

# Toward Scaling Hardware Security Module for Emerging Cloud Services

Juhyeng Han\*, Seongmin Kim\*, Taesoo Kim<sup>†</sup>, Dongsu Han

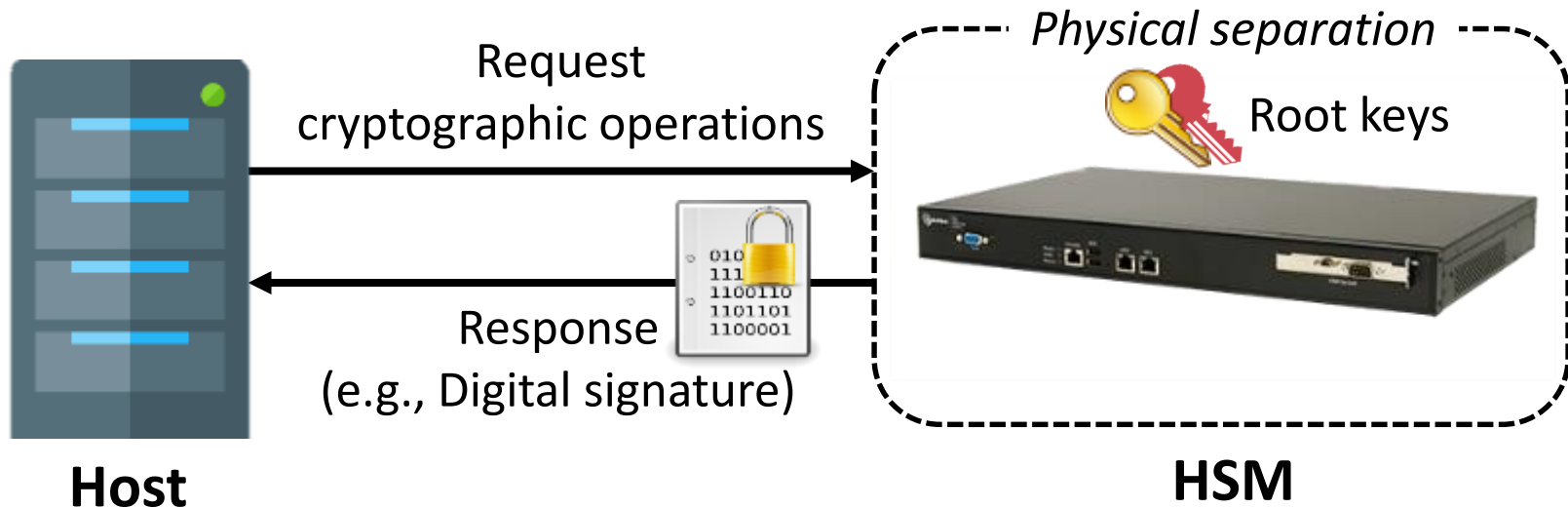
KAIST<sup>†</sup> Georgia Tech

\* The first two authors contributed equally to this work.

# Hardware Security Modules (HSMs)

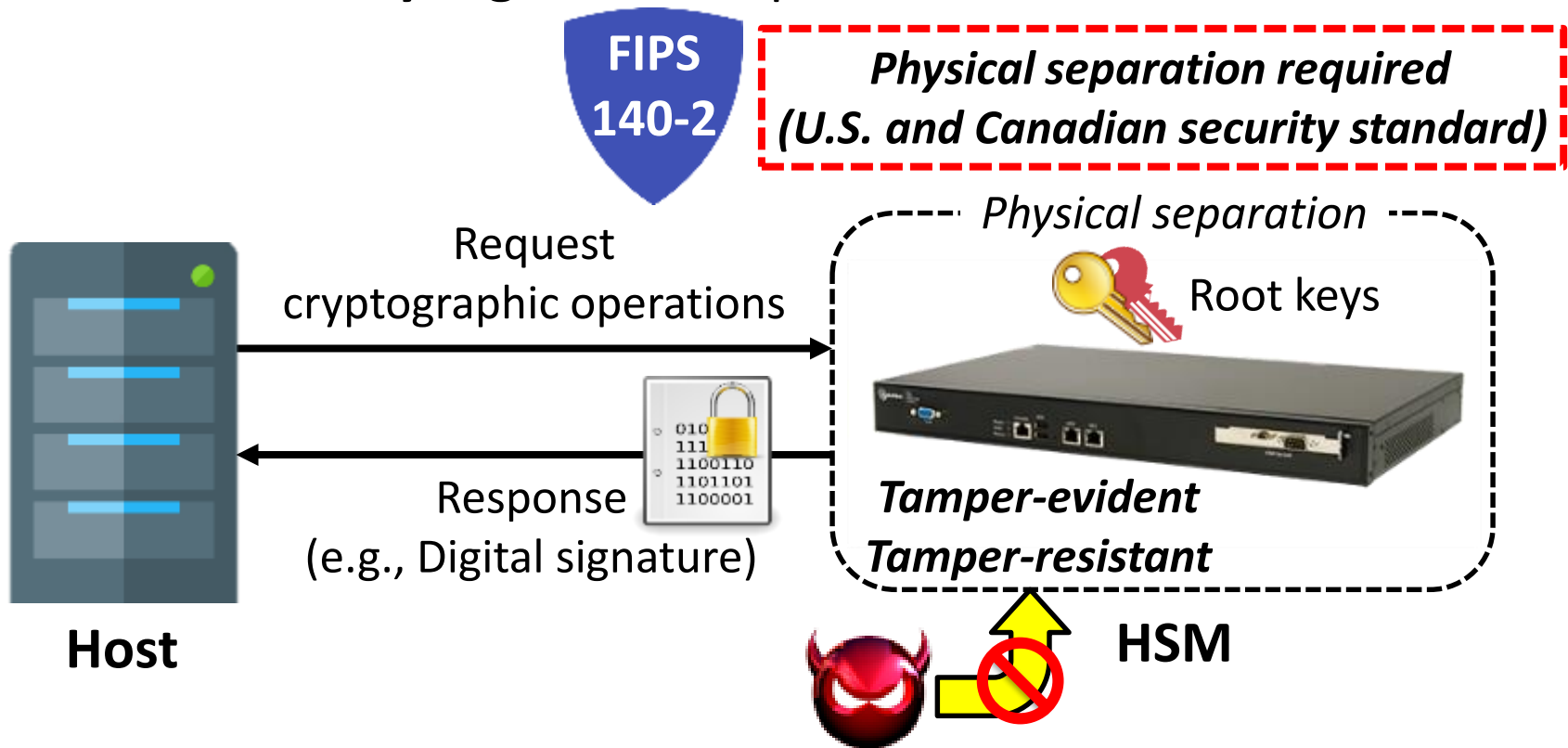
---

- **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



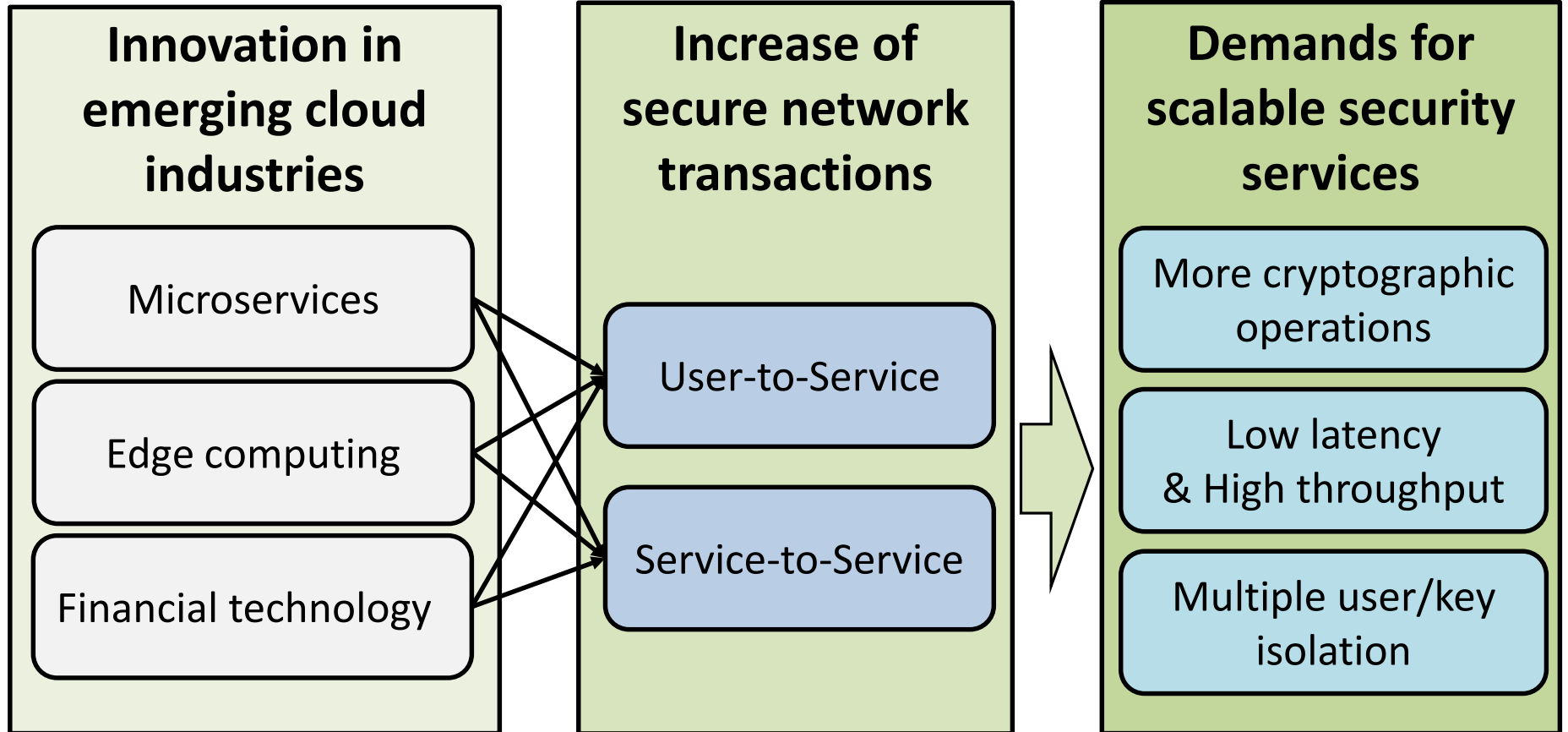
# Hardware Security Modules (HSMs)

