Toward Scaling Hardware Security Module for Emerging Cloud Services

Juhyeng Han*, Seongmin Kim*, Taesoo Kim †, Dongsu Han

KAIST †Georgia Tech

* The first two authors contributed equally to this work.
Hardware Security Modules (HSMs)

- **Root of trust** for various key management services (KMS)
  - Their root keys should be stored in HSMs
- **Secure physical separation and protection**
- **Satisfies security regulation** requirements such as FIPS 140-2
Hardware Security Modules (HSMs)

- **Root of trust** for various key management services (KMS)
  - Root keys should be stored in HSMs
- **Secure physical separation and protection**
- **Satisfies security regulation** requirements such as FIPS 140-2

FIPS 140-2

- **Physical separation required** (U.S. and Canadian security standard)

Host

Request cryptographic operations

Response (e.g., Digital signature)

Tamper-evident

Tamper-resistant

Root keys

HSM

Physical separation

Tamper-resistant
Demands for Scalable Security Services

Innovation in emerging cloud industries
- Microservices
- Edge computing
- Financial technology

Increase of secure network transactions
- User-to-Service
- Service-to-Service

Demands for scalable security services
- More cryptographic operations
- Low latency & High throughput
- Multiple user/key isolation
Problem: Limited Scalability of HSMs

Multiple services → Lots of request → Network → HSM (Dedicated hardware)

Performance bottleneck!

Signing speed: 10,000 tps (RSA-2048)
Price: $29,900
Problem: Limited Scalability of HSMs

- Expensive solution!
- Many on-premises HSMs
- Lots of requests
- Multiple services
- Network
**Problem: Limited Scalability of HSMs**

- **Multiple services**
- **Lots of requests**
- **Network**
- **Expensive solution!**

*Price: $1,250 per month (IBM Cloud HSM)*

Cloud HSM
Problem: Limited Scalability of HSMs

Can we efficiently scale out HSMs for key management services?
Alternative Approach

- Leverages commodity Trusted Execution Environment (TEE) instead of HSMs
  
Limitation of the Alternative Approach

- Leverages commodity Trusted Execution Environment (TEE) instead of HSMs


Does not provide physical separation & protection
Approach: Combining HSMs with TEE-based KMS

- Achieves cost-efficient scalability with SGX technology
- Maintains security level of physical separation with HSMs
- SGX enclaves and HSMs collaborate for key management
Deployment Assumption & Threat Model

- Microservices (KMS clients)
- KMS request
- Untrusted Platform
  - Root-privileged attacker
  - Fake Enclave
  - Invalid access
- Multiple SGX Enclaves (Trusted)
- Physical separation
- Root keys (Root-of-trust)
- HSM (Trusted)
Challenge 1: Scaling Performance

- Frequent private key operation requests to HSMs can incur performance bottleneck.

Untrusted Platform

Microservices (KMS clients)

Multiple SGX Enclaves

Physical separation

Root keys (Root-of-trust)

HSM
Challenge 1: Scaling Performance

- Frequent private key operation requests to HSMs can incur performance bottleneck.

1. Frequent short-living authentication requests

Heavy private key operations

Untrusted Platform

Multiple SGX Enclaves

Physical separation

Root keys (Root-of-trust)

HSM
Challenge 1: Scaling Performance

- Frequent private key operation requests to HSMs can incur performance bottleneck.
Challenge 1: Scaling Performance

- Frequent private key operation requests to HSMs can incur performance bottleneck.
Challenge 2: Validation between Enclaves and HSMs

- KMS clients, SGX enclaves and HSMs should trust each other.
- Lack of validation mechanism between SGX enclaves and HSMs.

