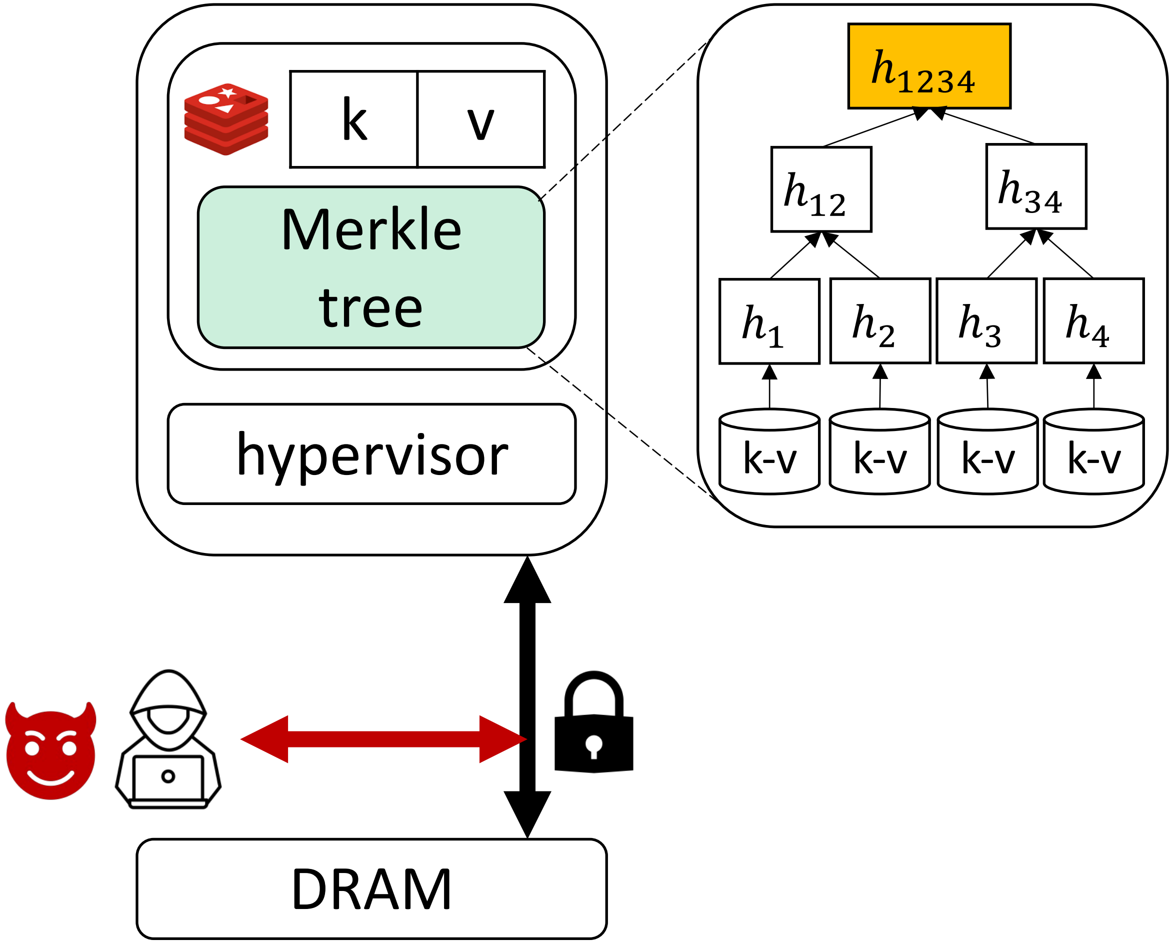


Operation	*Time*
PUT(k,v)	t_0
PUT(k,v’)	t_1
GET(k)	t_2

- **Merkle tree (MT)**
  - SW-only data structure for data integrity
  - Update on PUT, verify on GET
- **Problem**
  - Replay MT with k-v pair

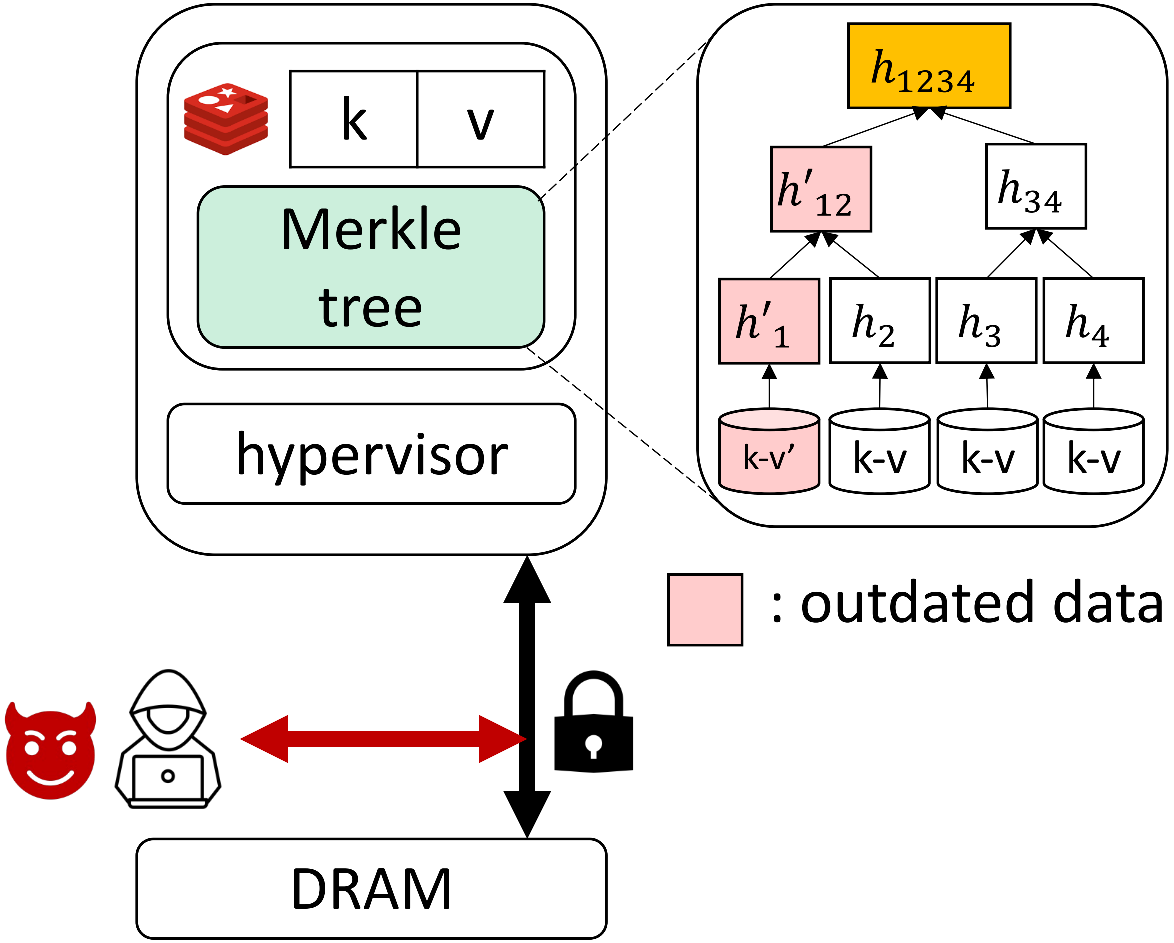
Returns outdated value ‘v’ passing Merkle tree verification

Our Observations



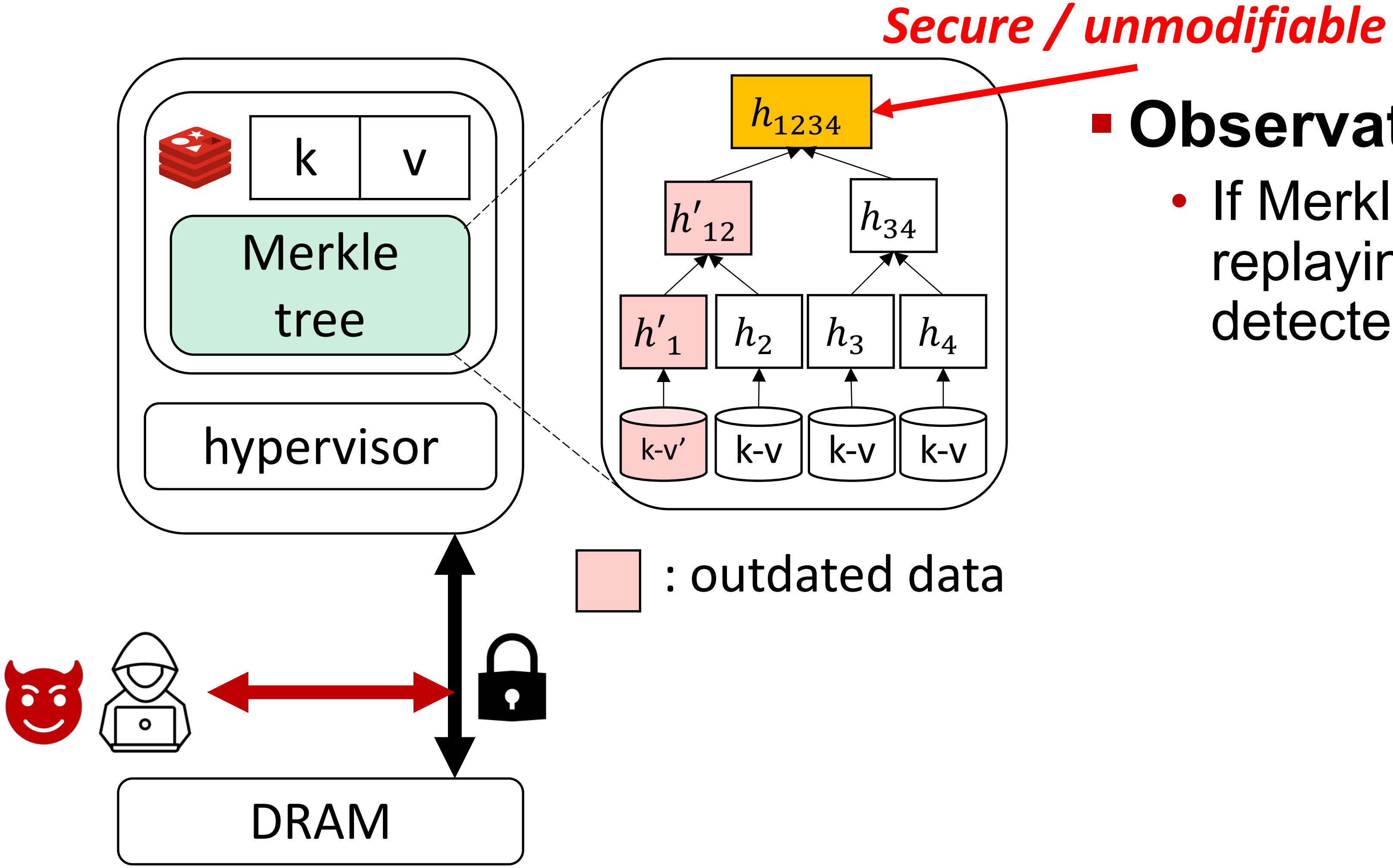
- **Observation 1**
  - If Merkle root is secure, modifying/ replaying internal nodes will be detected during root computation

Our Observations



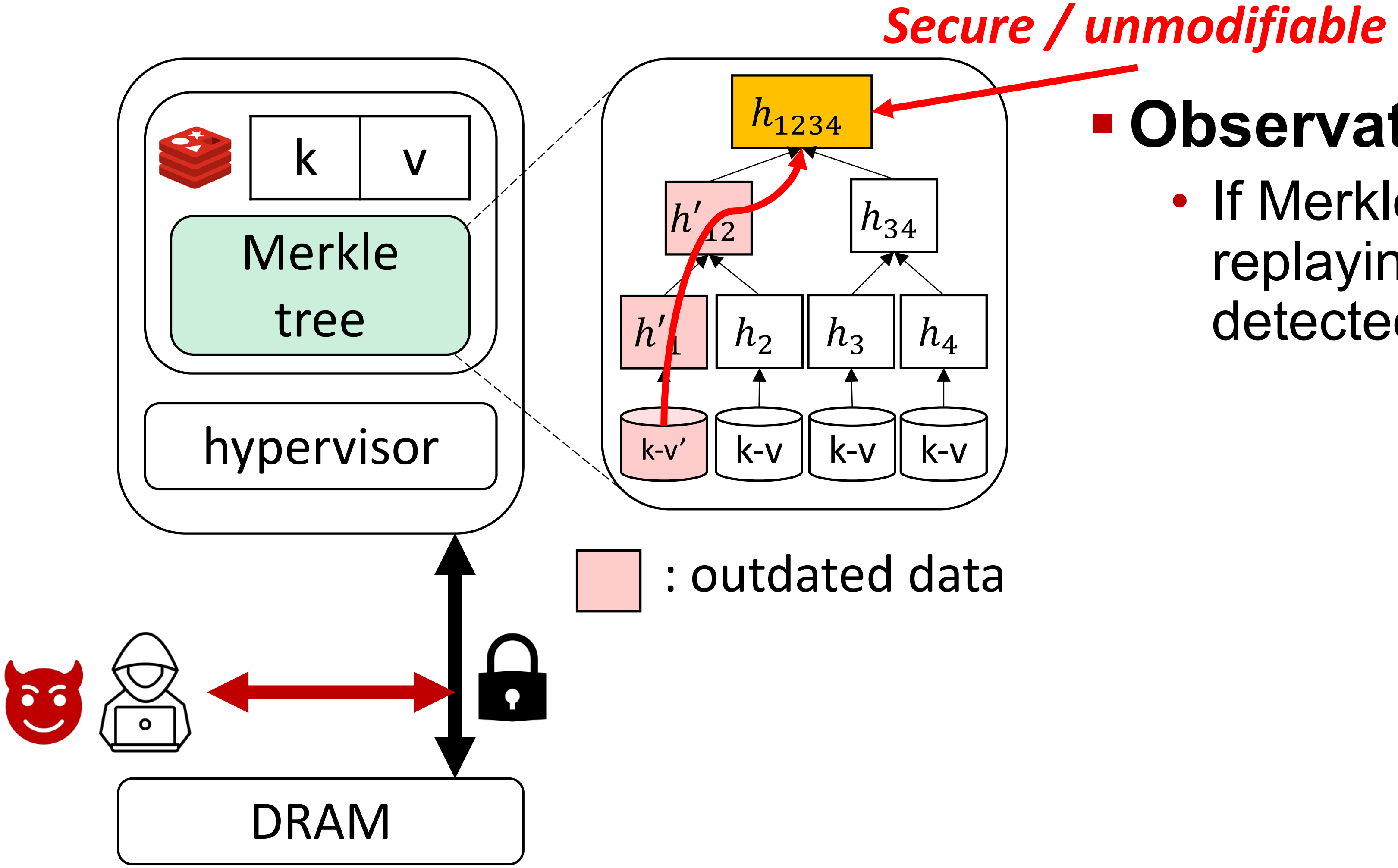
- **Observation 1**
  - If Merkle root is secure, modifying/ replaying internal nodes will be detected during root computation