- **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

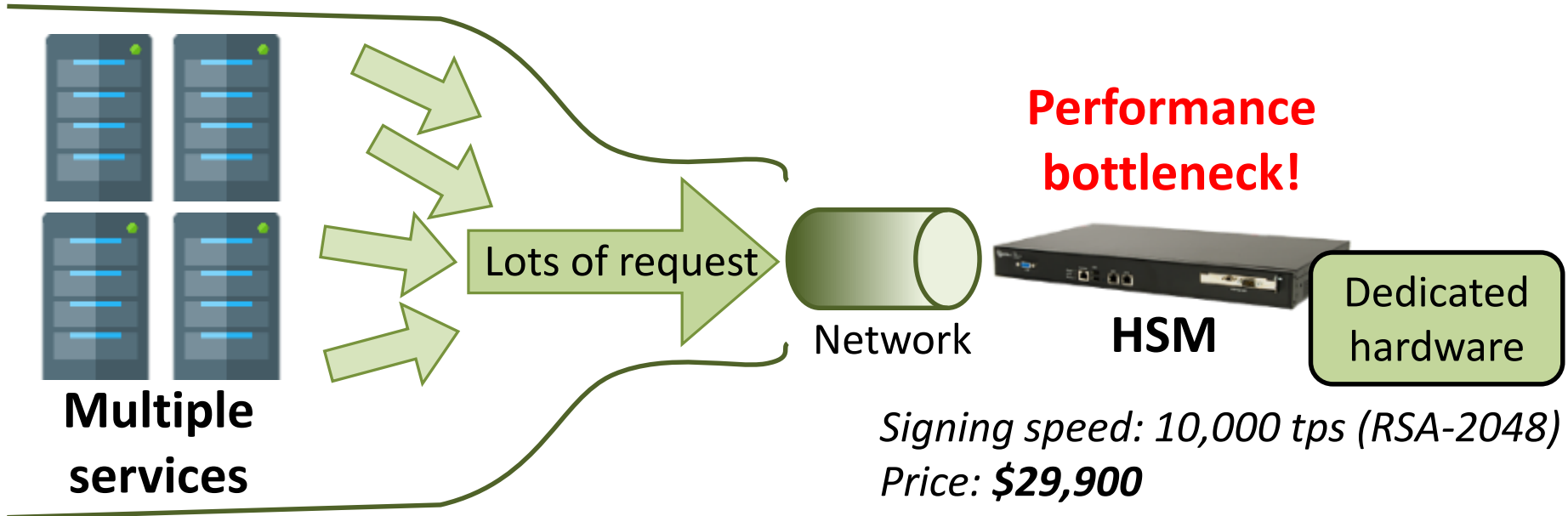


# Demands for Scalable Security Services

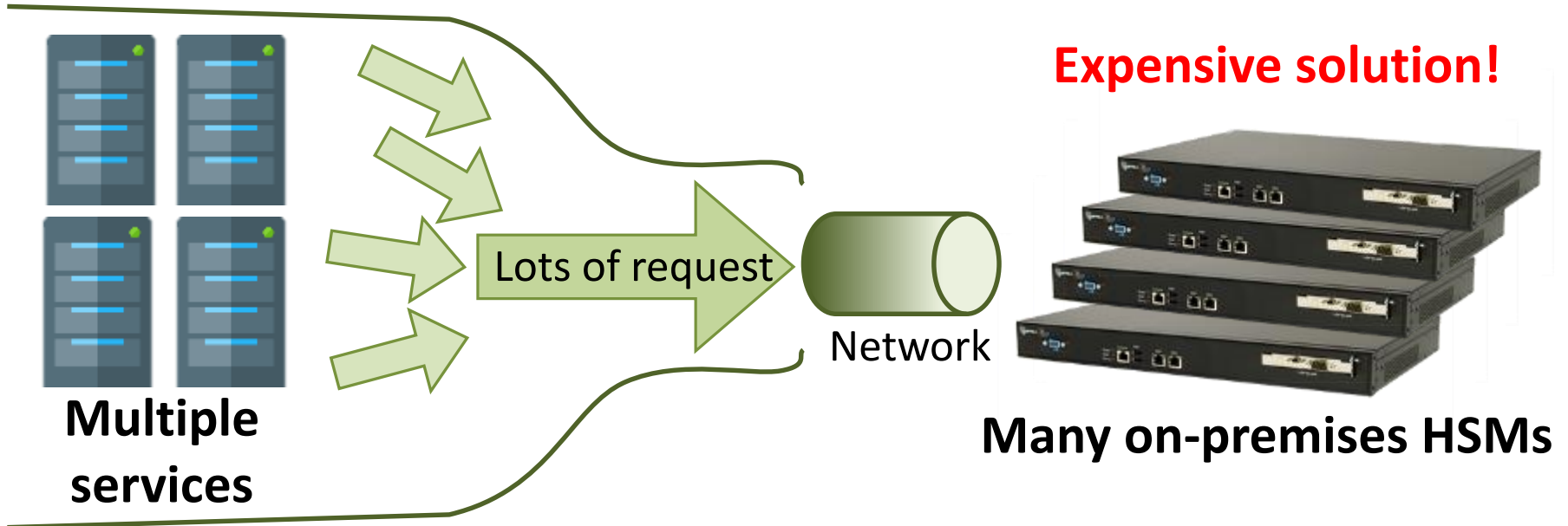
---



# Problem: Limited Scalability of HSMs

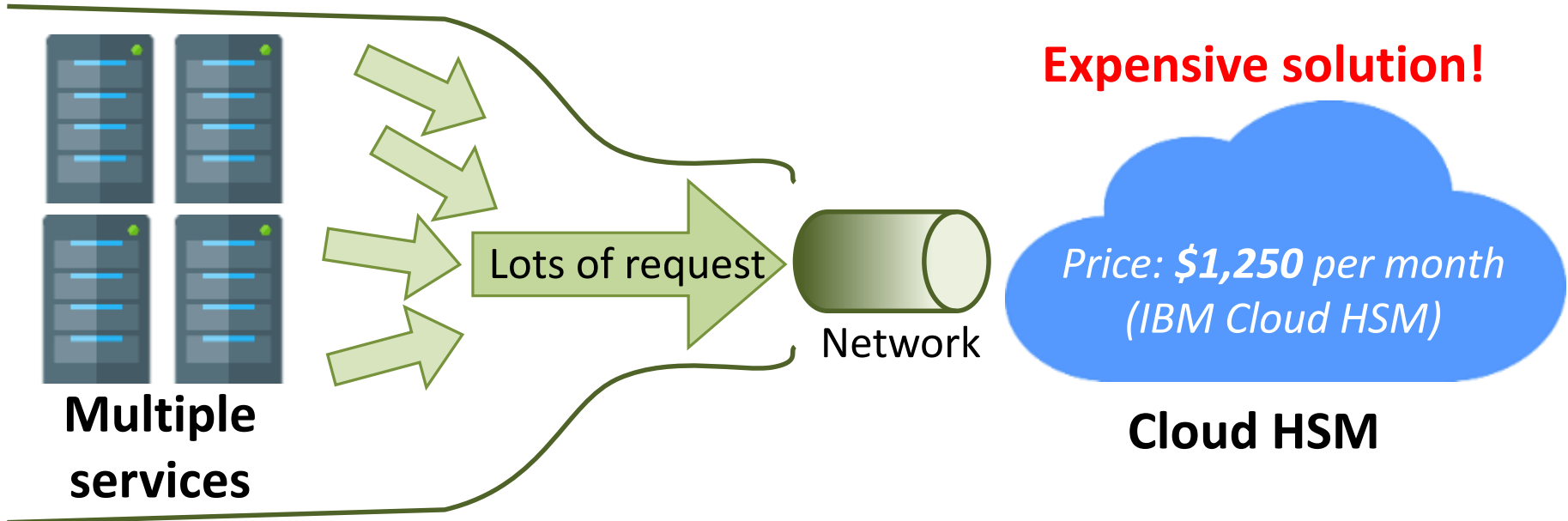


# Problem: Limited Scalability of HSMs



# Problem: Limited Scalability of HSMs

---



## **Problem:** Limited Scalability of HSMs

---

**Can we efficiently scale out HSMs  
for key management services?**

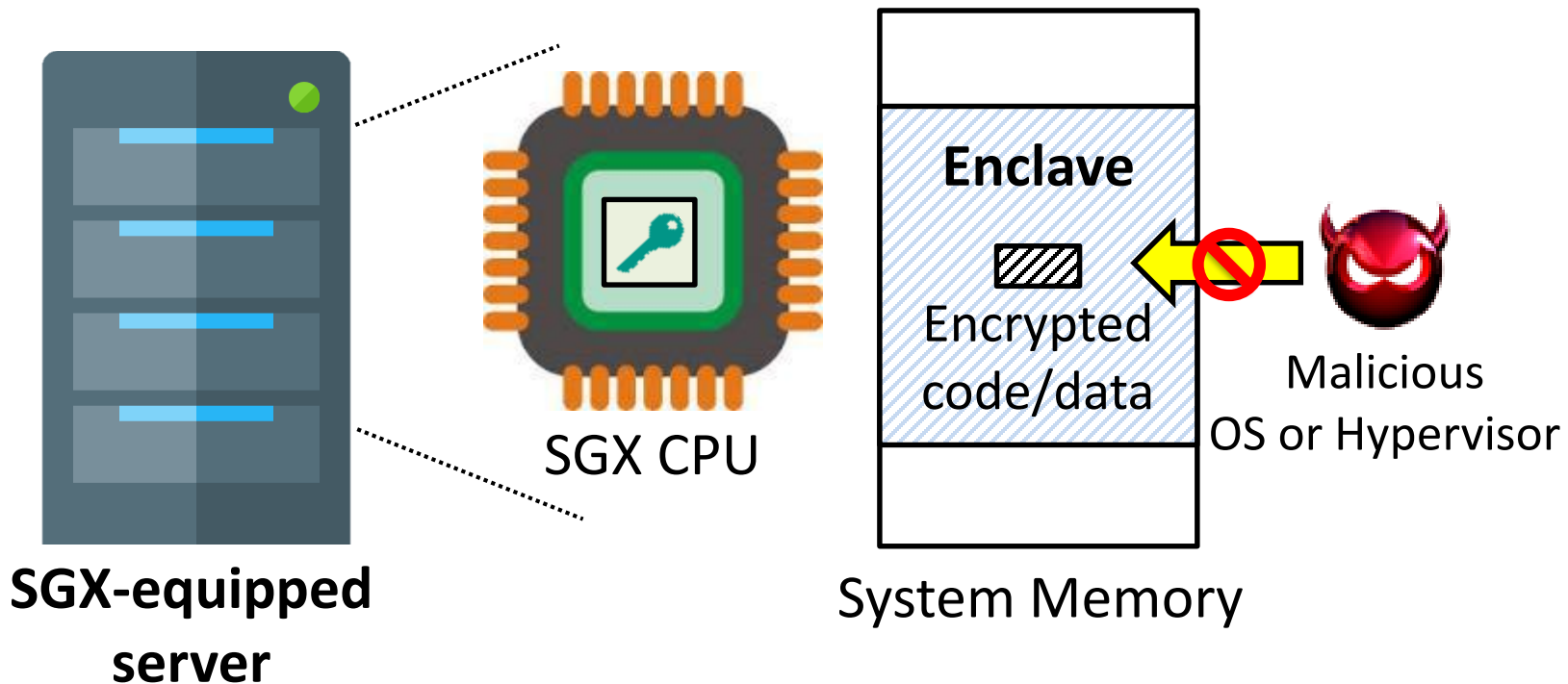