Microservices (KMS clients) → Request → Response → Multiple SGX Enclaves → Physical separation → Root keys (Root-of-trust) → HSM
Challenge 2: Validation between Enclaves and HSMs

- KMS clients, SGX enclaves and HSMs should trust each others
- Lack of validation mechanism between SGX enclaves and HSMs
Design Goals of ScaleTrust

1. **Scalable performance**
   Enhances performance by scaling out and does not make an HSM a performance bottleneck

2. **Cost-effectiveness**
   Cost-efficiently scales out for key management services

3. **Security**
   Preserves a chain-of-trust from an HSM to clients
Design Overview

Untrusted Platform

Microservices (KMS clients)

Trusted Host

Bootstrapping Enclave

Physical separation

Root key pair (Root-of-trust)

HSM

KMS Enclaves
Design Overview

Untrusted Platform

Microservices (KMS clients)

KMS request

KMS Enclaves

Trusted Host

Bootstrapping Enclave

Physical separation

Root key pair (Root-of-trust)

HSM
Design Overview

Untrusted Platform

Microservices (KMS clients)

KMS request

KMS Enclaves

PKCS#11 API calls

Physical separation

Trusted Host

Bootstrapping Enclave

Root key pair (Root-of-trust)

HSM
**Design Overview**

- **Microservices (KMS clients)**
- **Trusted Host**
- **Bootstrapping Enclave**
- **Untrusted Platform**
- **KMS Enclaves**
  - **PKCS#11 API calls**
  - **Derived keys**
- **Physical separation**
  - **Root key pair** (Root-of-trust)
- **HSM**
Design Overview

- **Microservices (KMS clients)**
  - KMS request
  - KMS Enclaves
    - Derived keys
      - PKCS#11 API calls

- **Untrusted Platform**
  - Physical separation
    - Root key pair (Root-of-trust)
      - HSM
    - Offline key deployment (Trusted)
      - Root public key

- **Trusted Host**
  - Bootstrapping Enclave
Secure bootstrapping ①: An HSM generates a root key pair.
Secure bootstrapping ②: The HSM shares root public key with bootstrapping enclave

Secure bootstrapping:
The HSM shares root public key with bootstrapping enclave

Microservices (KMS clients)

Trusted Host

Bootstrapping Enclave

Untrusted Platform

KMS Enclaves

Physical separation

Root key pair (Root-of-trust)

HSM

Trusted Host

Physical separation

Untrusted Platform

KMS Enclaves

Root key pair (Root-of-trust)

HSM

Secure bootstrapping

Microservices (KMS clients)

Trusted Host

Bootstrapping Enclave

Untrusted Platform

KMS Enclaves

Physical separation

Root key pair (Root-of-trust)

HSM

Secure bootstrapping

Microservices (KMS clients)

Trusted Host

Bootstrapping Enclave

Untrusted Platform

KMS Enclaves

Physical separation

Root key pair (Root-of-trust)

HSM

Secure bootstrapping

Microservices (KMS clients)

Trusted Host

Bootstrapping Enclave

Untrusted Platform

KMS Enclaves

Physical separation

Root key pair (Root-of-trust)

HSM

Secure bootstrapping

Microservices (KMS clients)

Trusted Host

Bootstrapping Enclave

Untrusted Platform

KMS Enclaves

Physical separation

Root key pair (Root-of-trust)

HSM

Secure bootstrapping

Microservices (KMS clients)

Trusted Host

Bootstrapping Enclave

Untrusted Platform

KMS Enclaves

Physical separation

Root key pair (Root-of-trust)

HSM

Secure bootstrapping

Microservices (KMS clients)

Trusted Host

Bootstrapping Enclave

Untrusted Platform

KMS Enclaves

Physical separation

Root key pair (Root-of-trust)

HSM

Secure bootstrapping

Microservices (KMS clients)

Trusted Host

Bootstrapping Enclave

Untrusted Platform

KMS Enclaves

Physical separation

Root key pair (Root-of-trust)

HSM

Secure bootstrapping

Microservices (KMS clients)