Our Observations



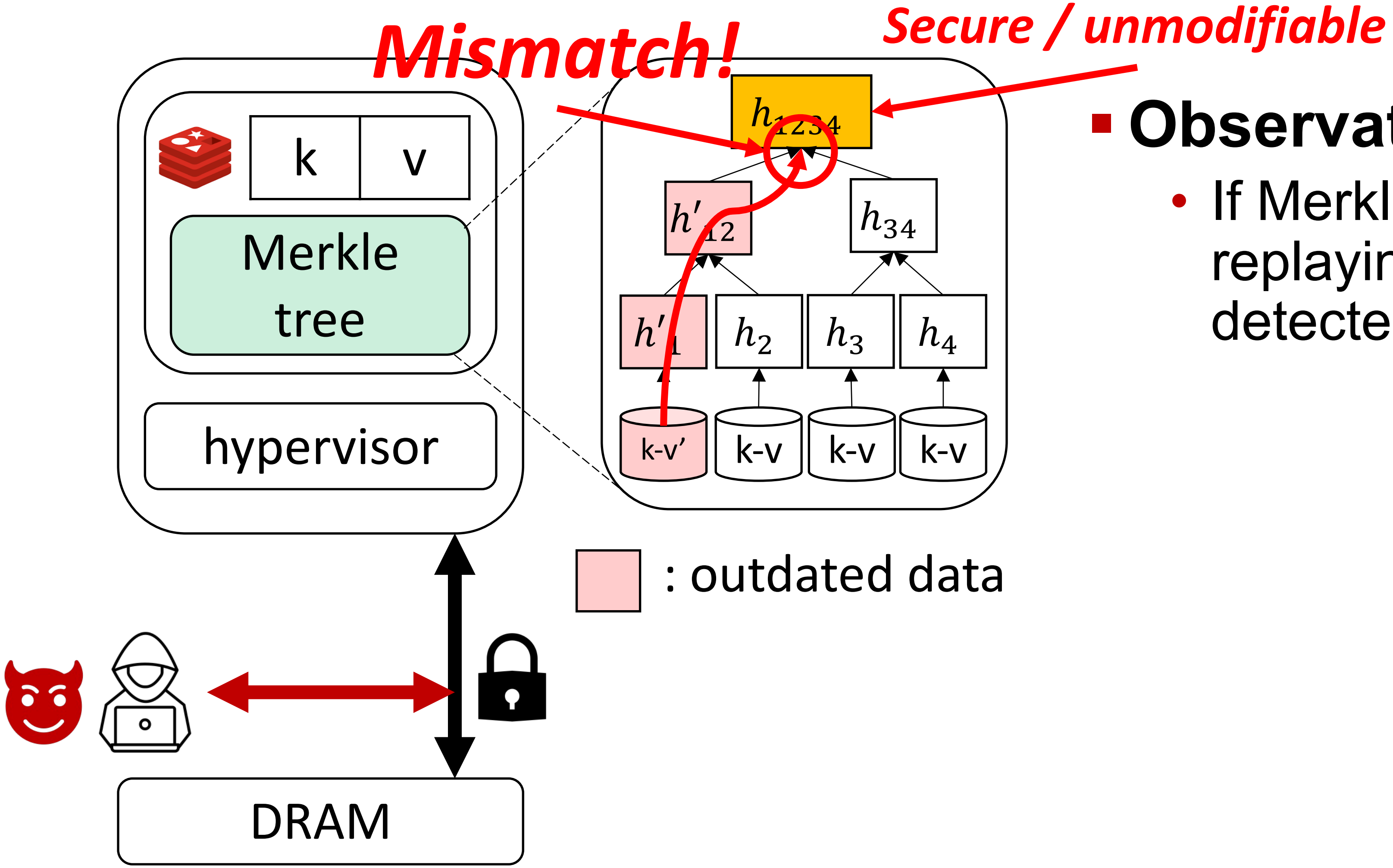
- **Observation 1**
  - If Merkle root is secure, modifying/ replaying internal nodes will be detected during root computation

Our Observations



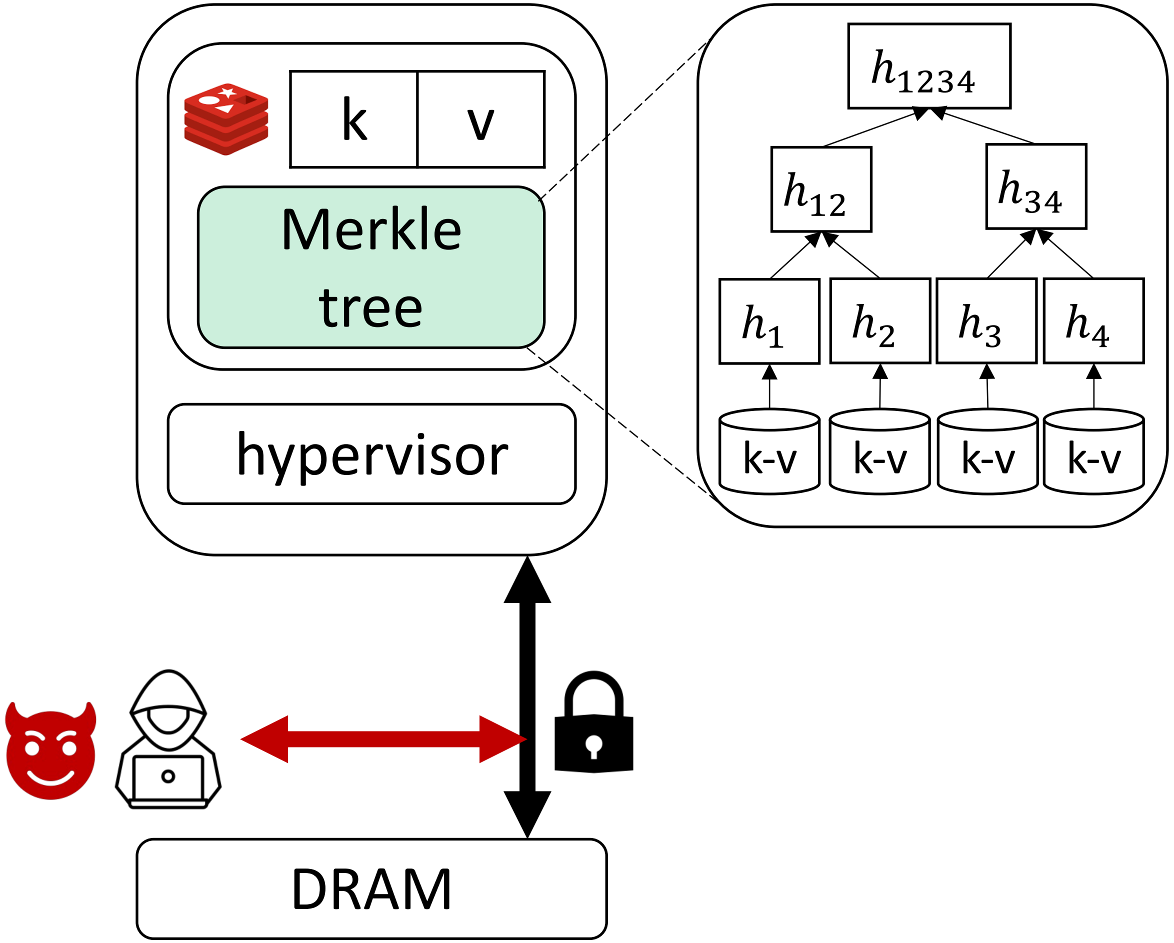
- **Observation 1**
  - If Merkle root is secure, modifying/ replaying internal nodes will be detected during root computation

Our Observations



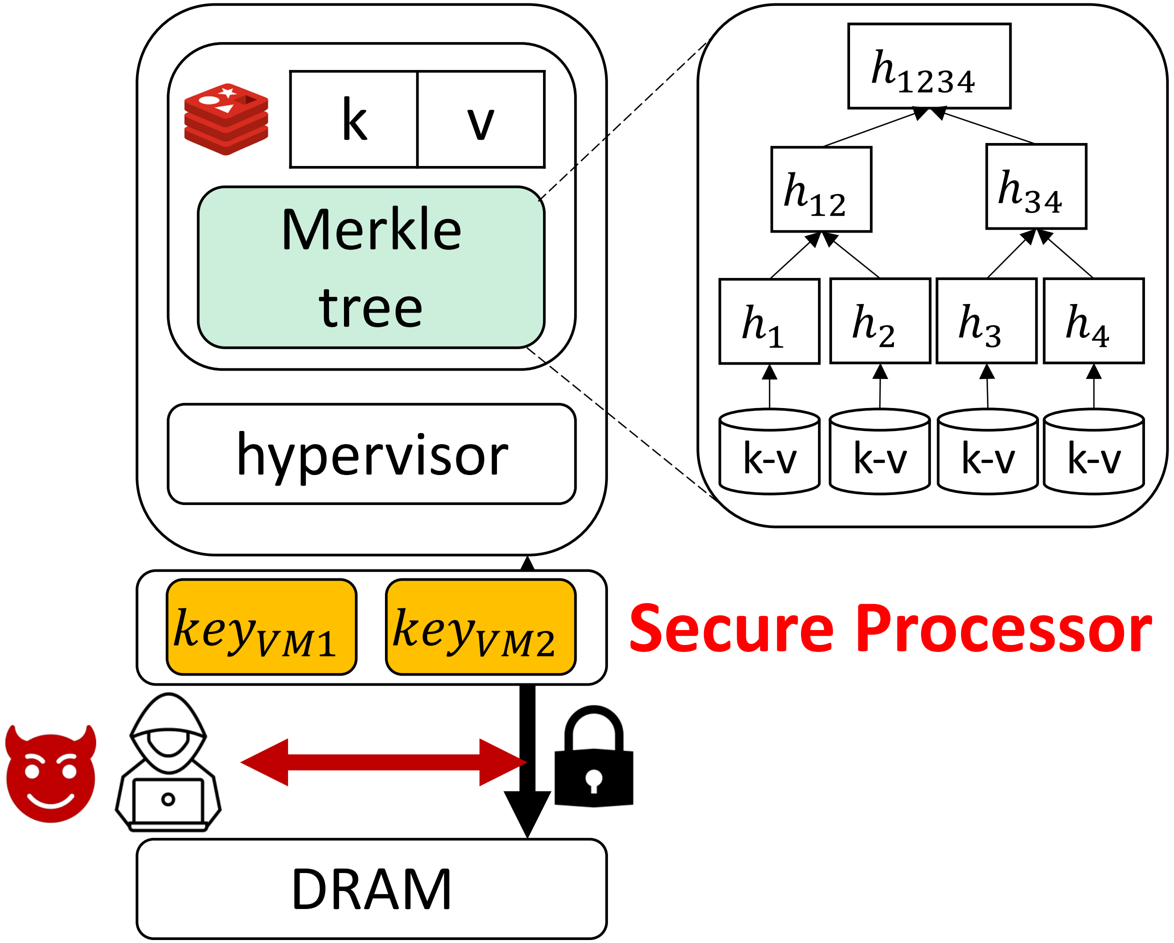
- **Observation 1**
  - If Merkle root is secure, modifying/ replaying internal nodes will be detected during root computation

Our Observations



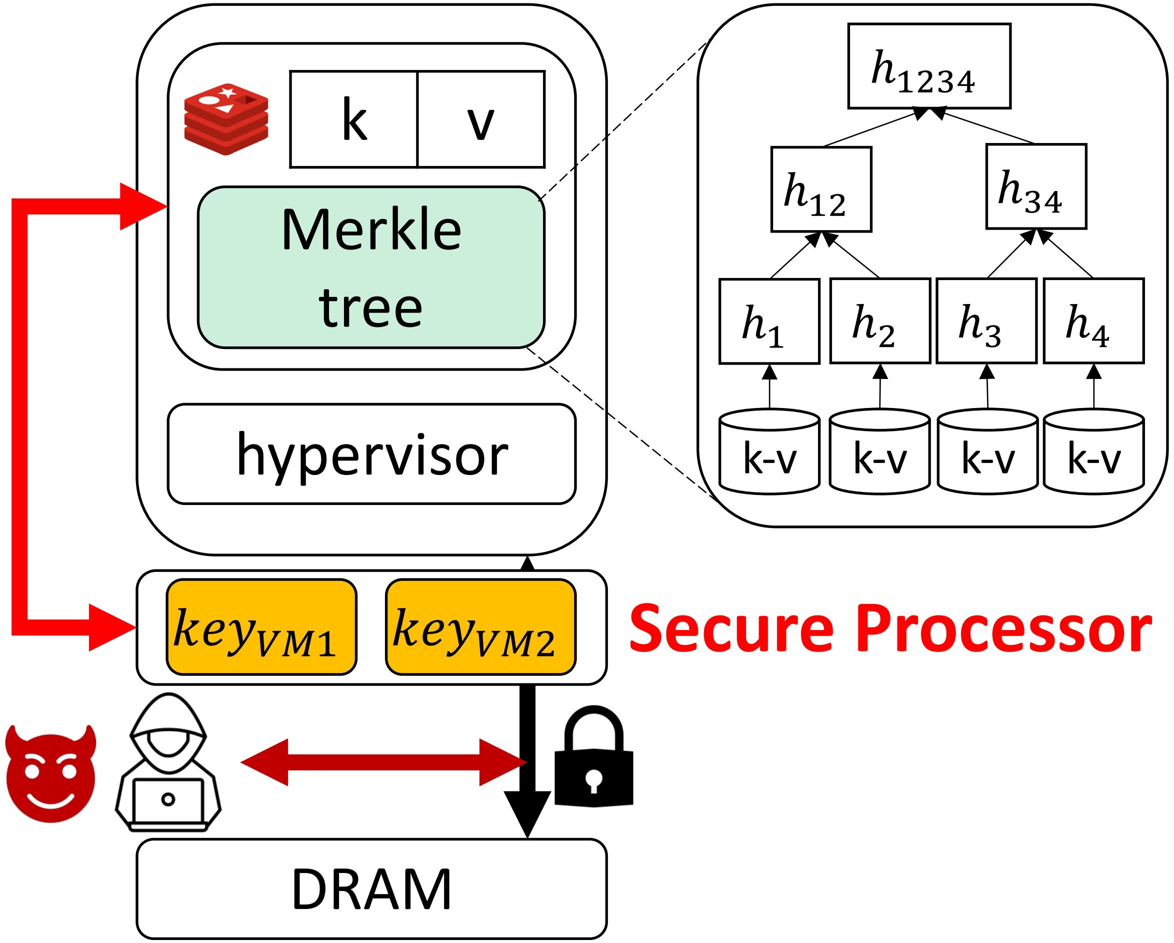
- **Observation 1**
  - If Merkle root is secure, modifying/ replaying internal nodes will be detected during root computation
- **Observation 2**
  - Different encryption keys are used for respective VMs