services



# Alternative Approach

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

[S. Chakrabarti et al. “Intel® SGX Enabled Key Manager Service with OpenStack Barbican.” arXiv preprint arXiv:1712.07694, 2017.]

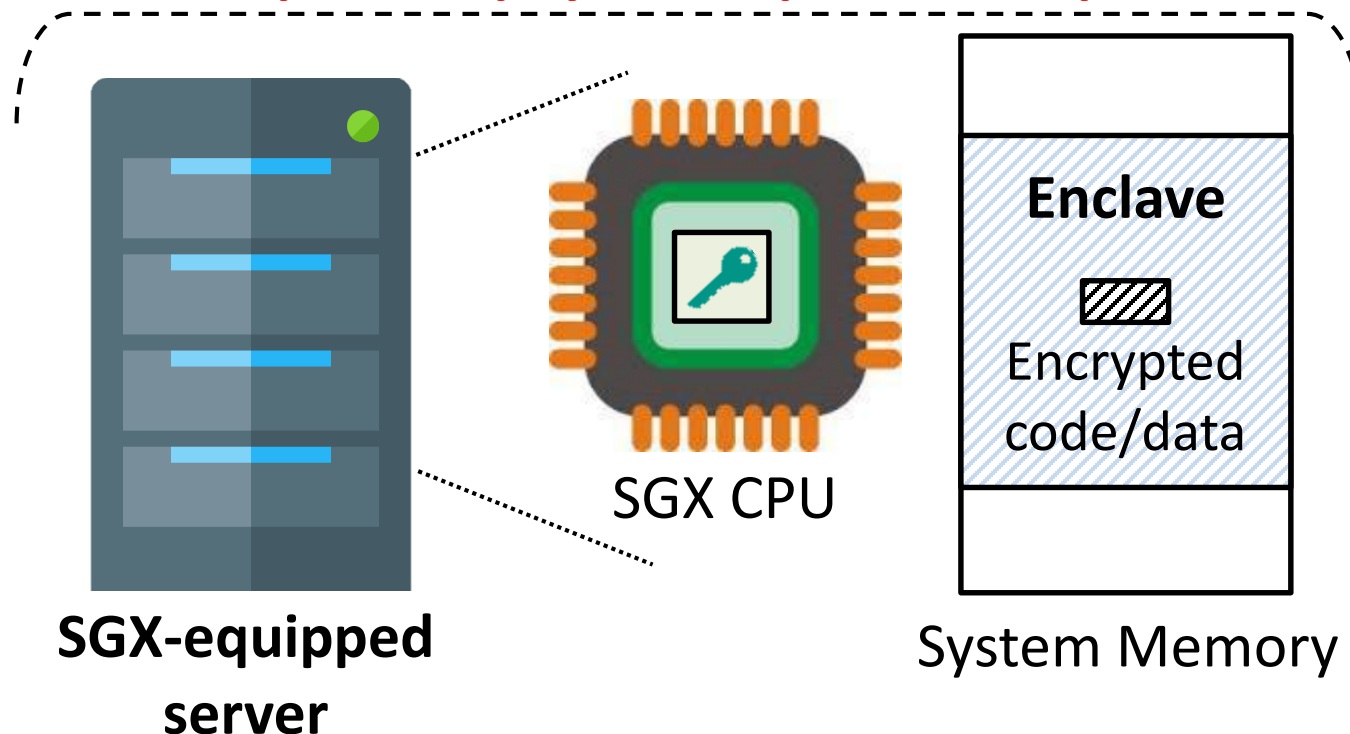


# Limitation of the Alternative Approach

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

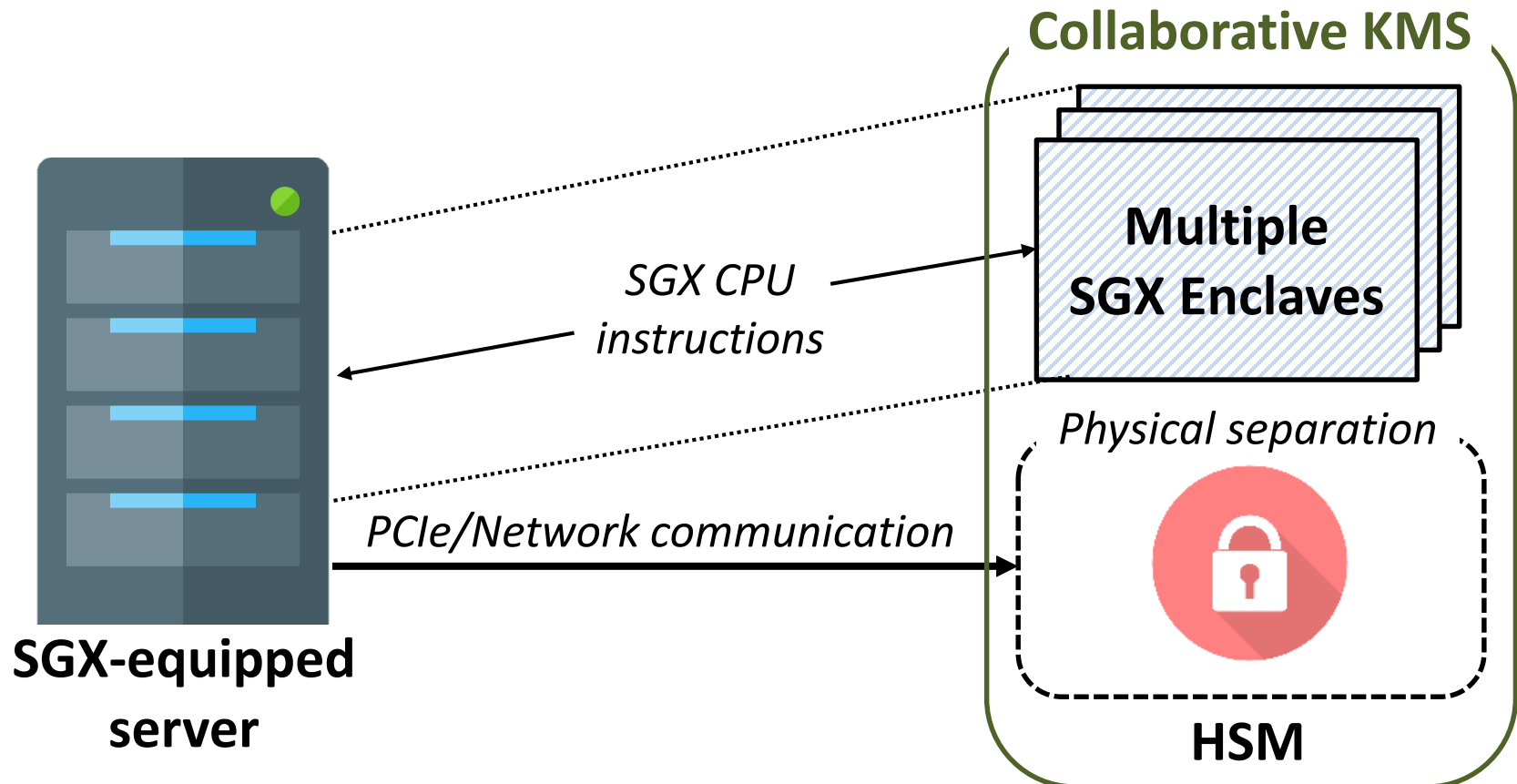
[S. Chakrabarti et al. “Intel® SGX Enabled Key Manager Service with OpenStack Barbican.” arXiv preprint arXiv:1712.07694, 2017.]