Trusted Host

Bootstrapping Enclave

Untrusted Platform

KMS Enclaves

Physical separation

Root key pair (Root-of-trust)

HSM

Secure bootstrapping

Microservices (KMS clients)

Trusted Host

Bootstrapping Enclave

Untrusted Platform

KMS Enclaves

Physical separation

Root key pair (Root-of-trust)

HSM
Secure bootstrapping ③: The bootstrapping enclave attests KMS enclaves

Microservices (KMS clients)

Secure bootstrapping enclave

Trusted Host

Remote attestation

Untrusted Platform

KMS Enclaves

Physical separation

Root key pair (Root-of-trust)

HSM
Secure bootstrapping ④:
The bootstrapping enclave shares the public key

Microservices (KMS clients)

Trusted Host

Bootstrapping Enclave

Untrusted Platform

KMS Enclaves

Physical separation

Root key pair (Root-of-trust)

HSM

Key deployment
Secure bootstrapping ⑤: The KMS enclaves attest the HSM and build secure channels.

Microservices (KMS clients)
Secure bootstrapping:
A fake enclave cannot build a secure channel with the HSM
Attestation on SGX Instances

Attestation on enclaves ①: When the client first requests to KMS server, it allocates KMS enclaves for the client.
Attestation on SGX Instances

**Attestation on enclaves ②:** After a new KMS enclave is created, the bootstrapping enclave attests it.

Diagram:
- Microservices (KMS clients)
- Trusted Host
- Bootstrapping Enclave
- Remote attestation
- KMS Enclaves
- Secure channel
- Physical separation
- Root key pair (Root-of-trust)
- HSM

Untrusted Platform
Attestation on enclaves ③:
Also, the client performs remote attestation to verify the KMS enclave.
Attestation on enclaves 

After the remote attestation, the client sends encrypted KMS requests to the enclave.
Attestation on SGX Instances

Attestation on enclaves:
A fake enclave cannot build a communication channel with the client

Microservices (KMS clients)

Bootstrapping Enclave

Trusted Host

KMS Enclaves

PKCS#11 API calls

Physical separation

Root key pair (Root-of-trust)

HSM
Hierarchical Design for Scaling

- KMS requests
- Microservices (KMS clients)
- Scalable security services
- KMS Enclave
- Physical separation
- Root-of-trust: Root key pair (root-of-trust)
- HSM
Hierarchical Design for Scaling

KMS requests

Scalable security services

Derived keys

Root-of-trust

KMS Enclave

Root key operation requests

Physical separation

Root key pair (root-of-trust)

HSM
Hierarchical Design for Scaling

KMS requests

Microservices (KMS clients)

Frequent cryptographic requests

Scalable security services

 Derived keys

Root key operation requests

Physical separation

Root key pair (root-of-trust)

HSM
JSON Web Token (JWT) for Microservice

JWT client

R : Refresh token
   (Lifetime: few hours)

A : Access token
   (Lifetime: more than a week)
JSON Web Token (JWT) for Microservice

- **JWT client**
  - R: Refresh token (Lifetime: few hours)
  - A: Access token (Lifetime: more than a week)

**Refresh token request**
JSON Web Token (JWT) for Microservice

**JWT client**

- **R**: Refresh token (Lifetime: few hours)
- **A**: Access token (Lifetime: more than a week)

**JWT auth server**

- Creates and signs the refresh token

**Refresh token request**
JSON Web Token (JWT) for Microservice

Web server

JWT client

JWT auth server

Access token request

R: Refresh token
(Lifetime: few hours)

A: Access token
(Lifetime: more than a week)

Verifies the refresh token
and sends a new access token

Refresh token key pair
Application Case Study: JWT Management

![Diagram of JWT Management System]

- **JWT client**
  - Physical separation
  - KMS Enclaves
  - HSM

- **JWT auth server**
  - Refresh token request

- **KMS**
  - Enclaves

- **HSM**
  - Refresh token key pair

- **Refresh token request**
  - **R**: Refresh token (Lifetime: few hours)
  - **A**: Access token (Lifetime: more than a week)
Application Case Study: JWT Management