Our Observations



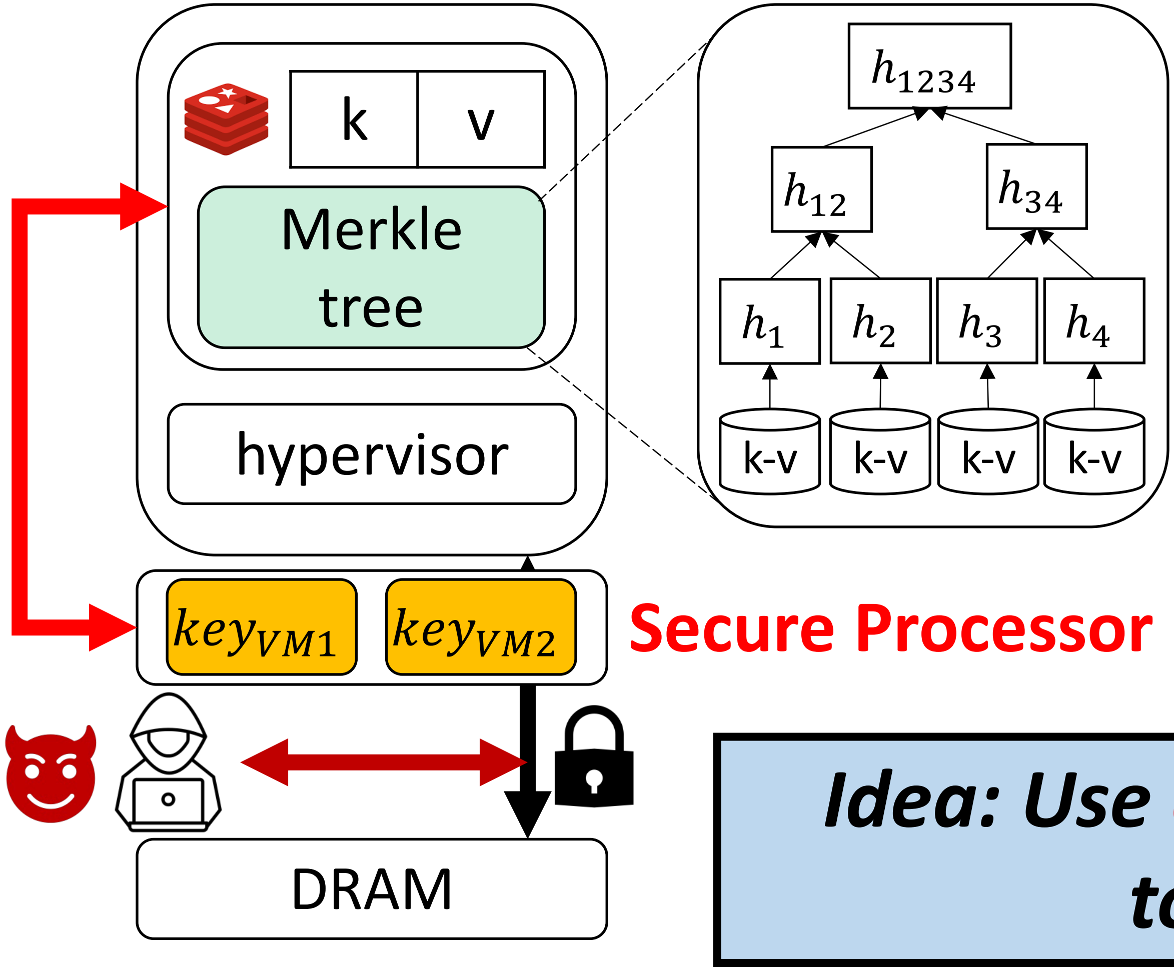
- **Observation 1**
  - If Merkle root is secure, modifying/ replaying internal nodes will be detected during root computation
- **Observation 2**
  - Different encryption keys are used for respective VMs

Our Observations



- **Observation 1**
  - If Merkle root is secure, modifying/ replaying internal nodes will be detected during root computation
- **Observation 2**
  - Different encryption keys are used for respective VMs
  - Secure channel btw. VMs and SP

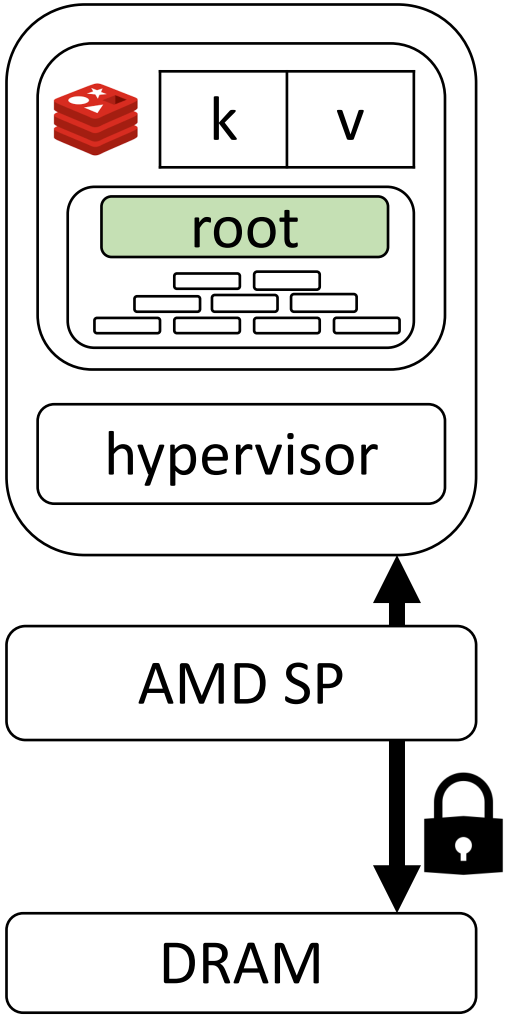
Our Observations



- **Observation 1**
  - If Merkle root is secure, modifying/ replaying internal nodes will be detected during root computation
- **Observation 2**
  - Different encryption keys are used for respective VMs
  - Secure channel btw. VMs and SP

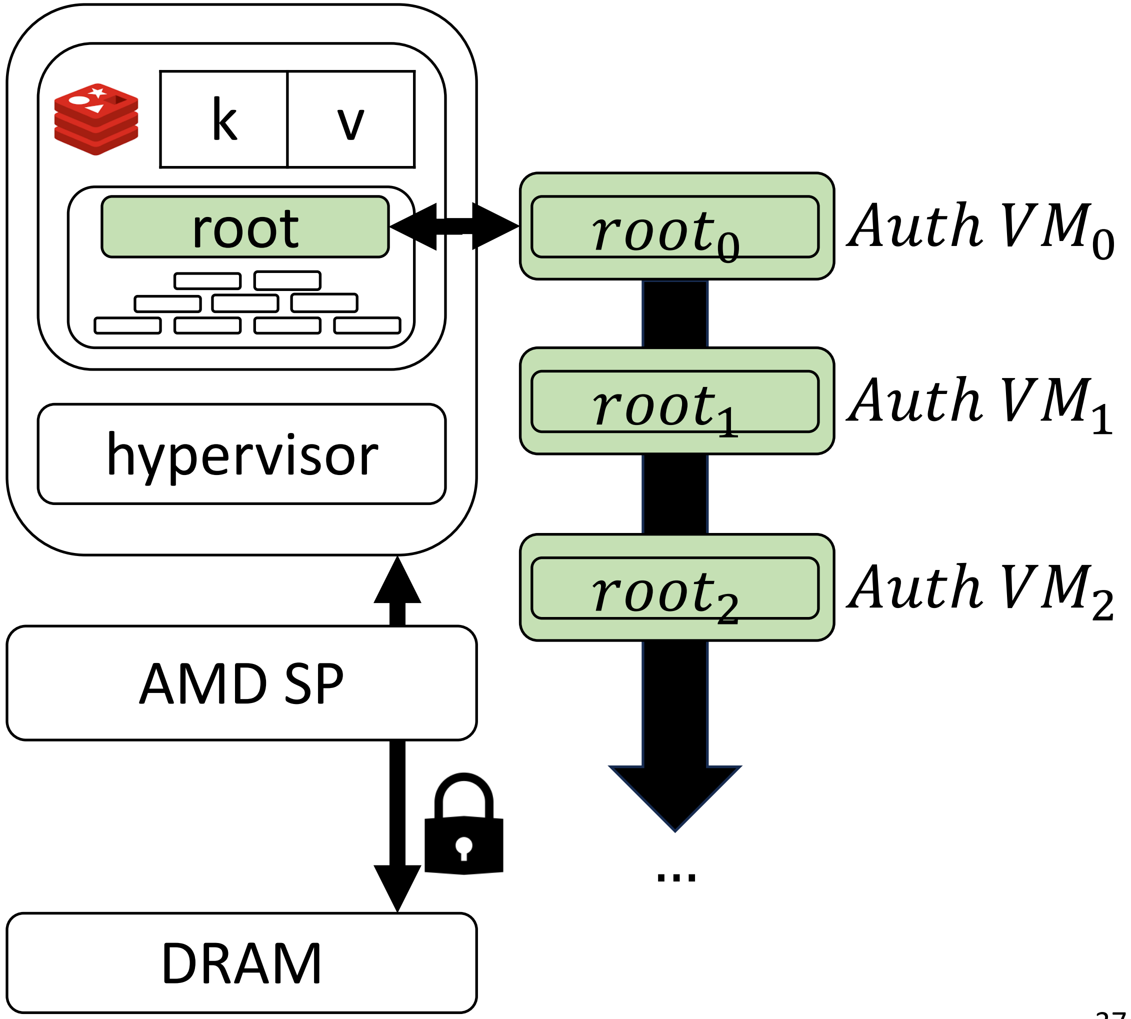
Idea: Use different VM encryption keys to secure Merkle root

Our Solution - KVSEV



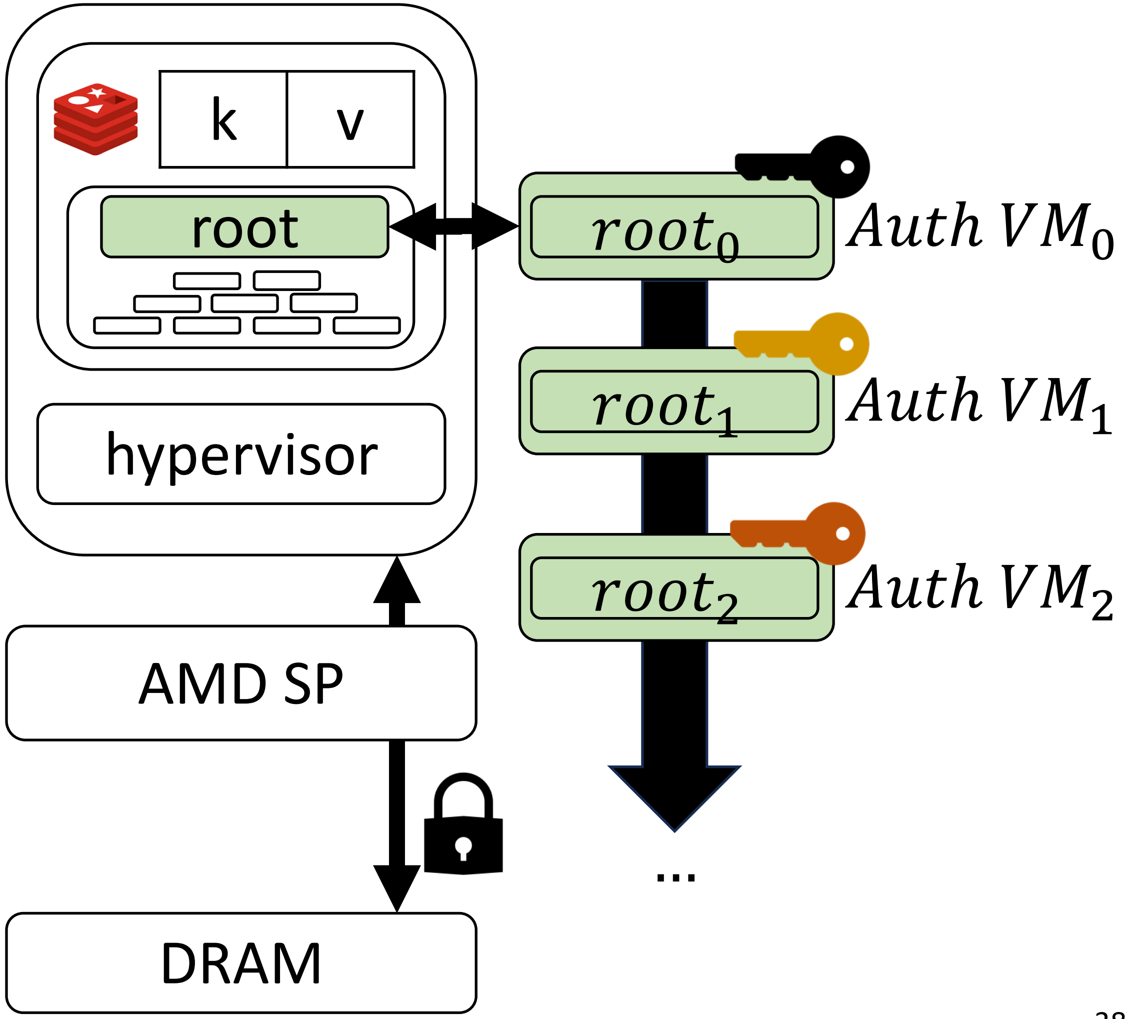
- Use **short-lived VMs** as **secure storage** for MT root