**Does not provide physical separation & protection**

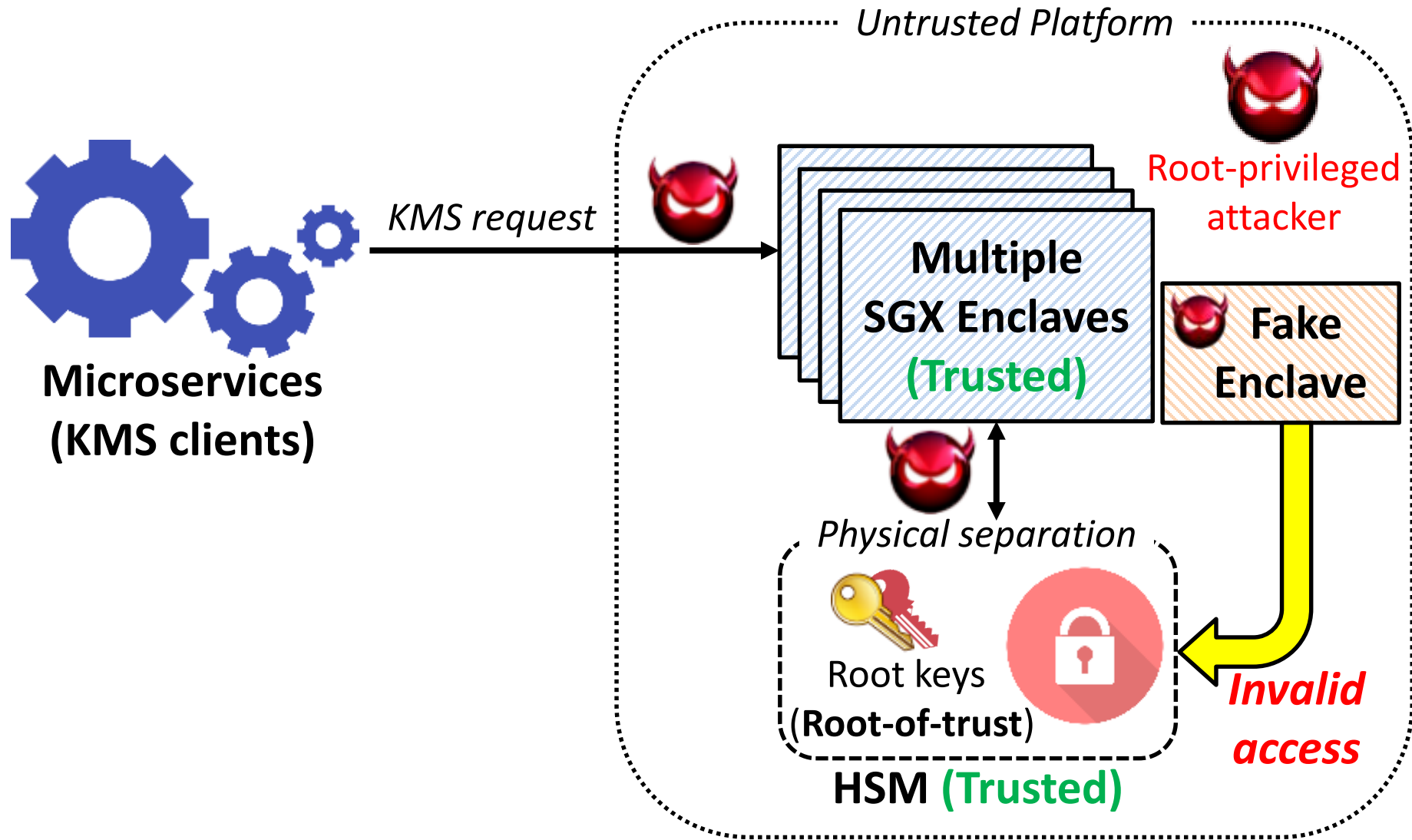


# Approach : Combining HSMs with TEE-based KMS

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

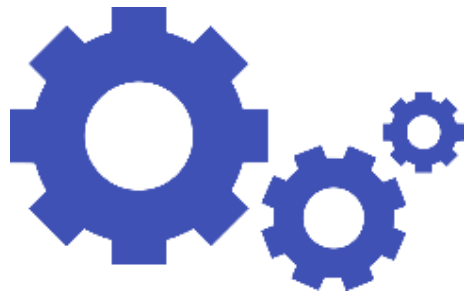


# Deployment Assumption & Threat Model

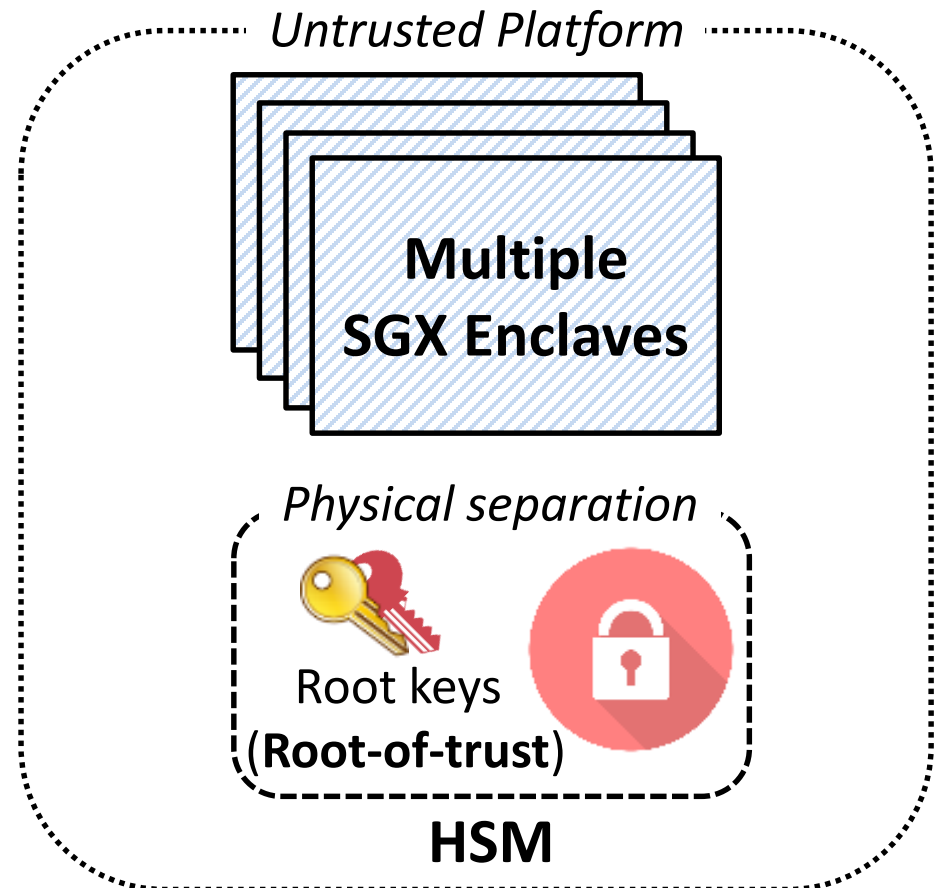