**JWT client**

- **R**: Refresh token (Lifetime: few hours)
- **A**: Access token (Lifetime: more than a week)

**JWT auth server**

- Refresh token request

**Enclaves**

- Public key of refresh token

**Physical separation**

- Refresh token request

**HSM**

- Refresh token key pair
Application Case Study: JWT Management

**JWT client**

- **R**: Refresh token (Lifetime: few hours)
- **A**: Access token (Lifetime: more than a week)

**JWT auth server**

- Access token request
  - Validate the refresh token
  - KMS Enclaves

**Physical separation**

- Refresh token key pair
  - HSM
Application Case Study: JWT Management

**JWT client**

- **R**: Refresh token (Lifetime: few hours)
- **A**: Access token (Lifetime: more than a week)

**JWT auth server**

- **Access token request**
- **Validate the refresh token**

**Physical separation**

- **Refresh token key pair**
- **HSM**

**Enclaves**
Application Case Study: JWT Management

JWT client

JWT auth server

Access token request

Enclaves

Refresh token request

Physical separation

Refresh token key pair

HSM

R: Refresh token (Lifetime: few hours)
A: Access token (Lifetime: more than a week)
Preliminary Evaluation

- **Environment setup**
  - CPU: Quad-core Intel Xeon E3-1280 v6 (SGX-enabled)
  - Intel SGX Linux SDK version 2.5
  - We use SoftHSM to emulate an HSM device.
  - Each enclave and HSM performs the same SHA-256 with RSA-2048 signing

[Diagram showing the process of JWT client sending token requests to an SGX-equipped server, which interacts with KMS Enclaves and SoftHSM using PKCS#11 API calls and Enclave calls.]
Preliminary Evaluation: Latency Improvement

- Scaling out KMS enclaves for latency improvement
### Preliminary Evaluation: Cost-effective Scaling

<table>
<thead>
<tr>
<th>Approach for KMS</th>
<th>Equipment</th>
<th>Performance (RSA-2048 sign)</th>
<th>Price</th>
<th>tps/$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ScaleTrust</strong> (on-premises SGX machine)</td>
<td>Xeon E3-1280 v6 CPU (Quad, 4.2 GHz)</td>
<td>3,600 tps</td>
<td>$500</td>
<td>7.2</td>
</tr>
<tr>
<td>On-premises HSMs-only</td>
<td>Luna SA A790 HSM</td>
<td>10,000 tps</td>
<td>$29,900</td>
<td>0.33</td>
</tr>
<tr>
<td><strong>ScaleTrust</strong> (in Azure cloud)</td>
<td>Xeon E-2176G CPU (Quad, 4.7 GHz)</td>
<td>&gt; 3,600 tps (estimated)</td>
<td>$500 per month</td>
<td>&gt; 7.2 for a month</td>
</tr>
<tr>
<td>Cloud HSM (Azure HSM)</td>
<td>Luna SA A790 HSM</td>
<td>10,000 tps</td>
<td>$5000 + $3,541 per month</td>
<td>1.17 for a month</td>
</tr>
</tbody>
</table>

*tps = transactions per second
Future work

- Evaluation with a real HSM device
Future work

- Physical separation by Intel VCA (SGX card)

![Diagram]

- Host
  - SGX node manager
  - MMIO region

- Intel VCA card

- PCIe communication
Conclusion

- We explore new design space to address the limited **scalability of HSMs** by combining TEE technology

- ScaleTrust preserves **chain-of-trust** from an HSM to clients

- ScaleTrust utilizes HSMs and SGX enclaves in a hierarchical model to **relieve the burden of HSMs**

- Our JWT case study shows that ScaleTrust can be applied to key management for microservices.
Thank You