Our Solution - KVSEV



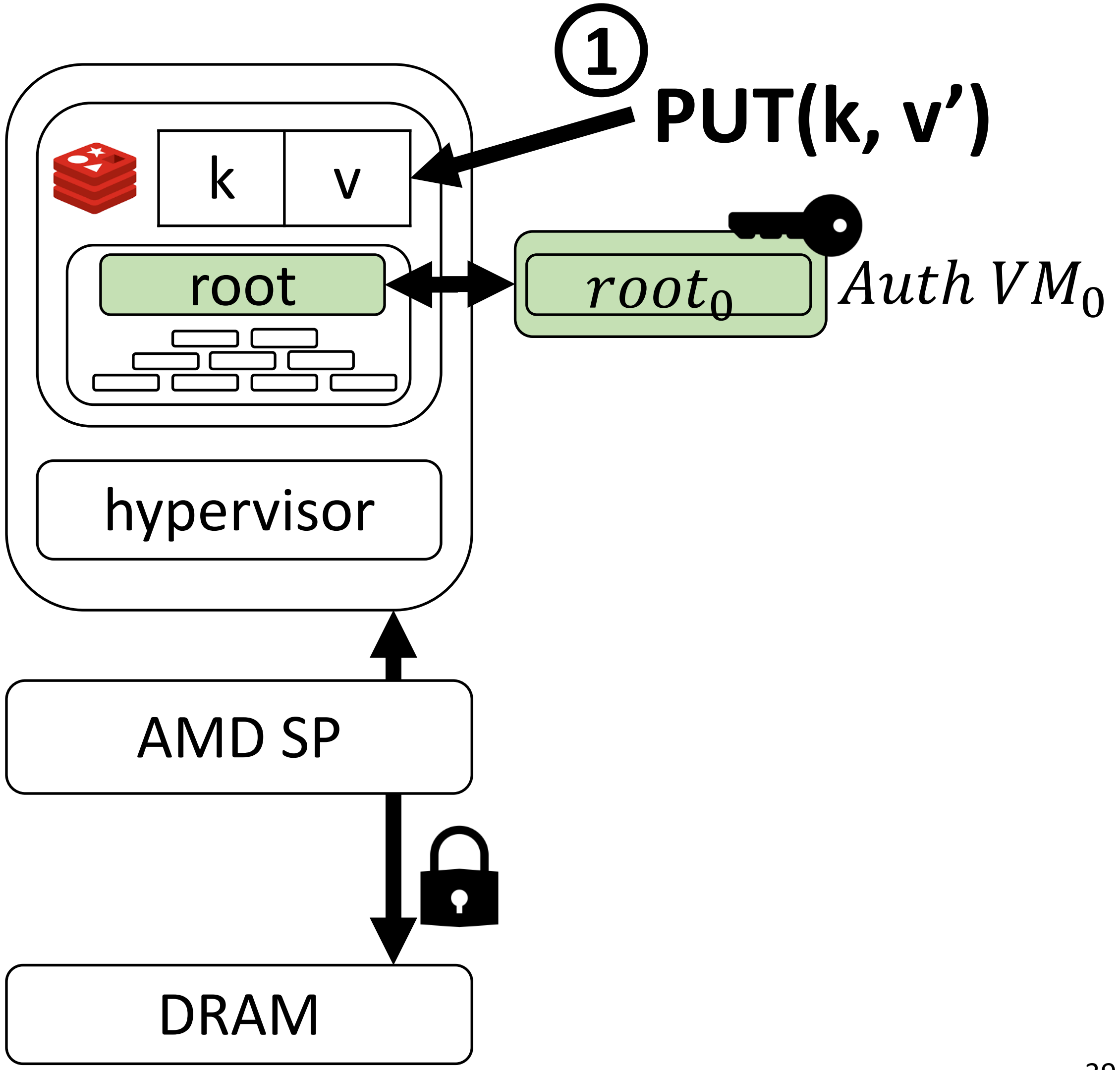
- **Use short-lived VMs as secure storage for MT root**

Our Solution - KVSEV



- **Use short-lived VMs as secure storage for MT root**
  - Different Auth-VM to store MT root on every root update
  - Different encryption keys

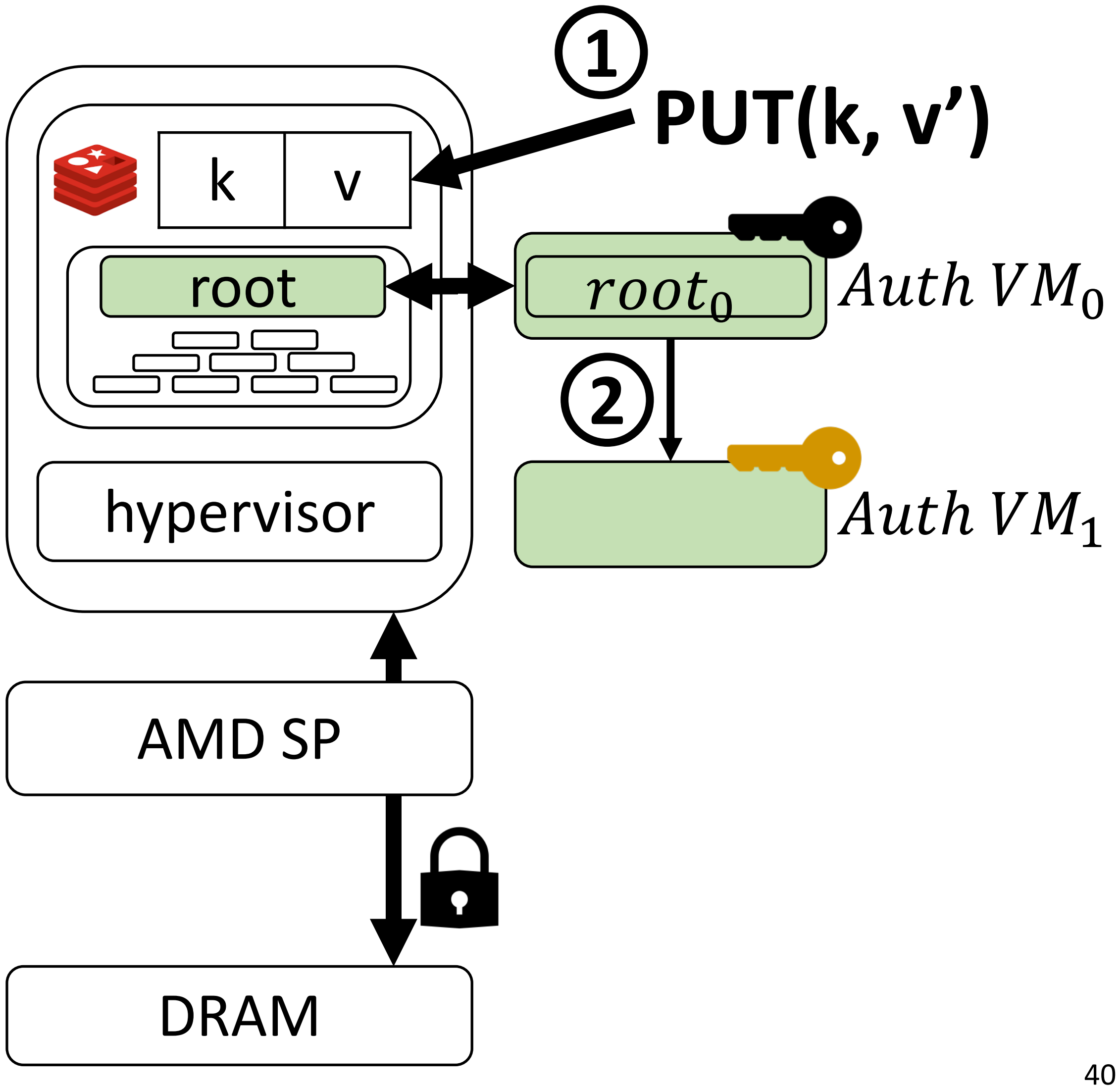
Our Solution - KVSEV



- **Use short-lived VMs as secure storage for MT root**
  - Different Auth-VM to store MT root on every root update
  - Different encryption keys

- **PUT**
  - Calculate $root_{new}$
  - Store $root_{new}$ in new Auth VM

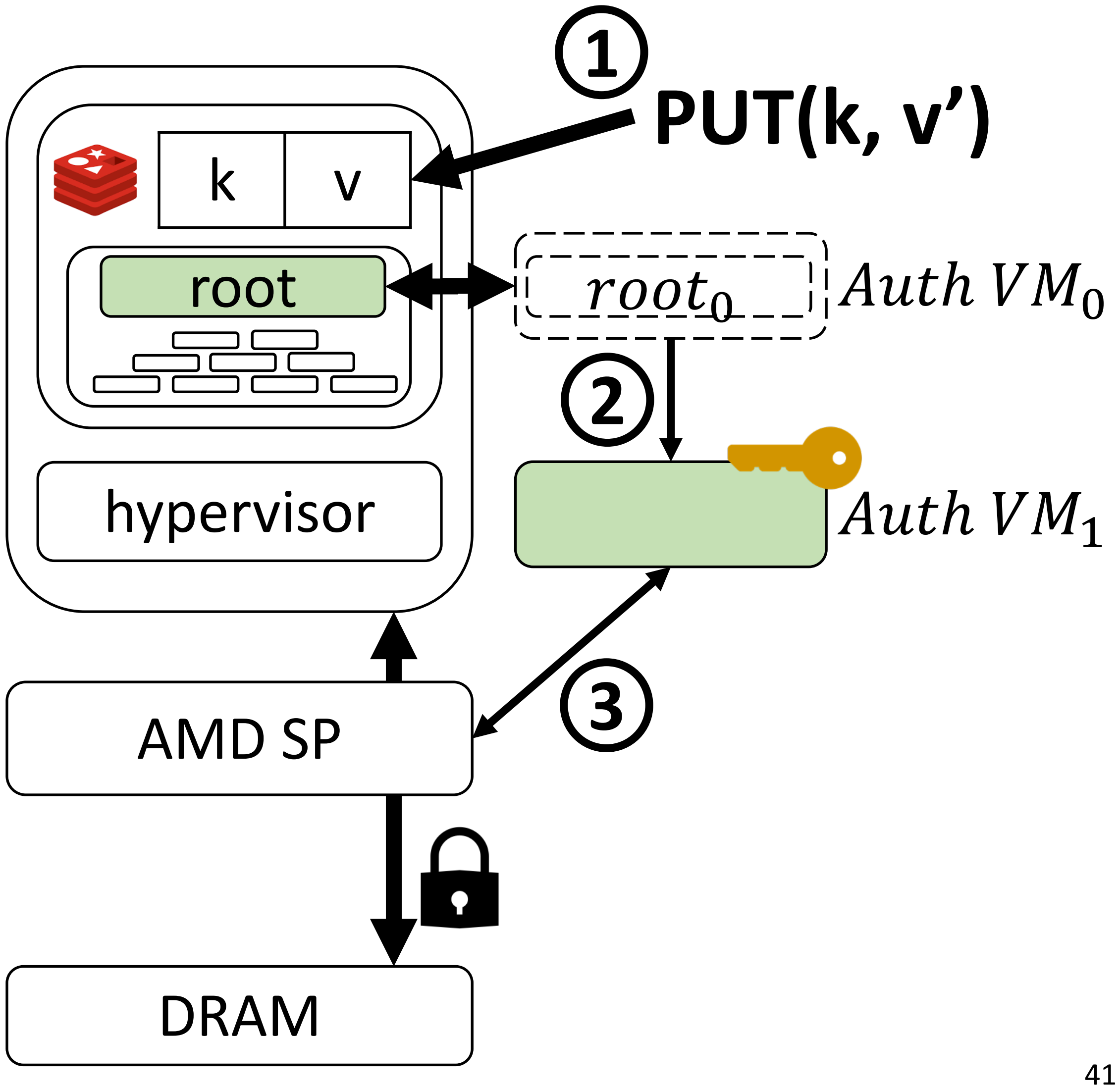
Our Solution - KVSEV



- **Use short-lived VMs as secure storage for MT root**
  - Different Auth-VM to store MT root on every root update
  - Different encryption keys

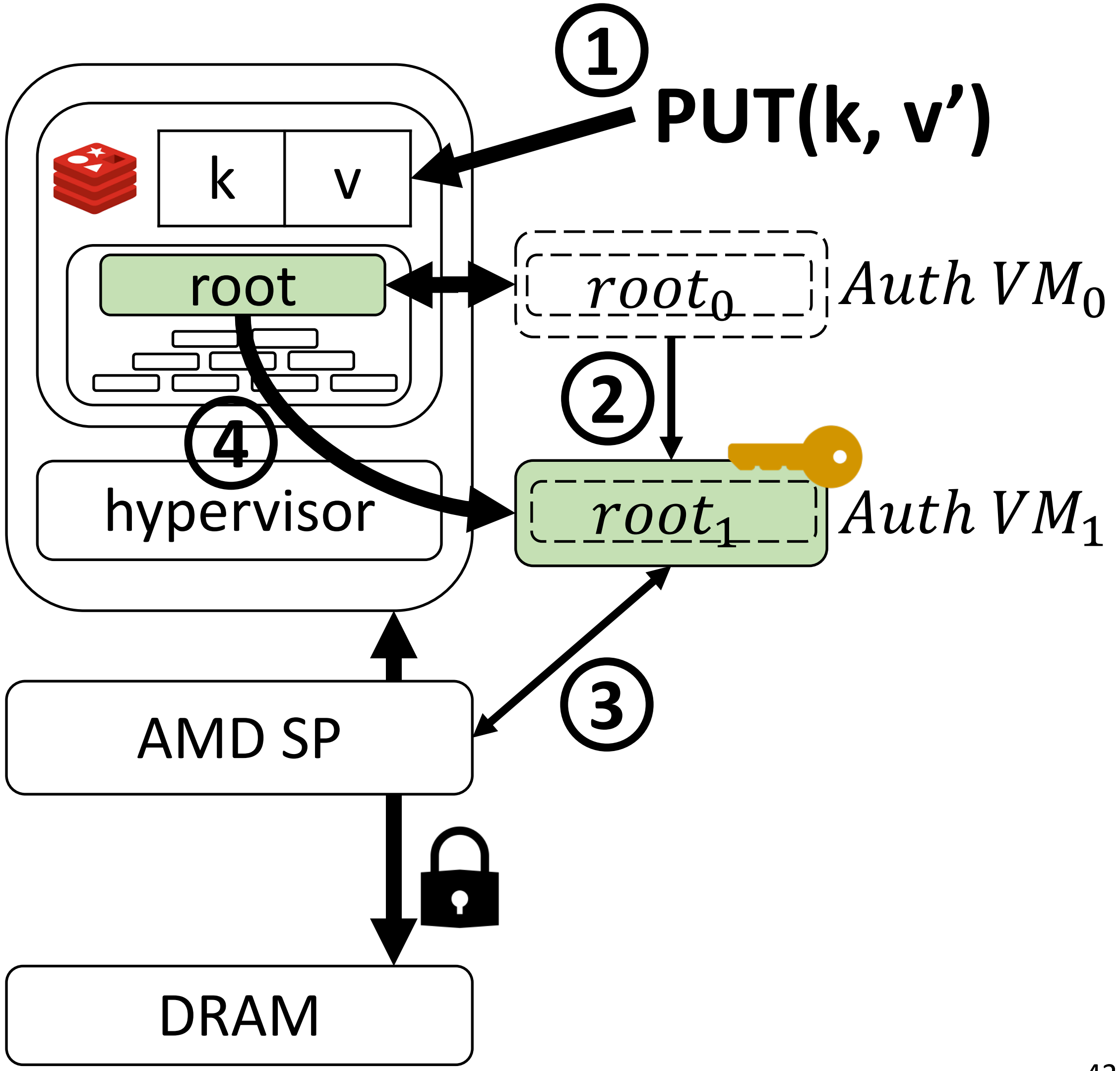
- **PUT**
  - Calculate $root_{new}$
  - Store $root_{new}$ in new Auth VM