# Challenge 1 : Scaling Performance

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

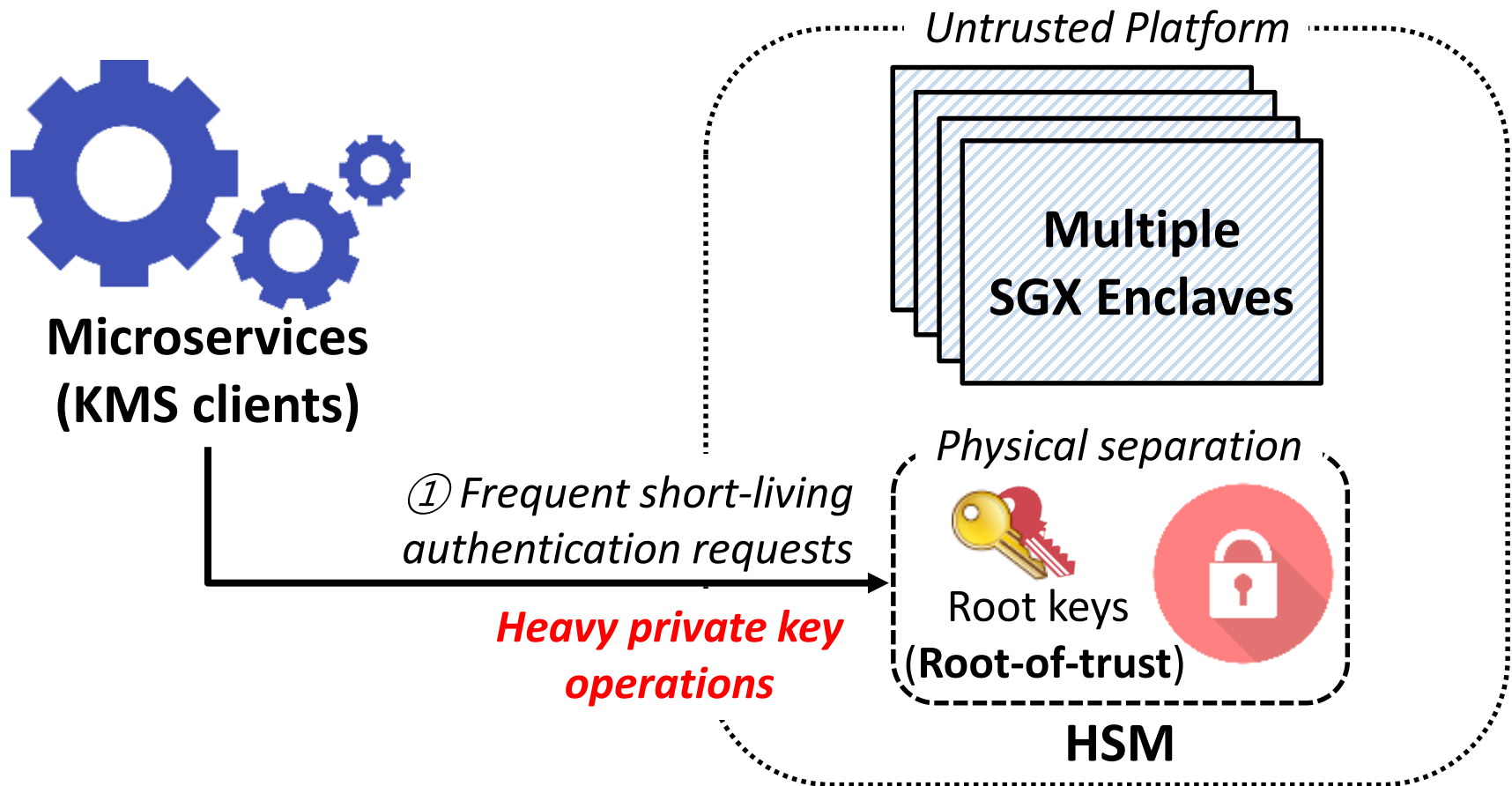


**Microservices  
(KMS clients)**



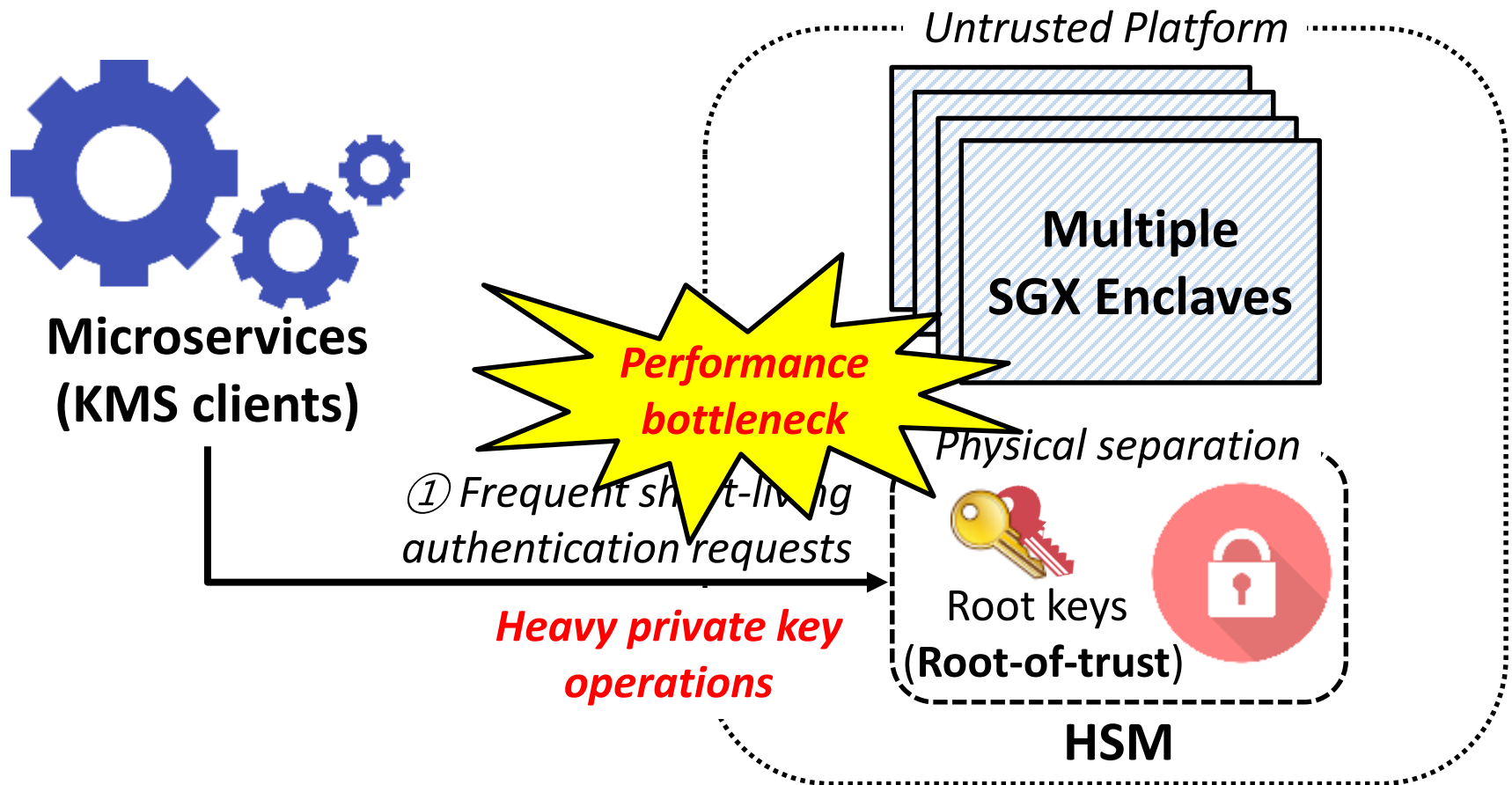
# Challenge 1 : Scaling Performance

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



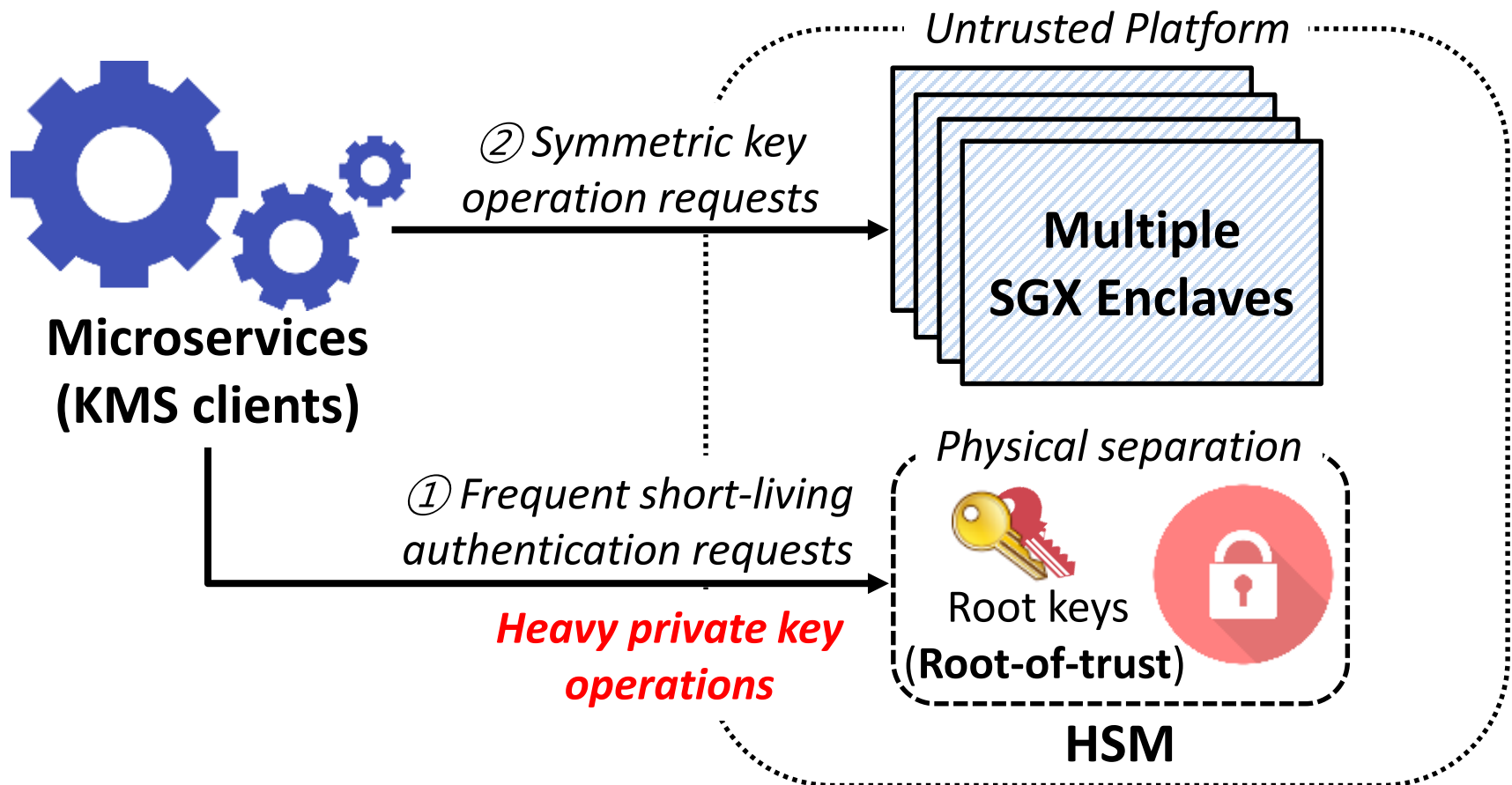