Our Solution - KVSEV



- **Use short-lived VMs as secure storage for MT root**
  - Different Auth-VM to store MT root on every root update
  - Different encryption keys
- **PUT**
  - Calculate $root_{new}$
  - Store $root_{new}$ in new Auth VM

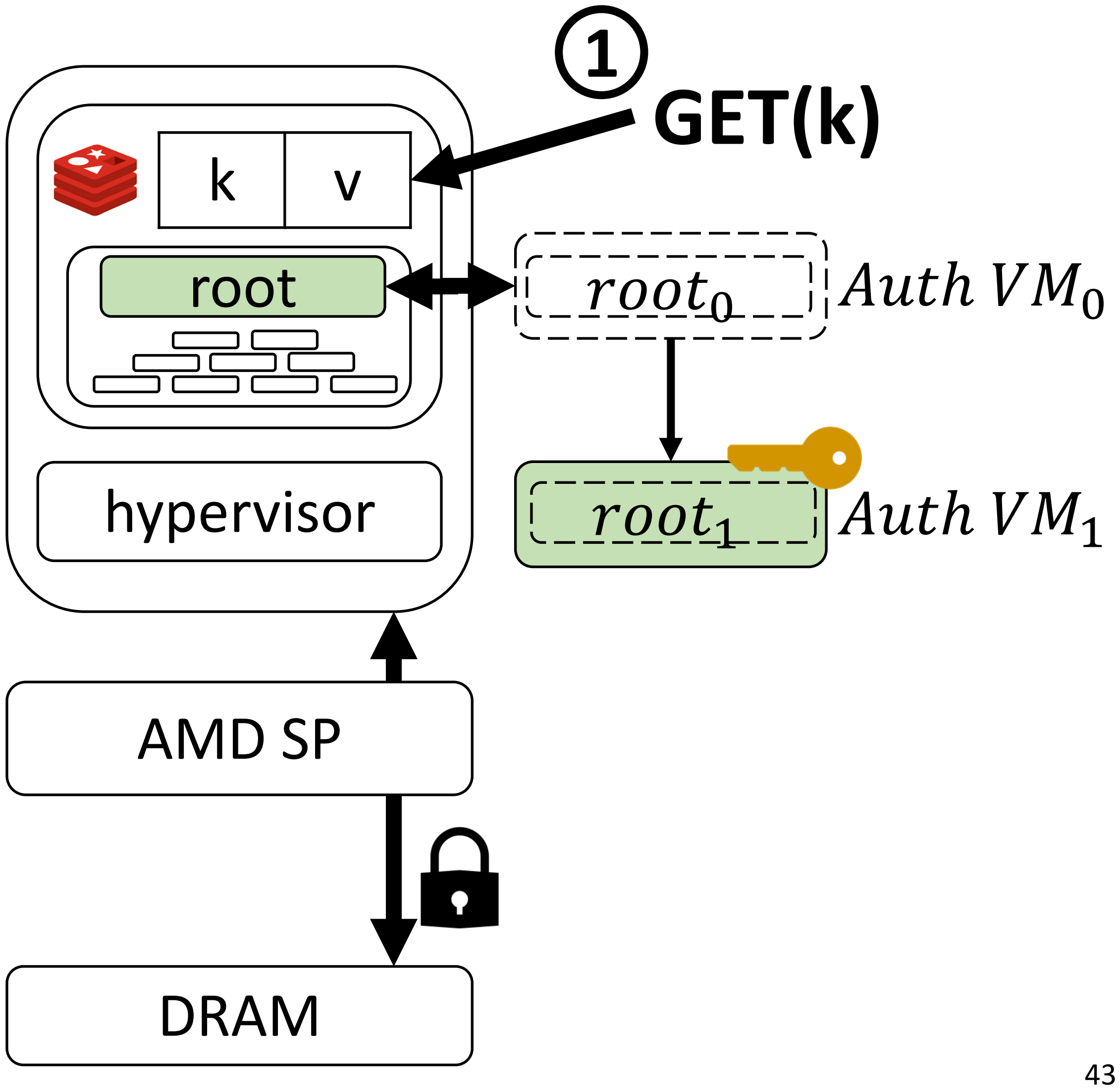
Our Solution - KVSEV



- **Use short-lived VMs as secure storage for MT root**
  - Different Auth-VM to store MT root on every root update
  - Different encryption keys

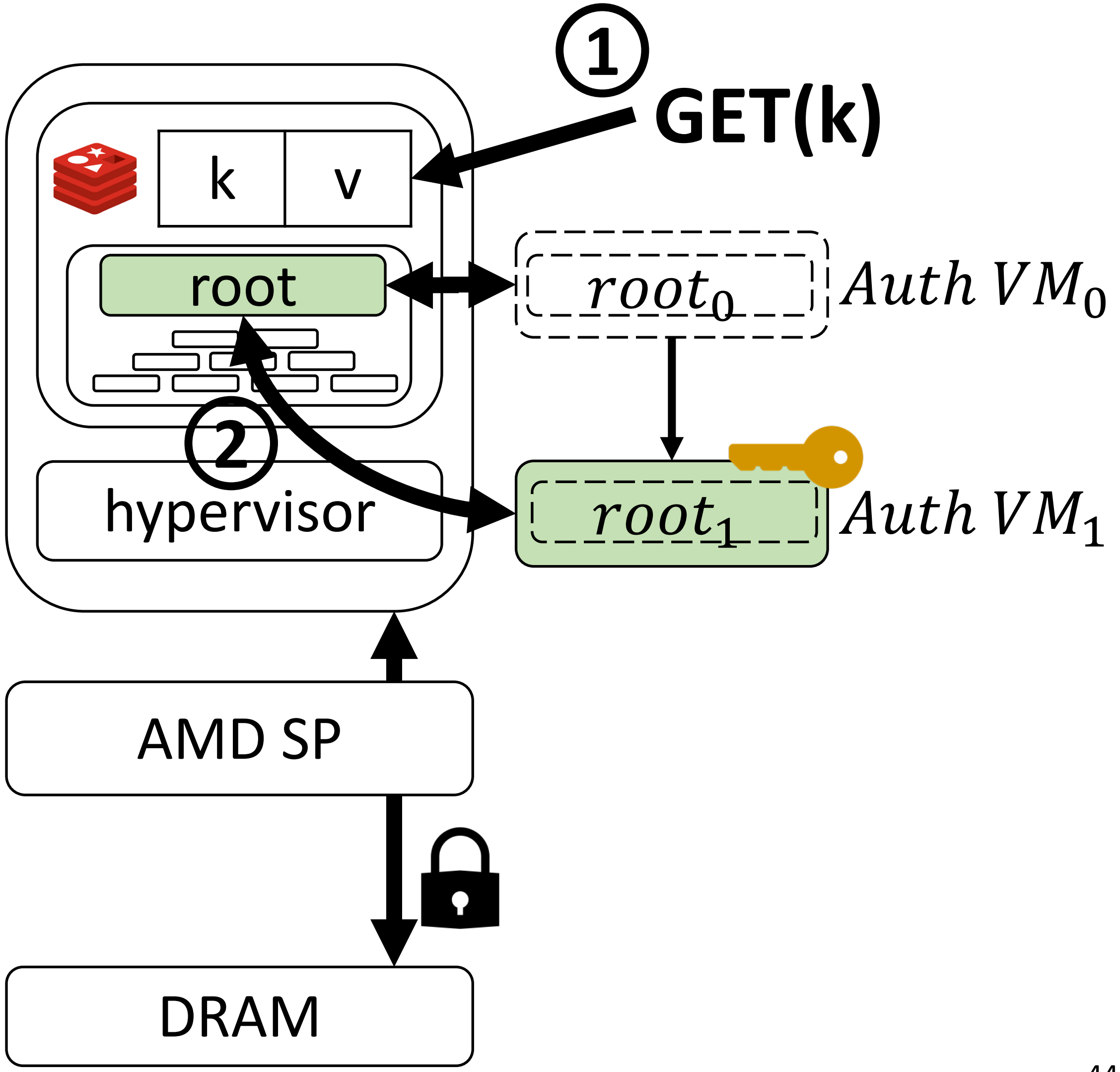
- **PUT**
  - Calculate $root_{new}$
  - Store $root_{new}$ in new Auth VM

Our Solution - KVSEV



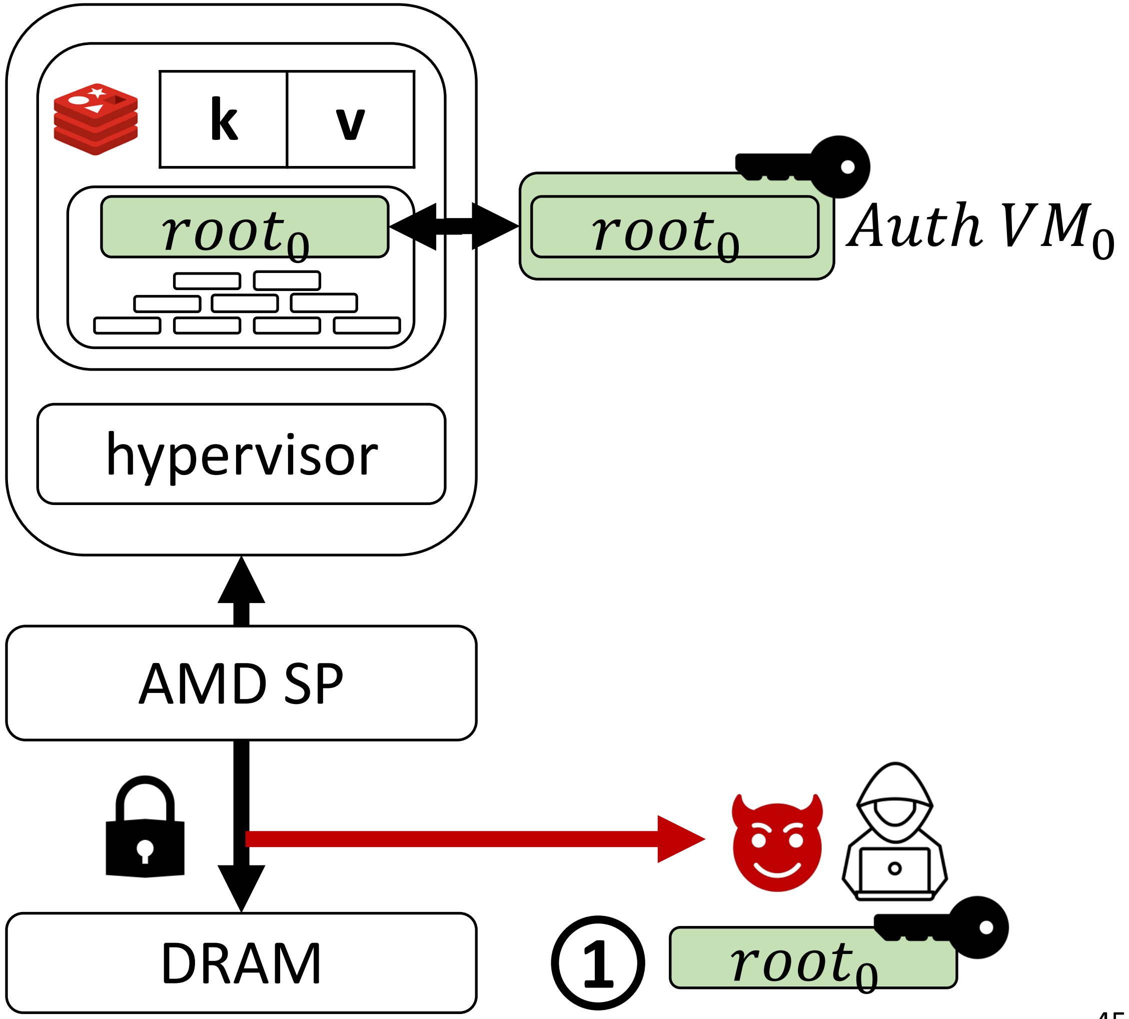
- **Use short-lived VMs as secure storage for MT root**
  - Different Auth-VM to store MT root on every root update
  - Different encryption keys
- **PUT**
  - Calculate $root_{new}$
  - Store $root_{new}$ in new Auth VM
- **GET** – verify MT root

Our Solution - KVSEV

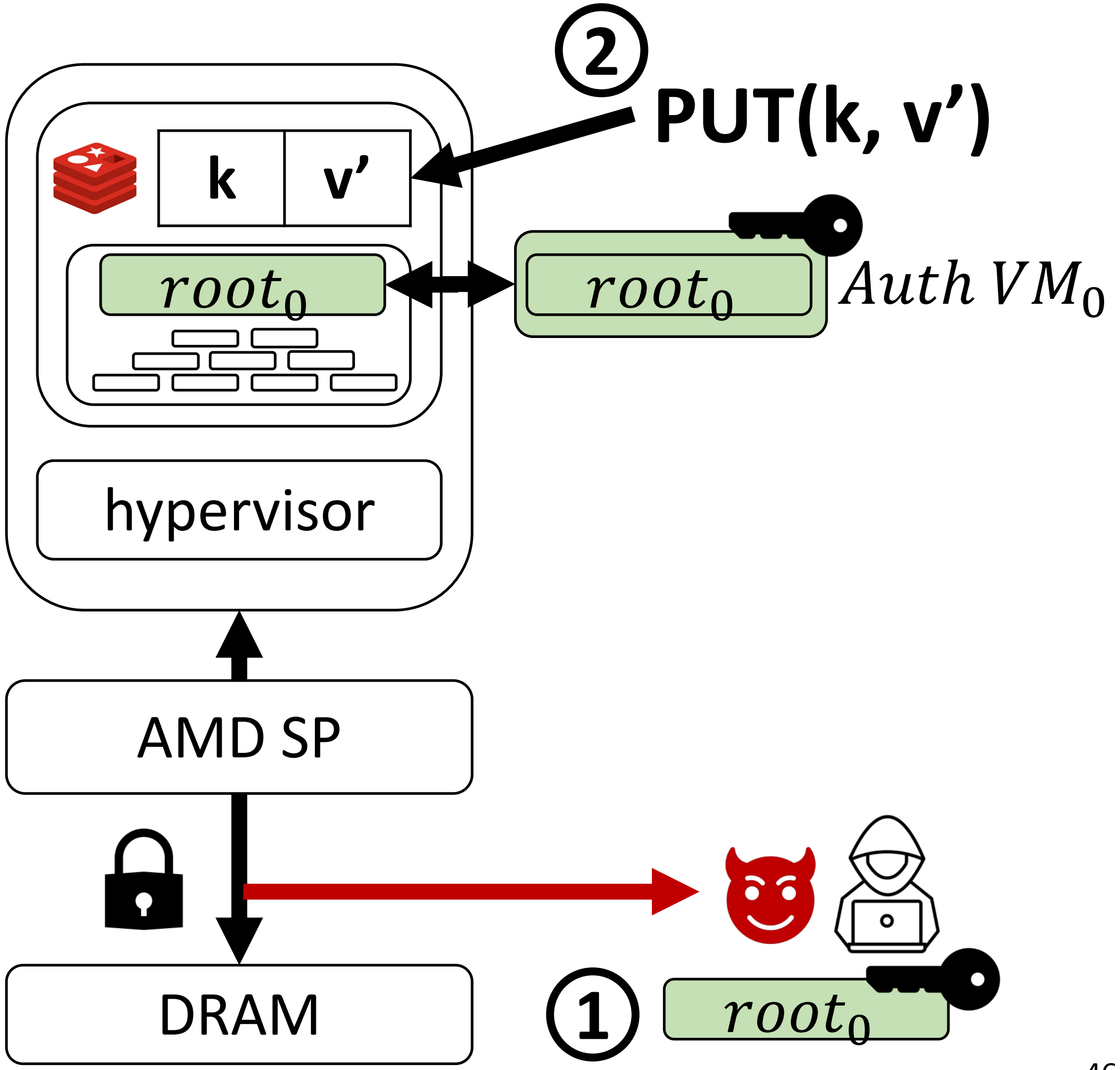


- **Use short-lived VMs as secure storage for MT root**
  - Different Auth-VM to store MT root on every root update
  - Different encryption keys
- **PUT**
  - Calculate $root_{new}$
  - Store $root_{new}$ in new Auth VM
- **GET** – verify MT root