# Challenge 1 : Scaling Performance

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



# Challenge 1 : Scaling Performance

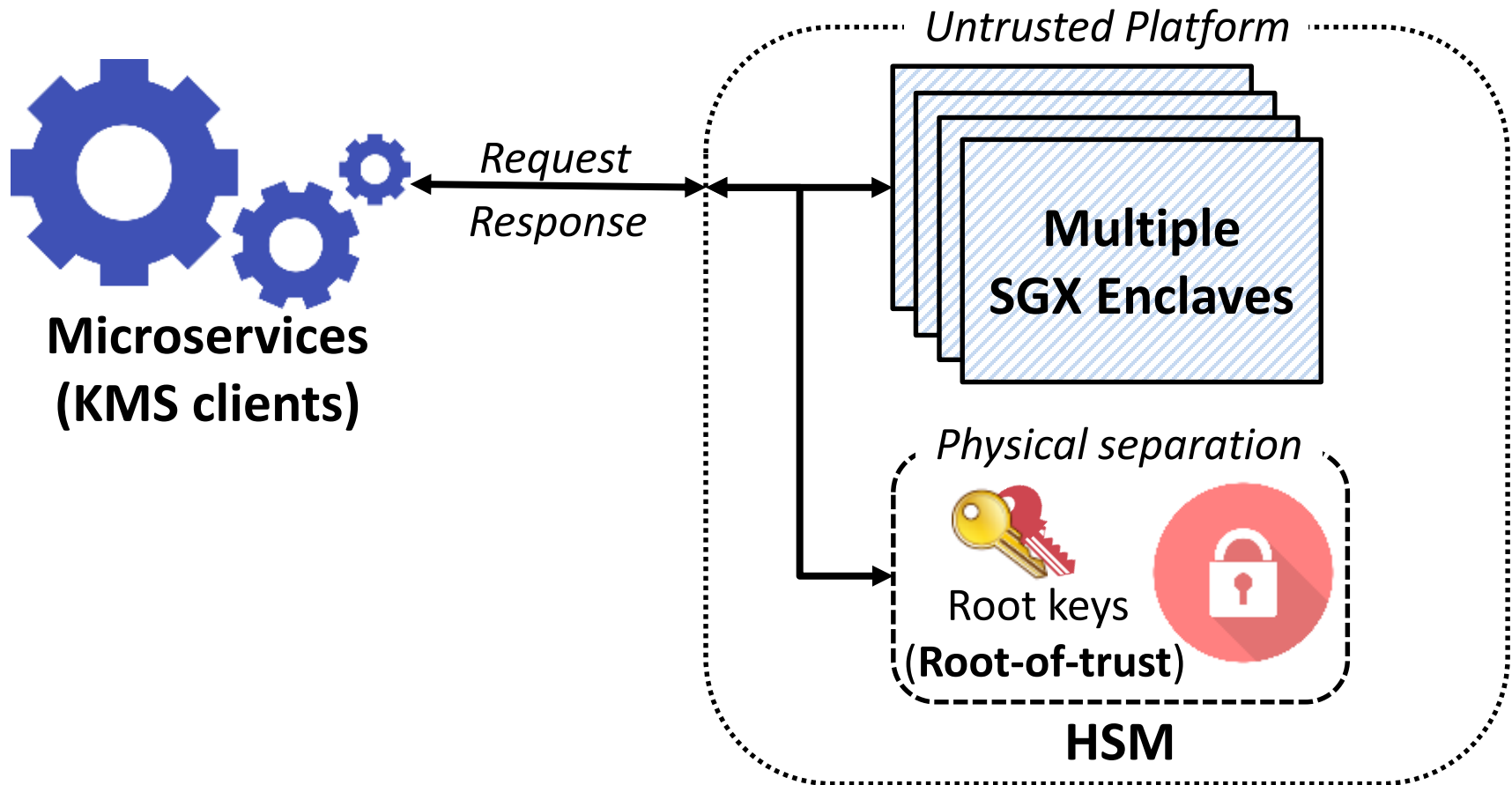
- Frequent private key operation requests to HSMs can incur performance bottleneck.





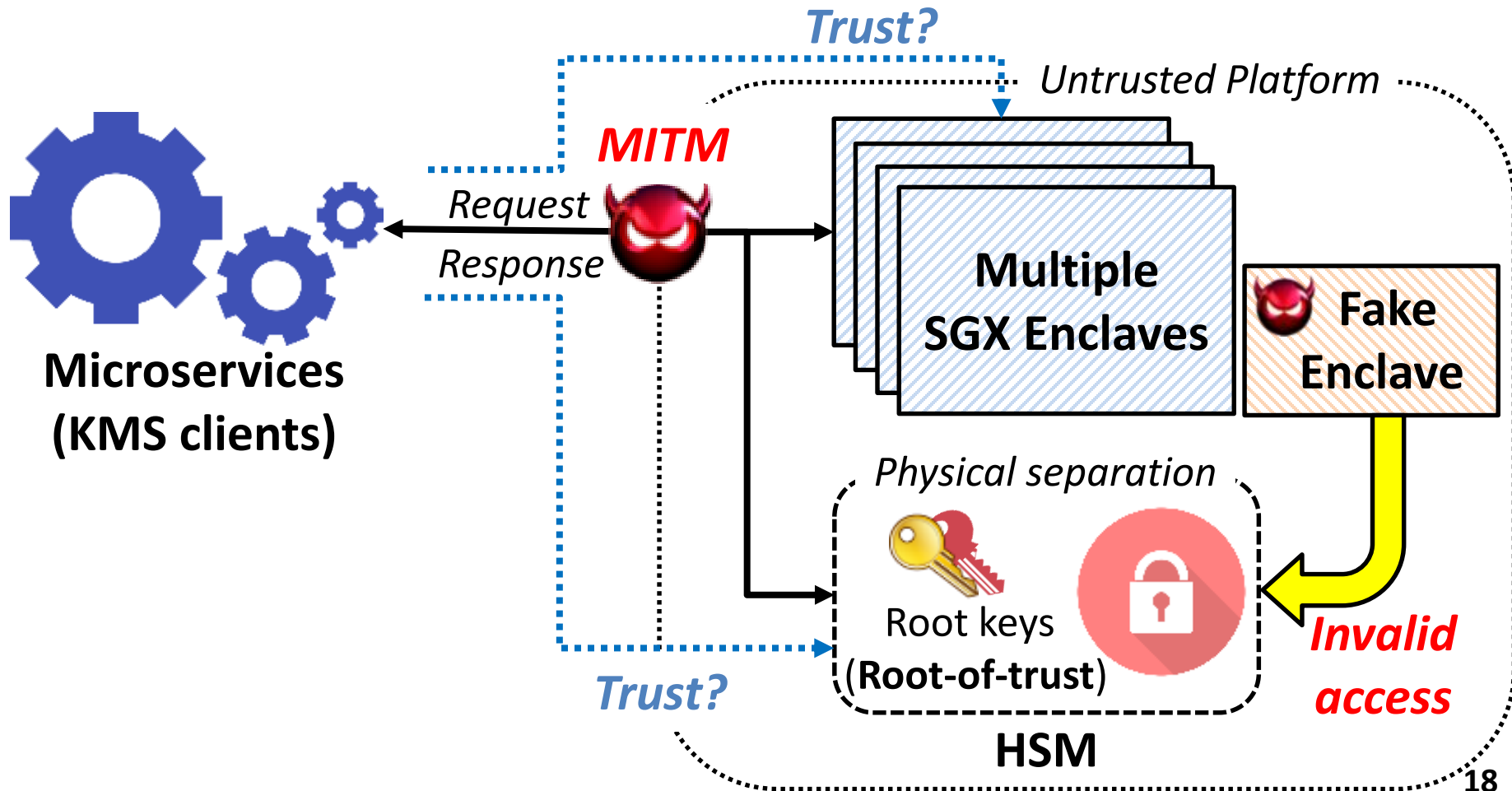
# Challenge 2 : Validation between Enclaves and HSMs