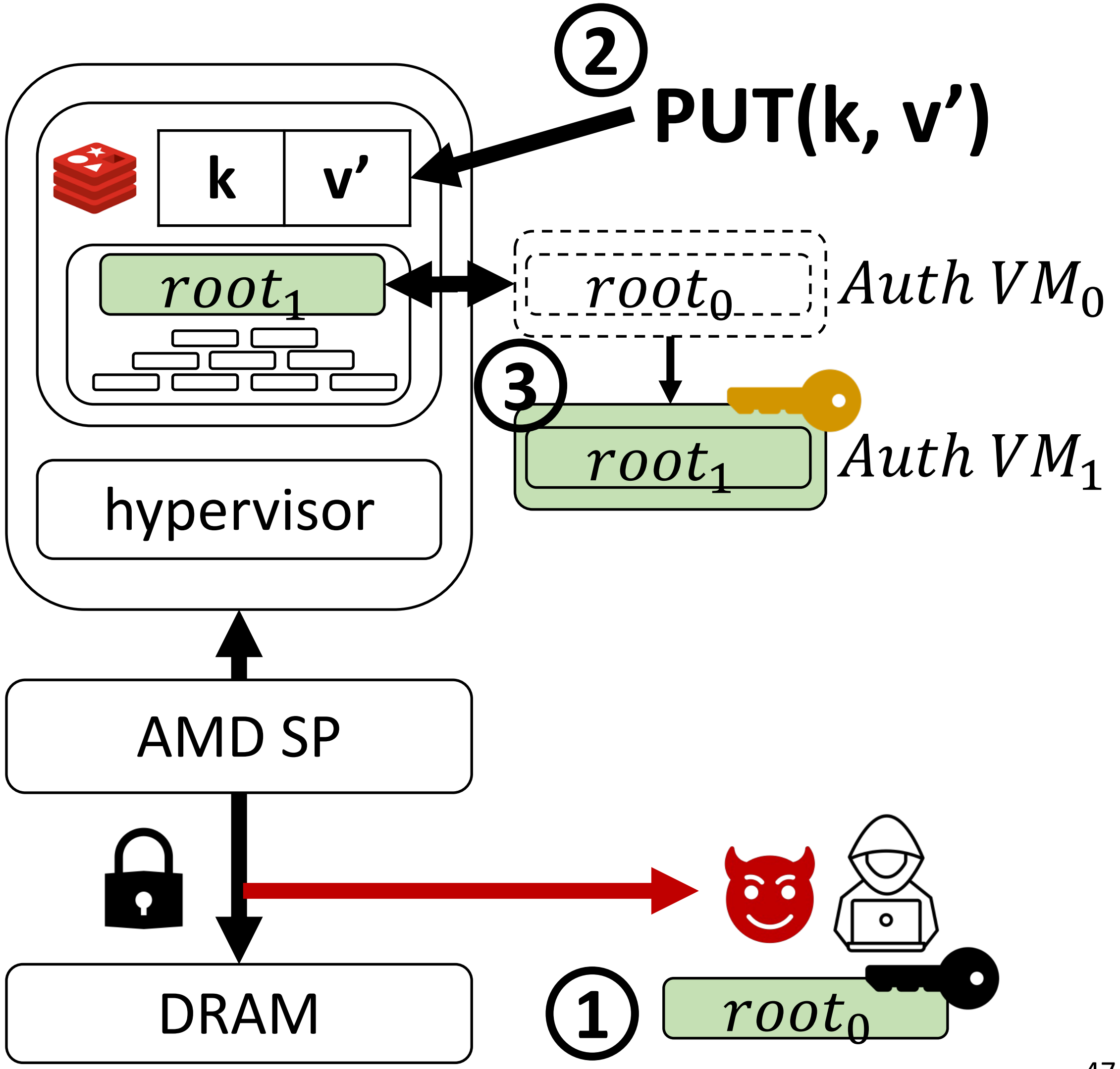
Our Solution - KVSEV



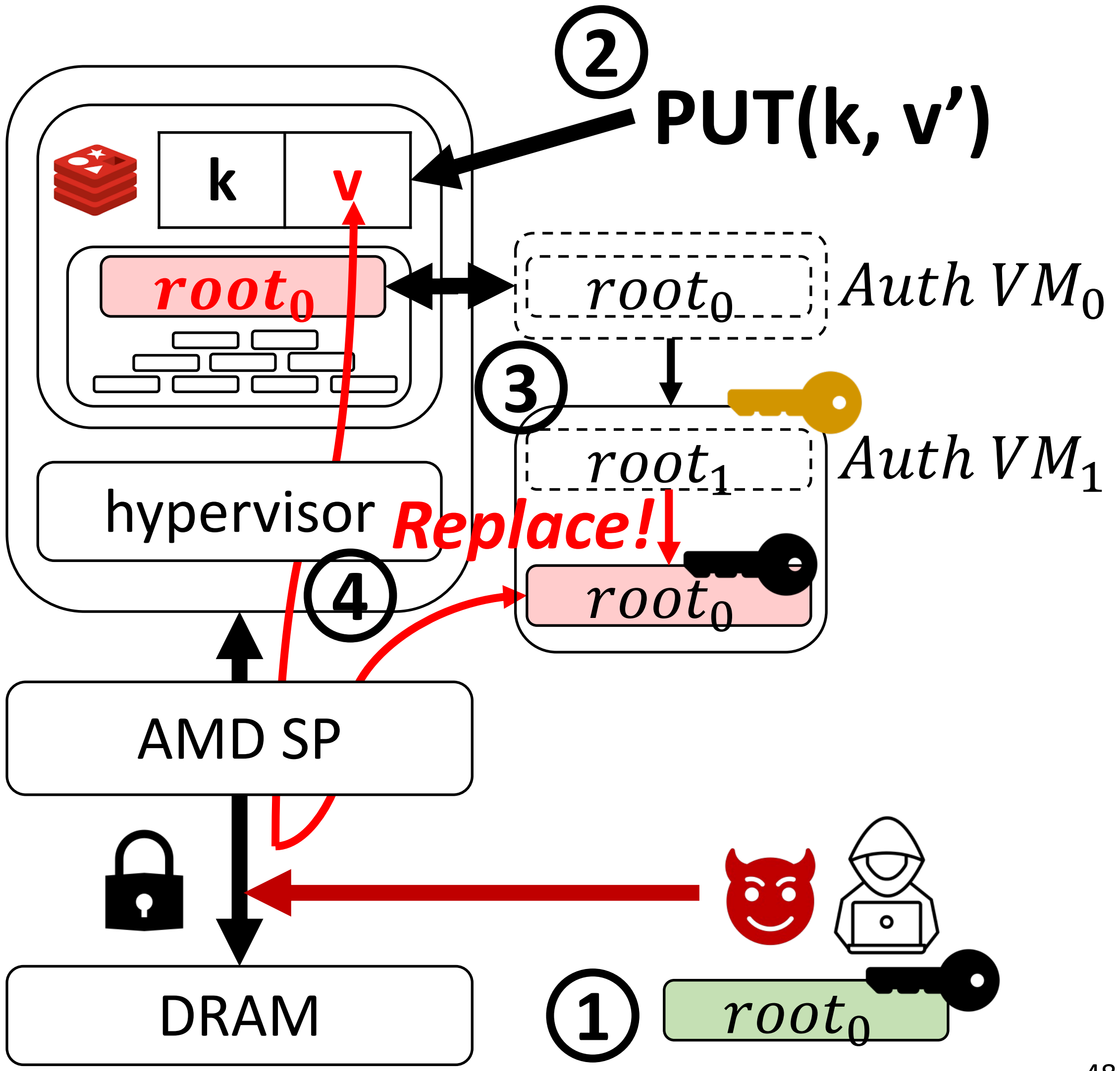
Our Solution - KVSEV



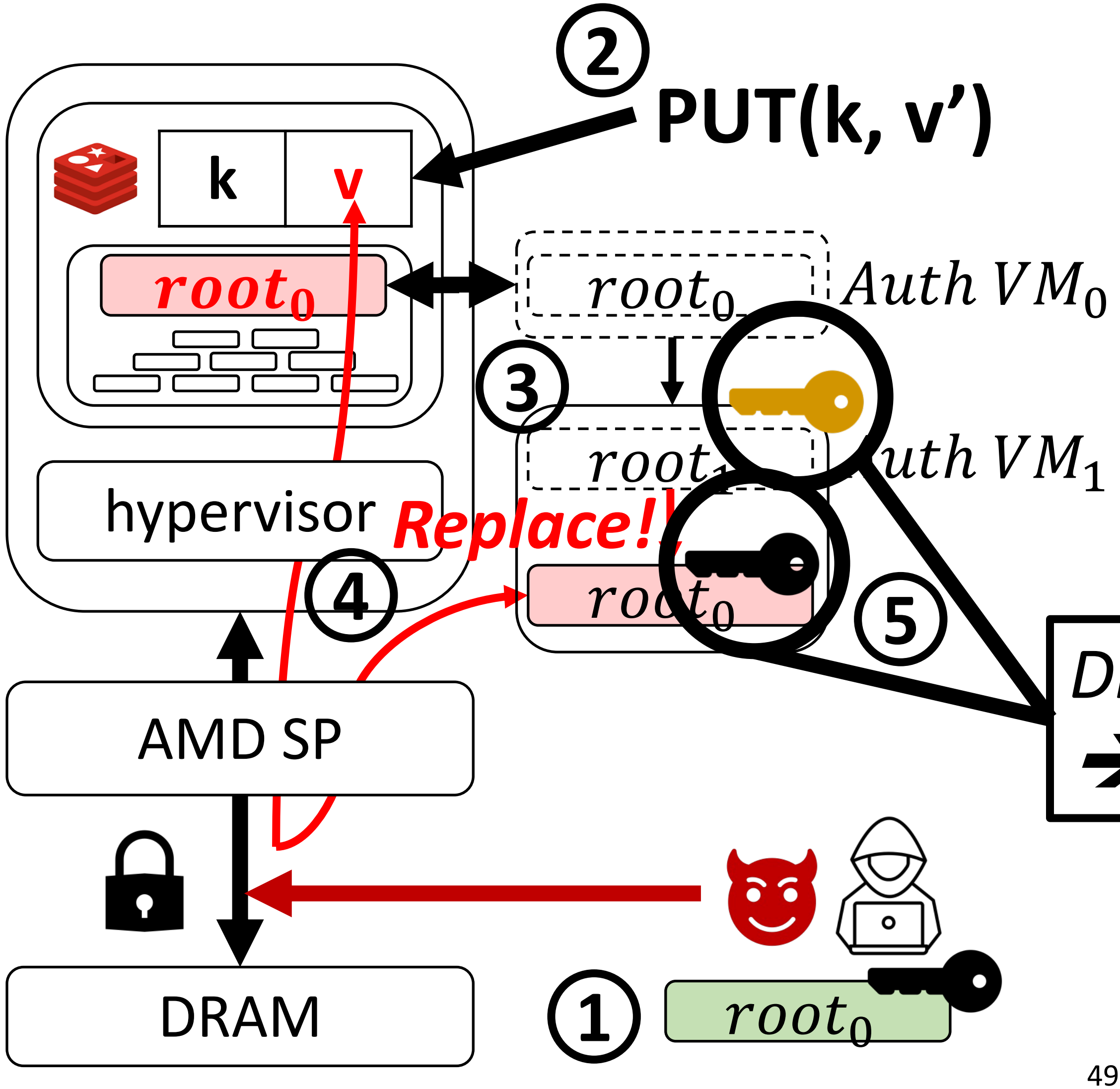
Our Solution - KVSEV



Our Solution - KVSEV

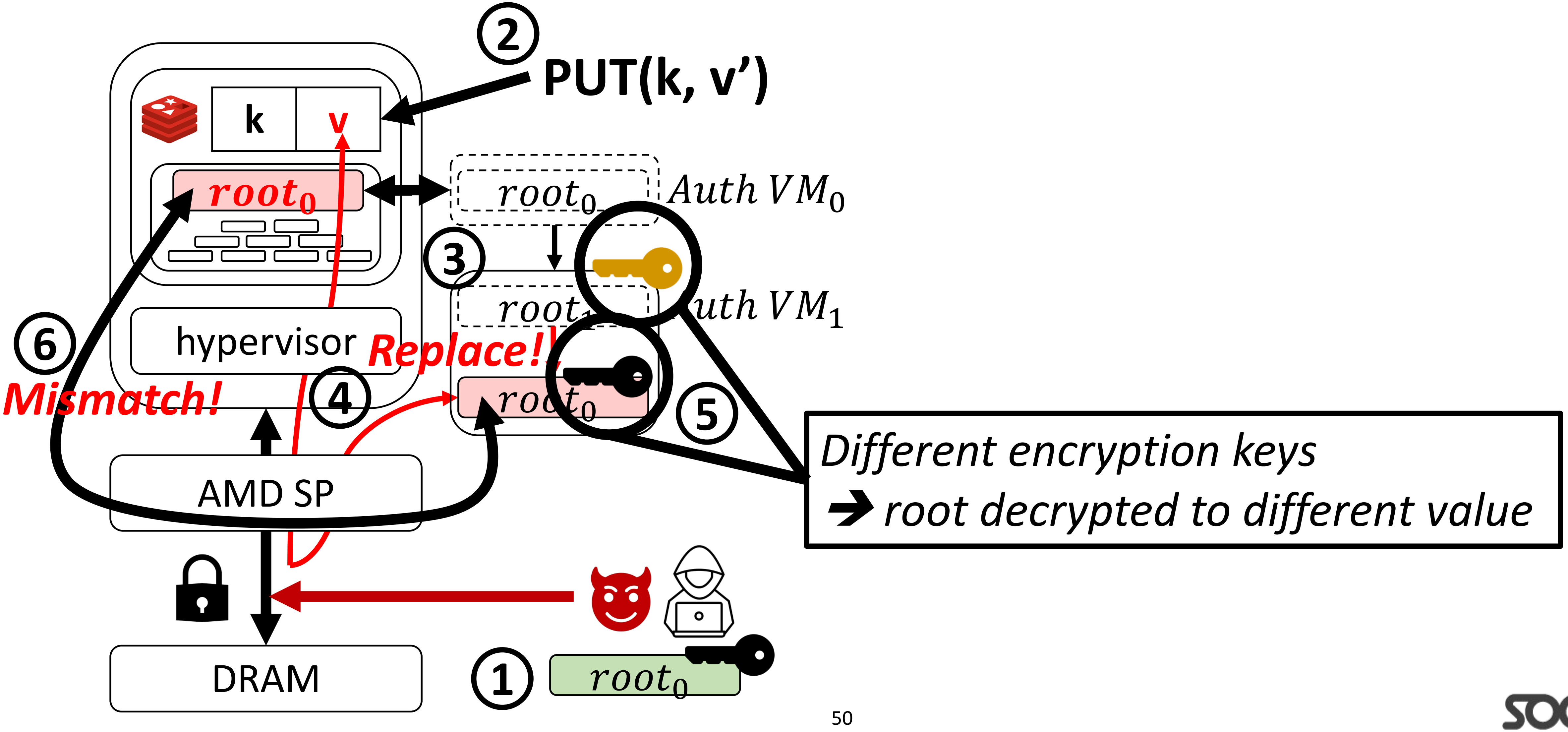


Our Solution - KVSEV



Different encryption keys
➔ root decrypted to different value

Our Solution - KVSEV



Different encryption keys

➔ *root decrypted to different value*

Additional Protection & Optimizations

- **Additional Protections**
  - (Issue 1) Integrity of new key-value pair(s)

Additional Protection & Optimizations

- **Additional Protections**
  - (Issue 1) Integrity of new key-value pair(s)
    ➔ Solution: verify before/after writing to database

Additional Protection & Optimizations

- **Additional Protections**
  - (Issue 1) Integrity of new key-value pair(s)
    ➔ Solution: verify before/after writing to database
  - (Issue 2) Protecting new Merkle root

Additional Protection & Optimizations

- **Additional Protections**
  - (Issue 1) Integrity of new key-value pair(s)
    ➔ Solution: verify before/after writing to database
  - (Issue 2) Protecting new Merkle root
    ➔ Solution: obscuring physical location of calculated Merkle root

Additional Protection & Optimizations

- **Additional Protections**
  - (Issue 1) Integrity of new key-value pair(s)
    ➔ Solution: verify before/after writing to database
  - (Issue 2) Protecting new Merkle root
    ➔ Solution: obscuring physical location of calculated Merkle root
  - (Issue 3) Protecting Merkle root calculation

Additional Protection & Optimizations

- **Additional Protections**
  - (Issue 1) Integrity of new key-value pair(s)
    ➔ Solution: verify before/after writing to database
  - (Issue 2) Protecting new Merkle root
    ➔ Solution: obscuring physical location of calculated Merkle root
  - (Issue 3) Protecting Merkle root calculation
    ➔ Solution: reserved register to hold intermediate nodes

Additional Protection & Optimizations

- **Additional Protections**
  - (Issue 1) Integrity of new key-value pair(s)
    ➔ Solution: verify before/after writing to database
  - (Issue 2) Protecting new Merkle root
    ➔ Solution: obscuring physical location of calculated Merkle root
  - (Issue 3) Protecting Merkle root calculation
    ➔ Solution: reserved register to hold intermediate nodes
- **Optimizations**
  - Eager Auth-VM creation
  - VM debloating
  - Asynchronous verification

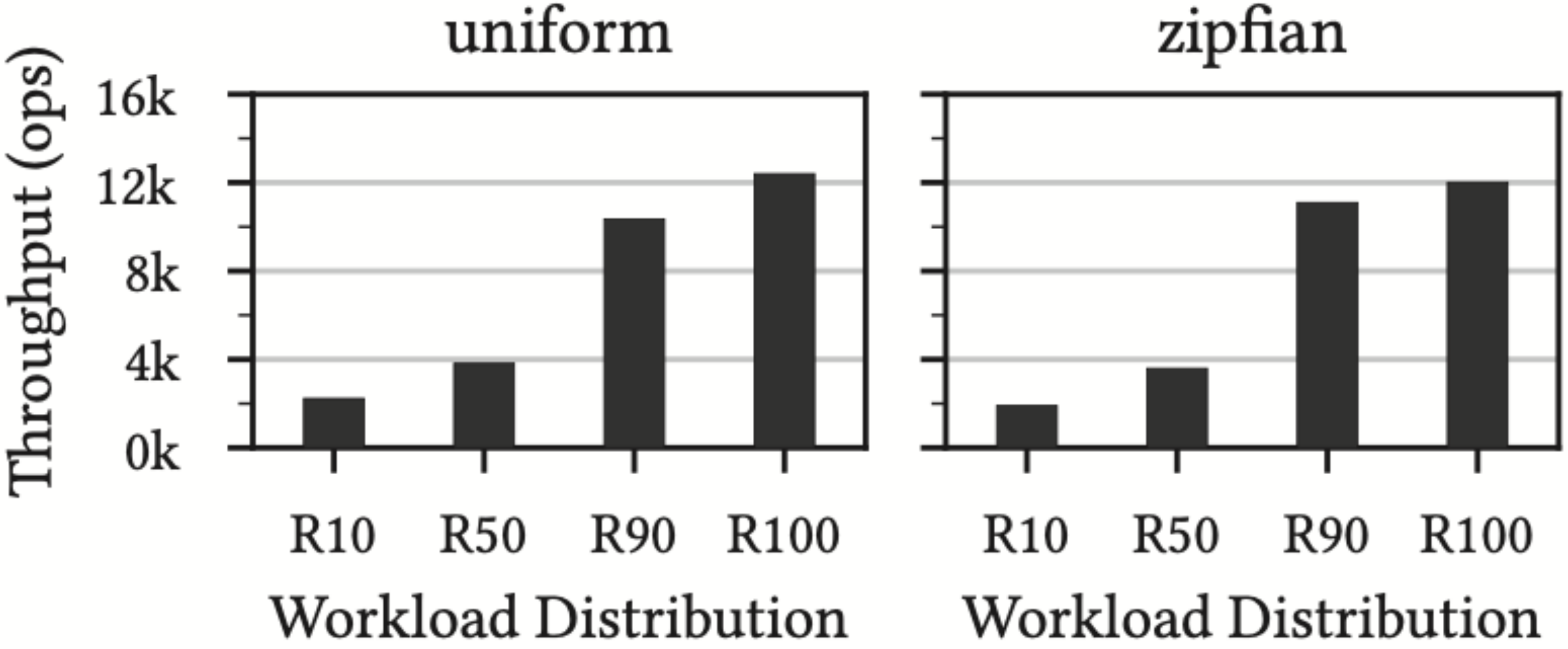
Additional Protection & Optimizations

- **Additional Protections**
  - (Issue 1) Integrity of new key-value pair(s)
    ➔ Solution: verify before/after writing to database
  - (Issue 2) Protecting new Merkle root
    ➔ Solution: obscuring physical location of calculated Merkle root
  - (Issue 3) Protecting Merkle root calculation
    ➔ Solution: reserved register to hold intermediate nodes
- **Optimizations**
  - Eager Auth-VM creation
  - VM debloating
  - Asynchronous verification

Please refer to the paper for details

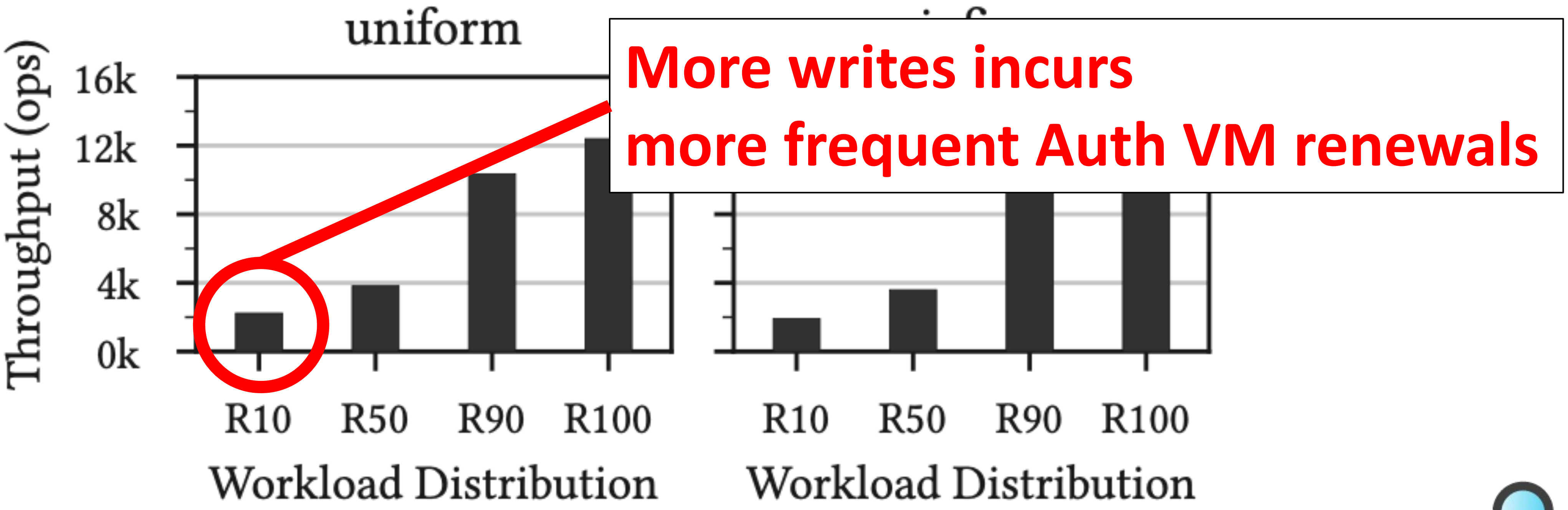
Standalone Evaluation

- **KVSEV performs**
  - (Baseline) **13.38x – 64.23x** slower than native KVS
  - Similar numbers across varying number of threads, value size, KVS size



Standalone Evaluation

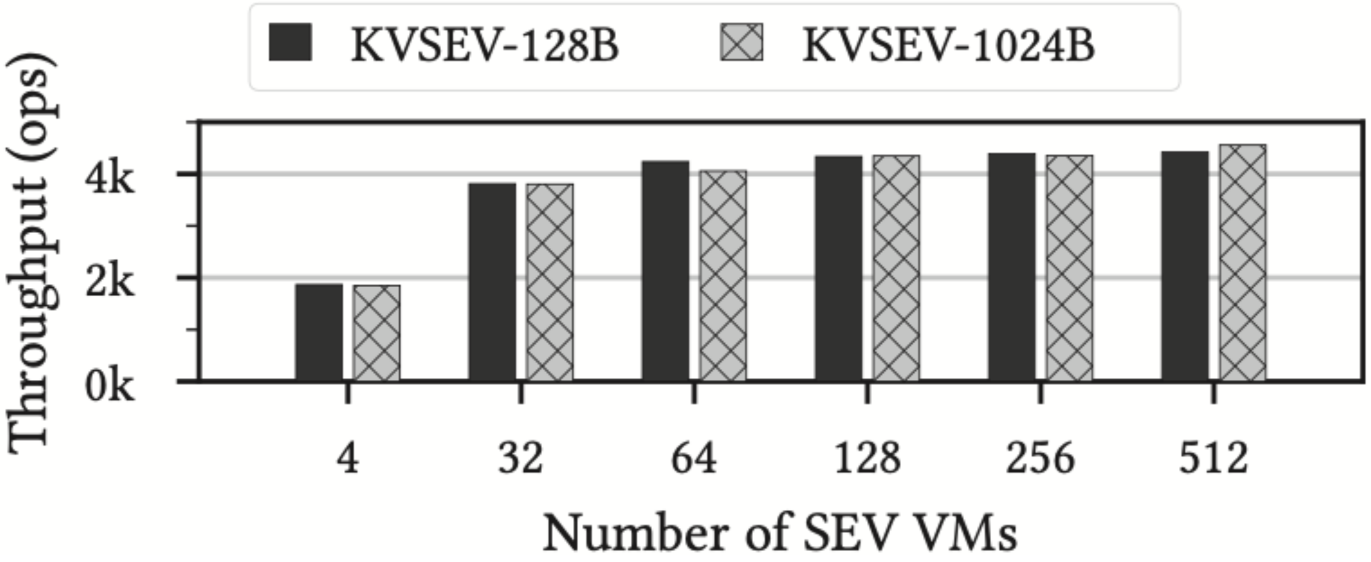
- **KVSEV performs**
  - (Baseline) **13.38x – 64.23x** slower than native KVS
  - Similar numbers across varying number of threads, value size, KVS size