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



# 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

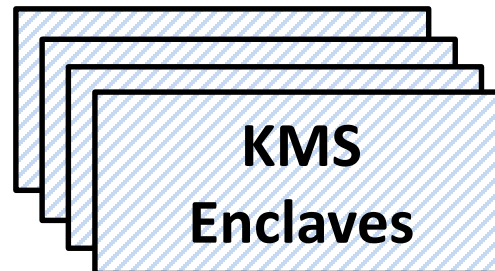


**Microservices  
(KMS clients)**

*Trusted Host*

**Bootstrapping  
Enclave**

*Untrusted Platform*



**KMS  
Enclaves**

*Physical separation*

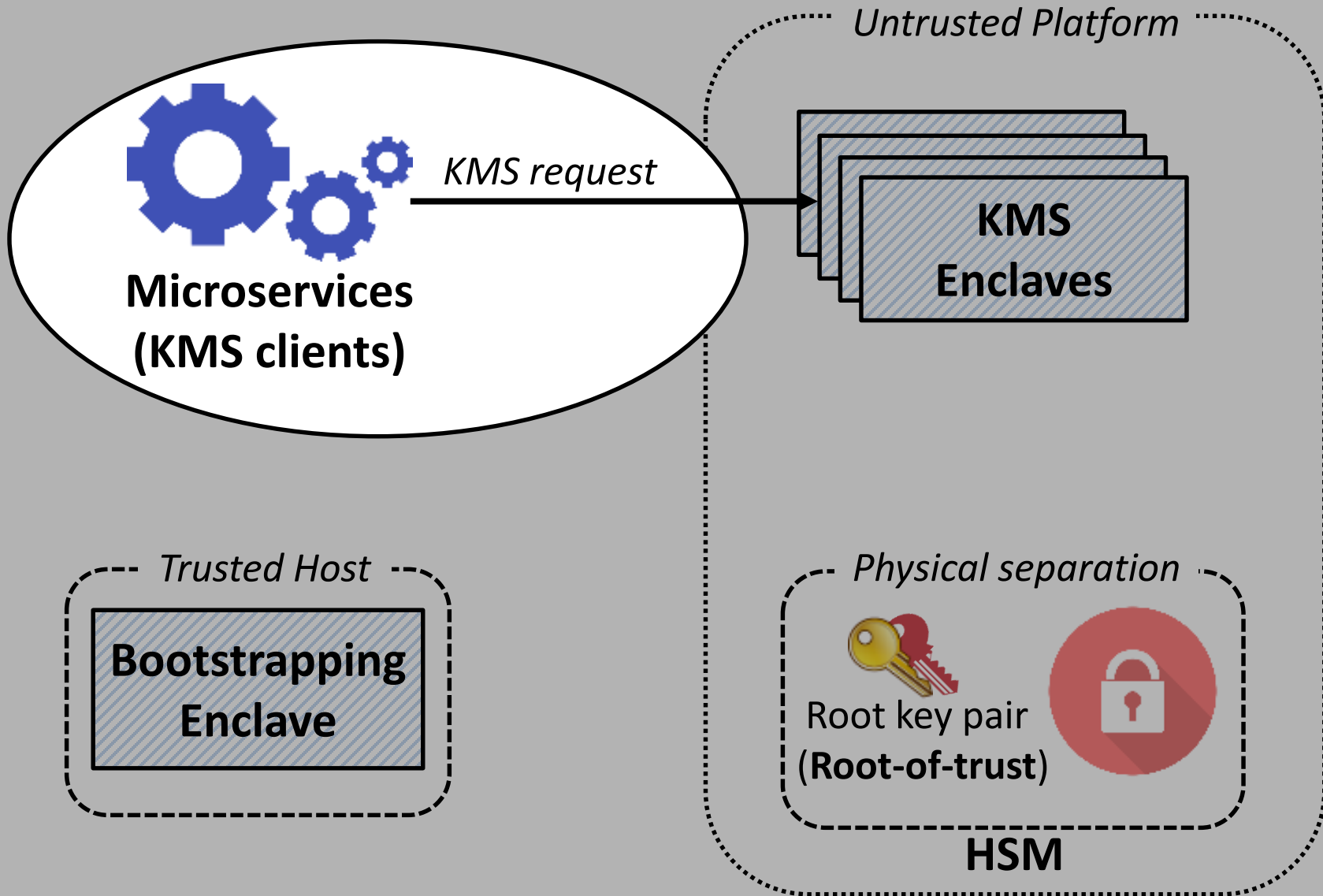


**Root key pair  
(Root-of-trust)**

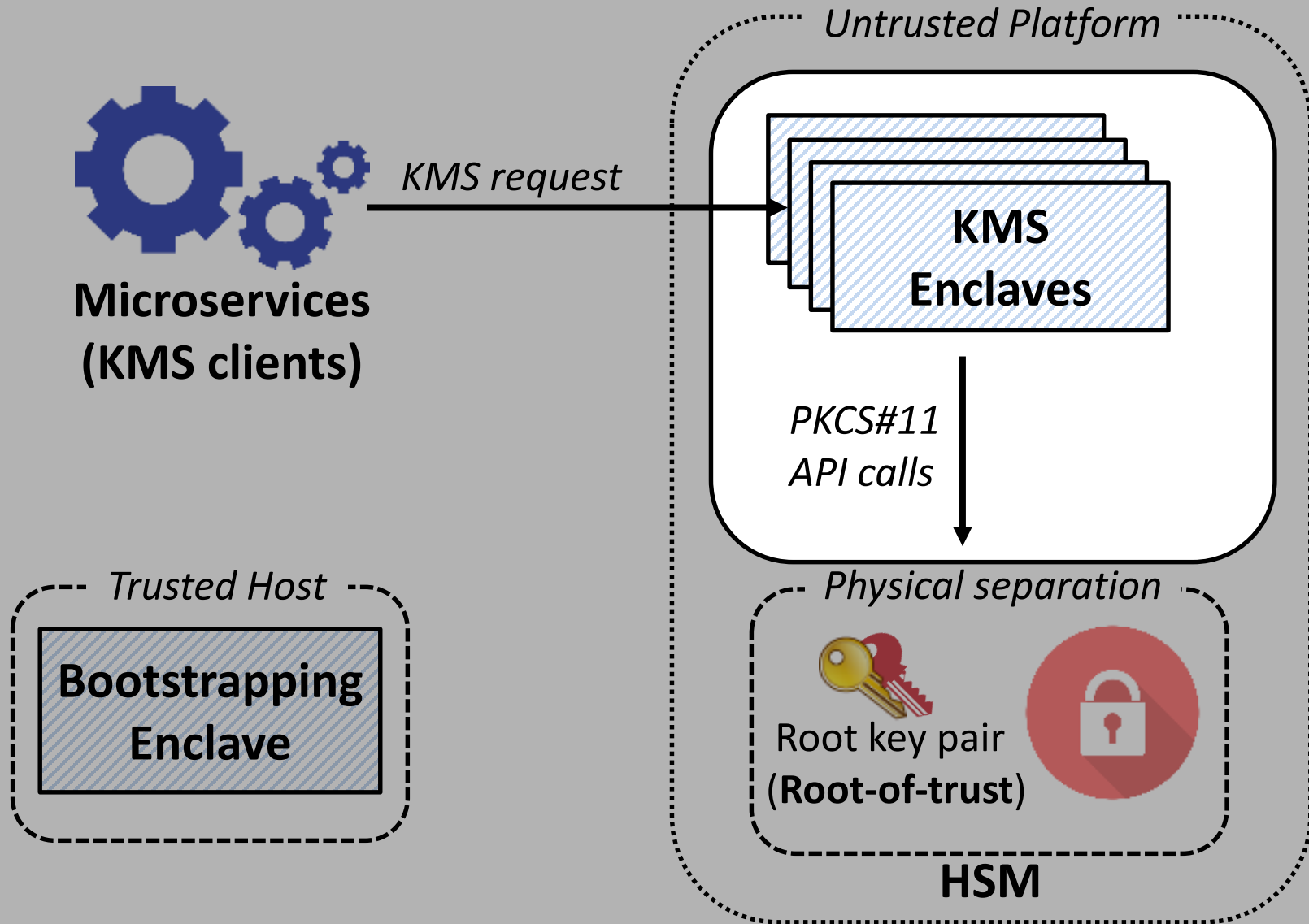


**HSM**

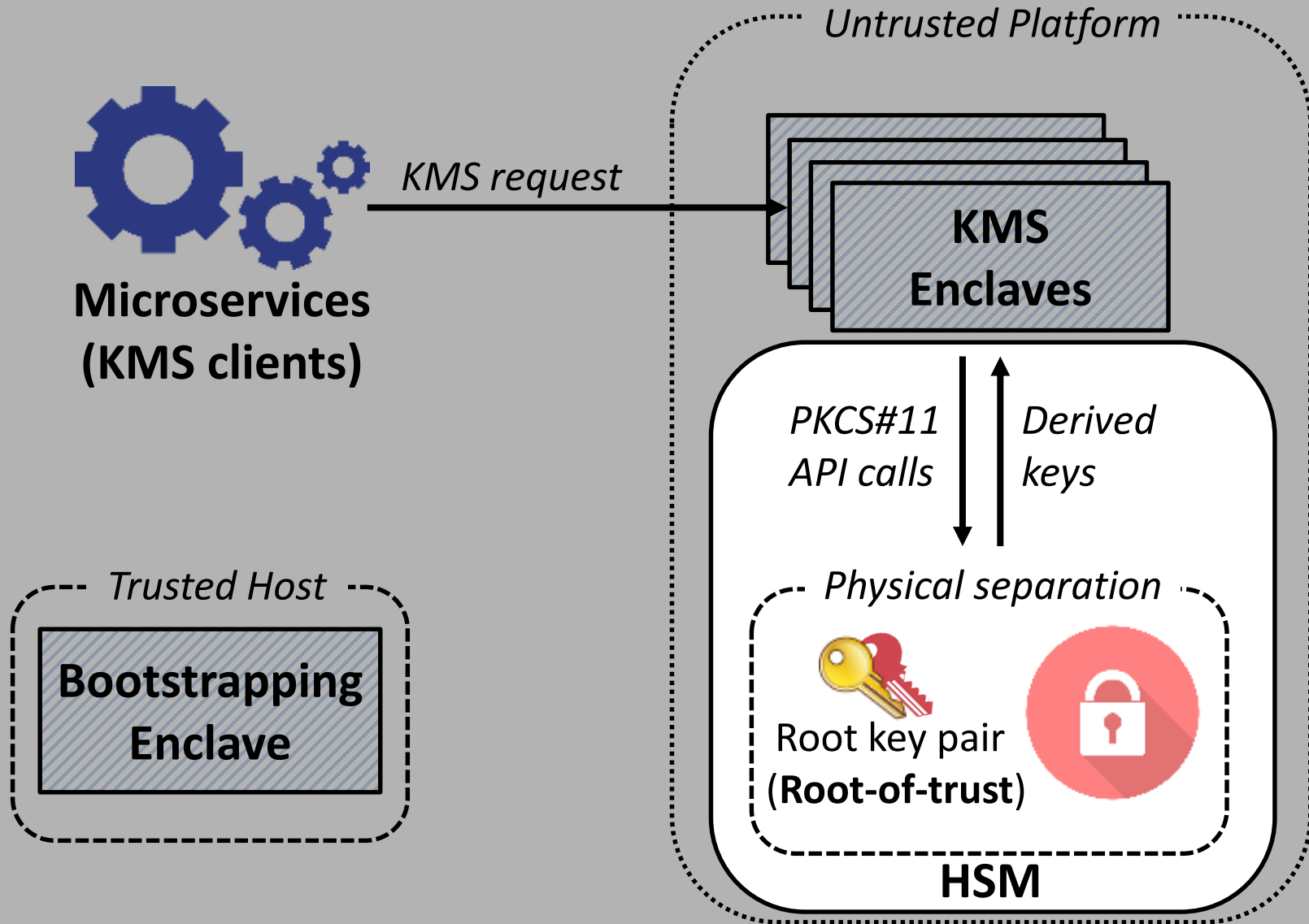
# Design Overview



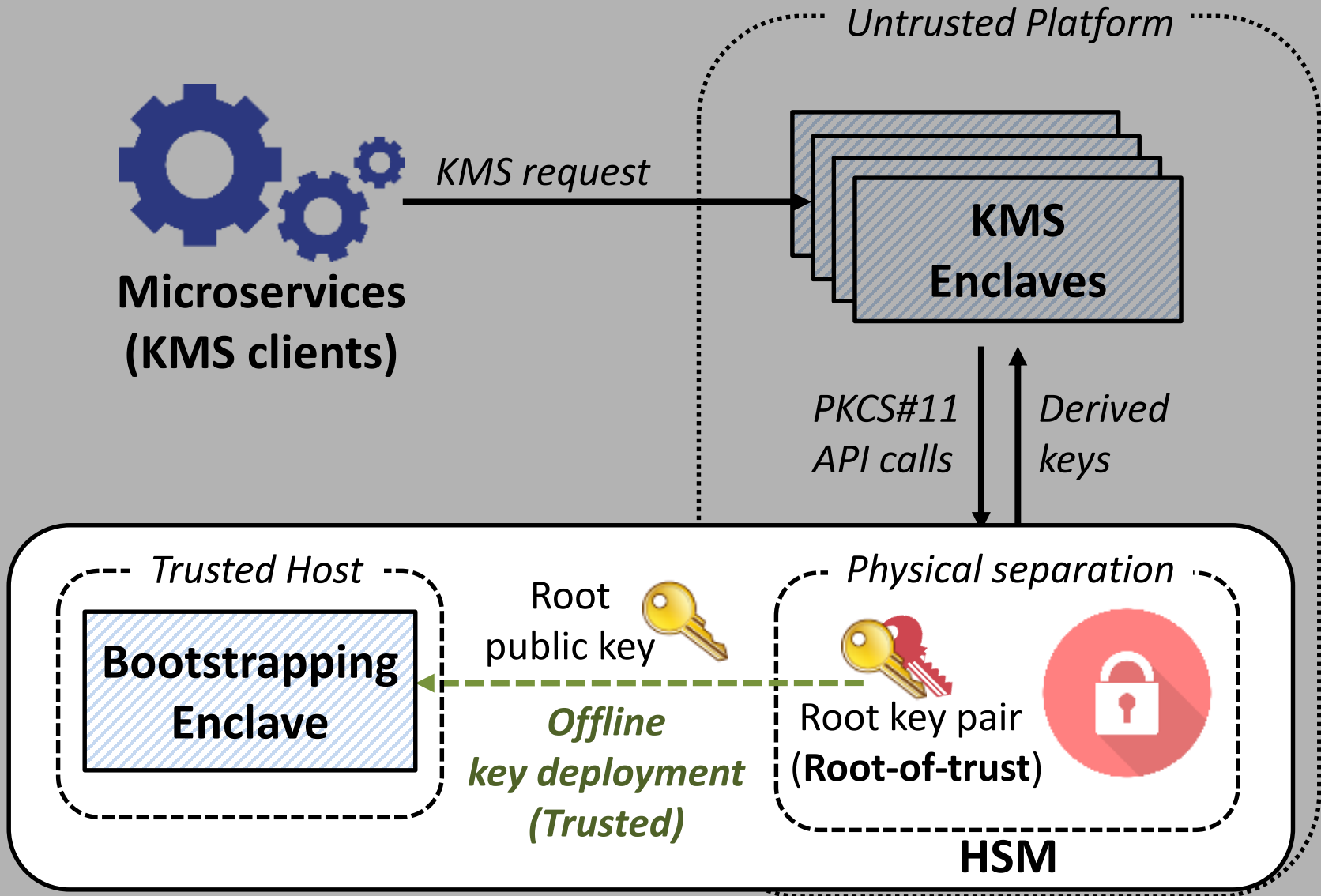
# Design Overview



# Design Overview



# Design Overview





# Secure bootstrapping

## Secure bootstrapping ① :

An HSM generates a root key pairs