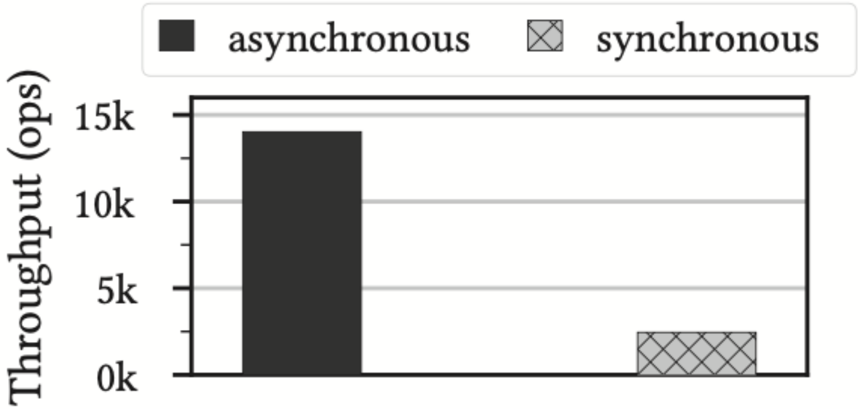


More writes incurs
more frequent Auth VM renewals

Impact of Optimizations

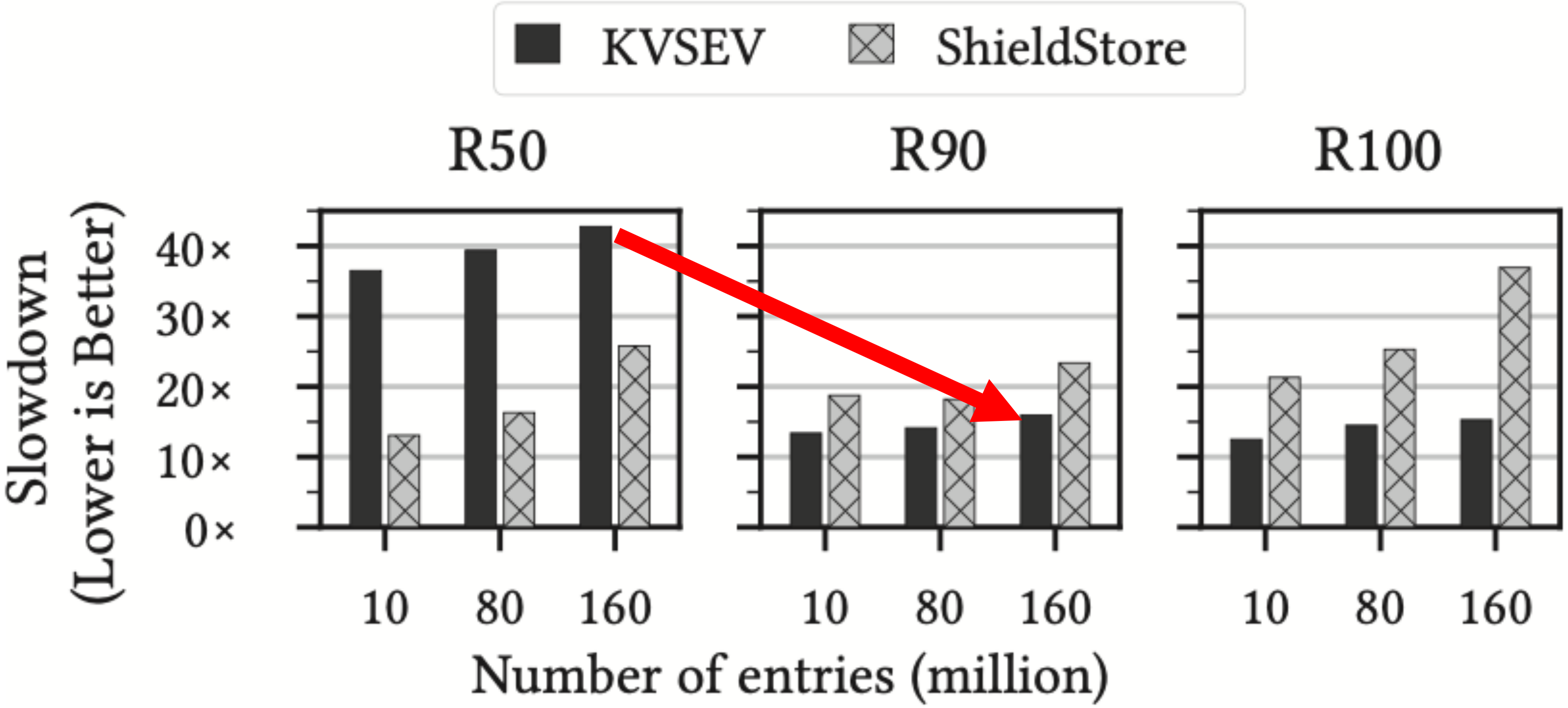
- **KVSEV improves performance by**
  - **2.3x** with *eager VM creation*
  - **5.7x** with *asynchronous verification*
  - **14x** with *Auth VM debloating* (151.1 VMs/s ➔ 2156.3 VMs/s)





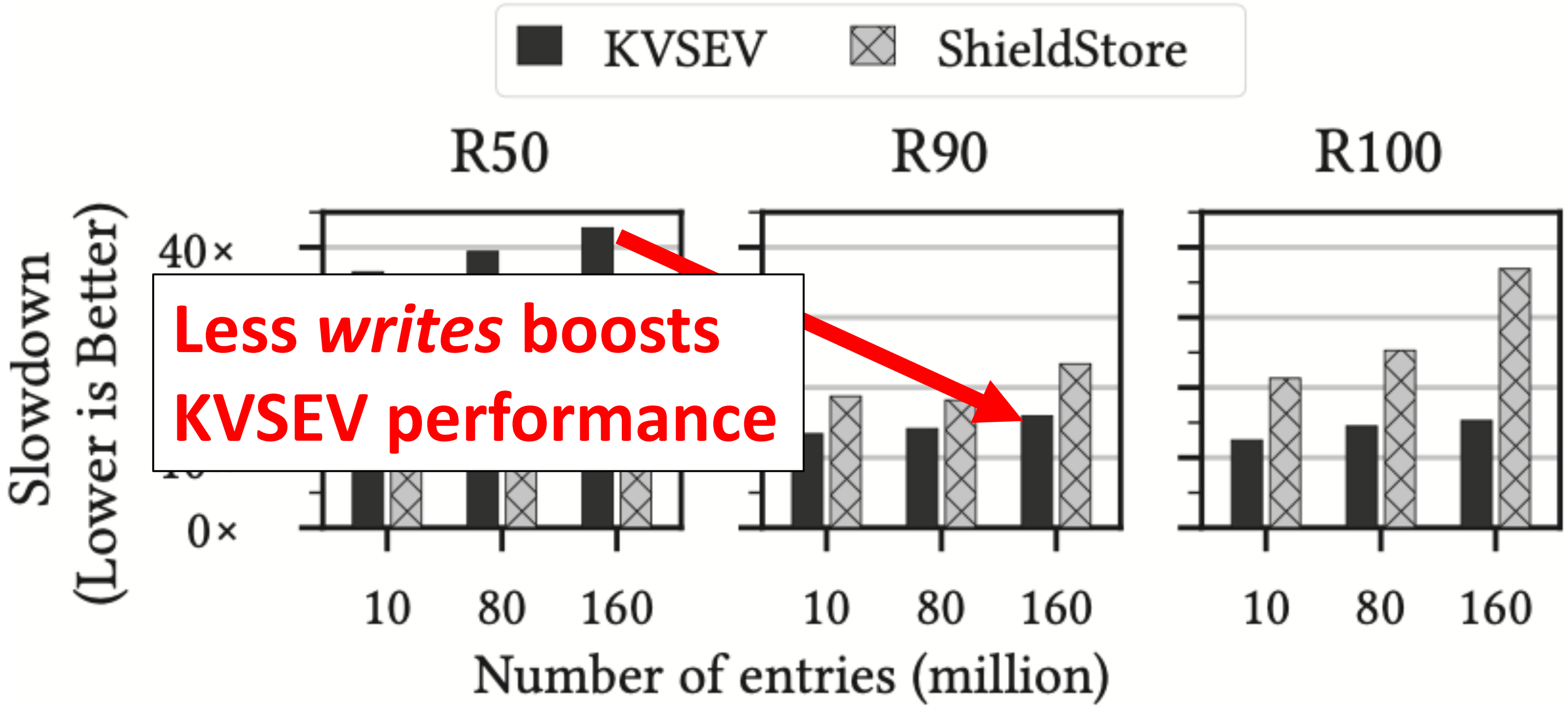
Comparison to SGX-based Secure KVS

- **KVSEV performs better than ShieldStore by**
  - **1.47x** when accommodating large number of key-value pairs



Comparison to SGX-based Secure KVS

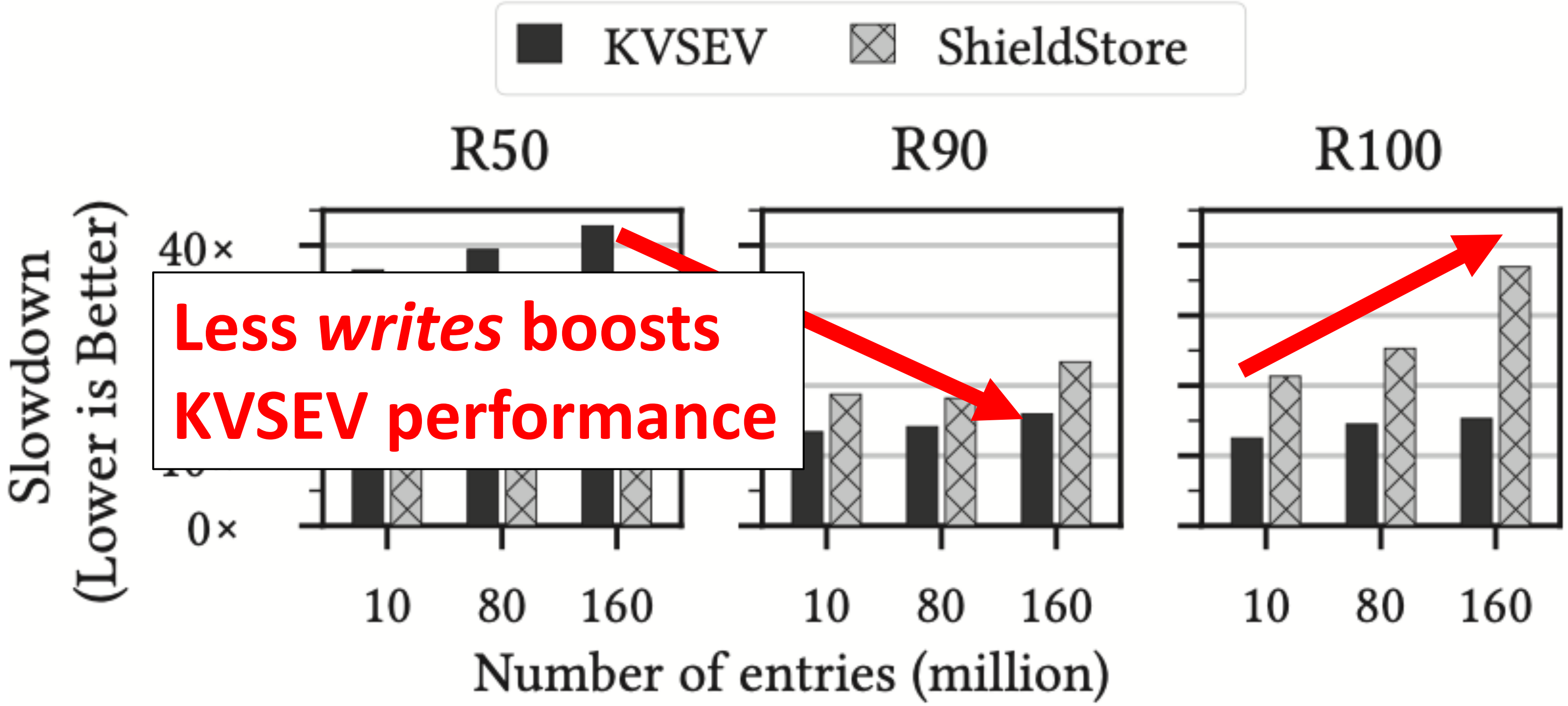
- **KVSEV performs better than ShieldStore by**
  - **1.47x** when accommodating large number of key-value pairs



Less *writes* boosts KVSEV performance

Comparison to SGX-based Secure KVS

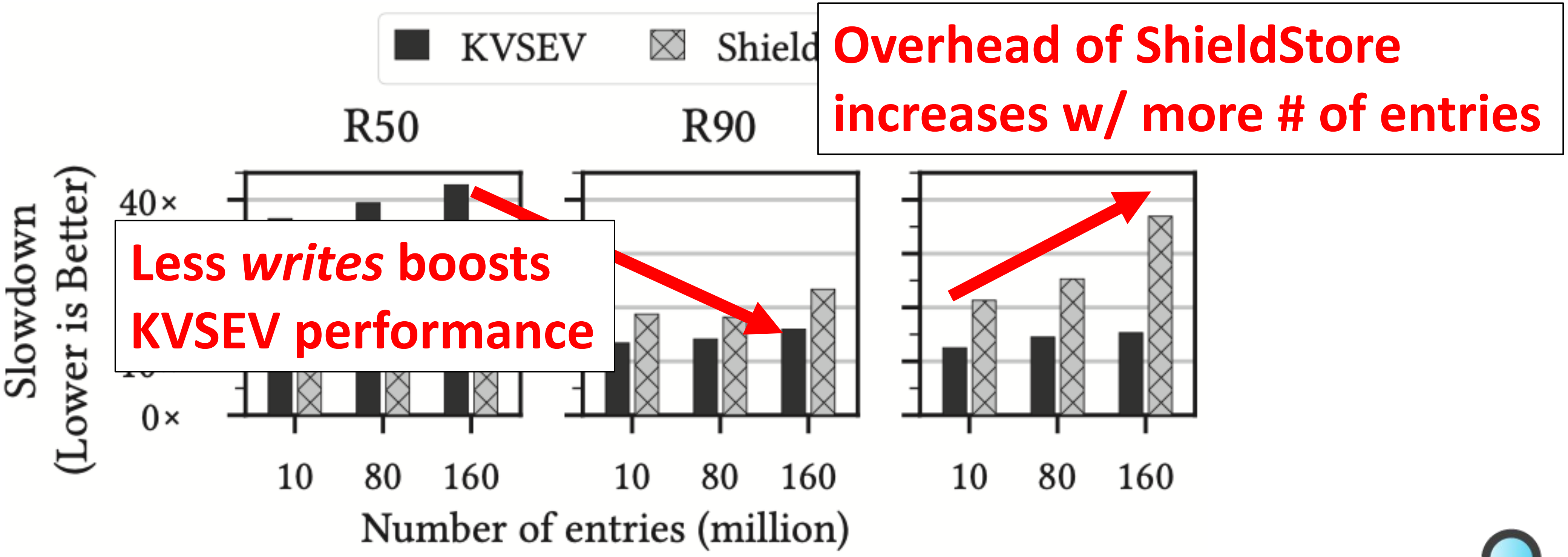
- **KVSEV performs better than ShieldStore by**
  - **1.47x** when accommodating large number of key-value pairs



Less *writes* boosts KVSEV performance

Comparison to SGX-based Secure KVS

- **KVSEV performs better than ShieldStore by**
  - **1.47x** when accommodating large number of key-value pairs



Overhead of ShieldStore increases w/ more # of entries

Less *writes* boosts KVSEV performance

Summary

- KVSEV is a **secure** in-memory KVS with AMD **SEV**
- KVSEV protects KVS from **physical adversaries** by using **ephemeral VMs** as safe storage for SW-only Merkle tree roots

KVSEV:
A Secure In-Memory Key-Value Store with Secure Encrypted Virtualization

Junseung You, Kyeongryong Lee, Hyungon Moon,
Yeongpil Cho, Yunheung Paek

Santa Cruz, USA
October 31-November 1, 2023