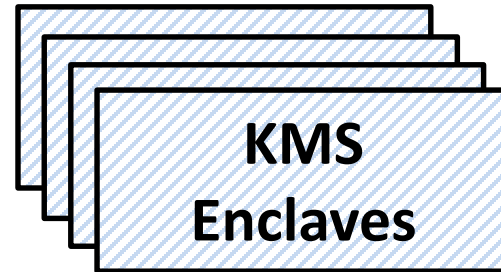


**Microservices  
(KMS clients)**

*Trusted Host*

**Bootstrapping  
Enclave**

*Untrusted Platform*



*Physical separation*



Root key pair  
(Root-of-trust)



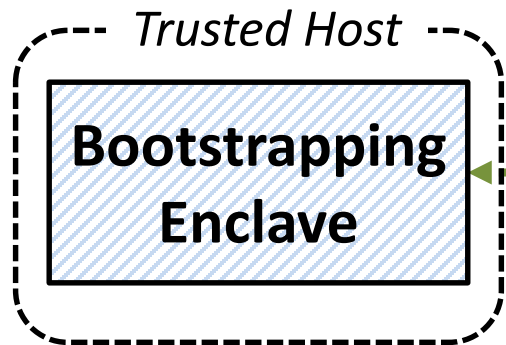
**HSM**

# Secure bootstrapping

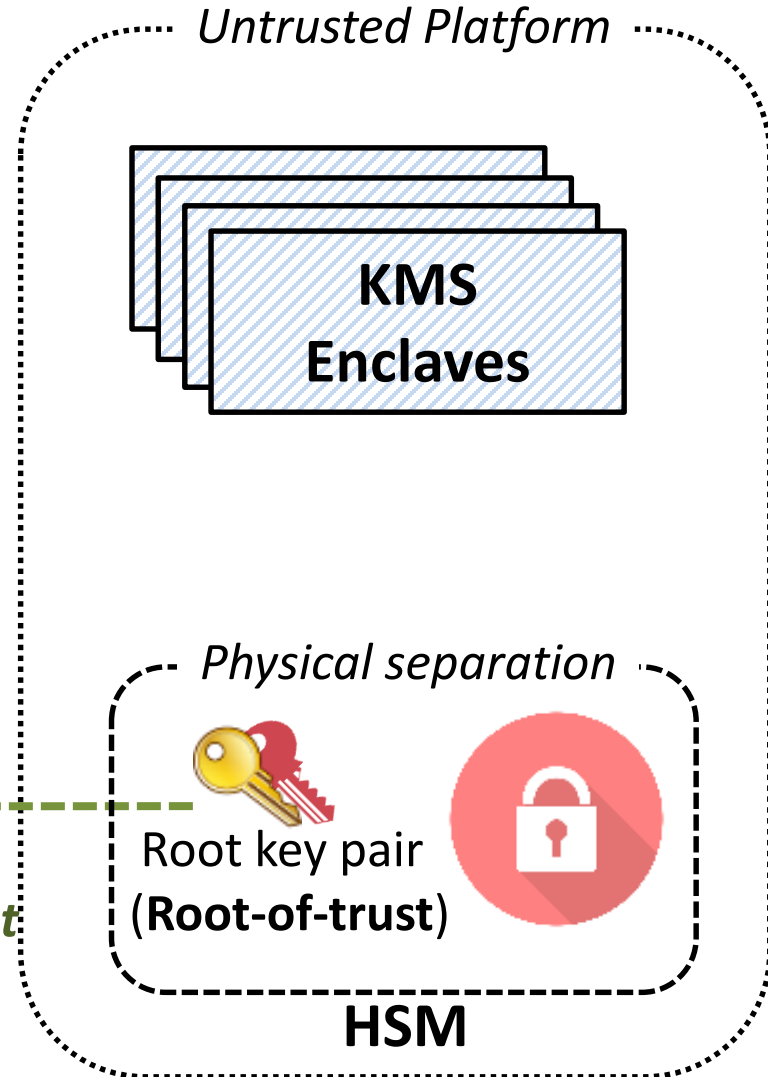
**Secure bootstrapping ② :**  
The HSM shares root public key with bootstrapping enclave



**Microservices  
(KMS clients)**



*Offline  
key deployment  
(Trusted)*



# Secure bootstrapping

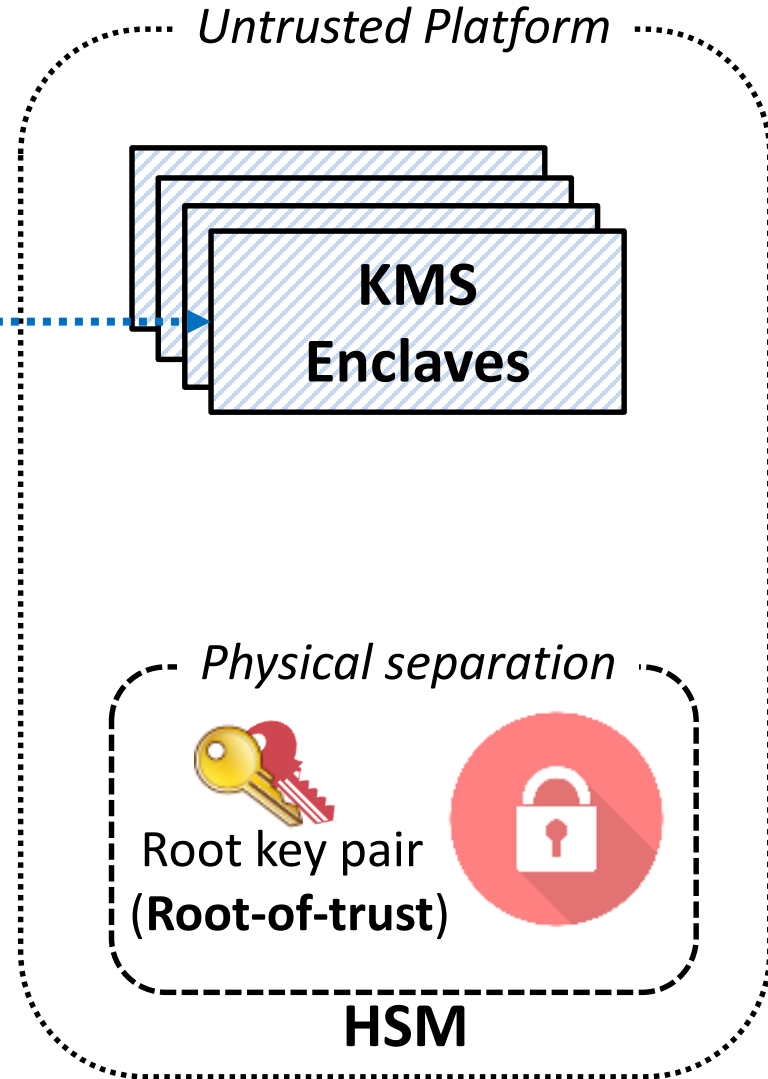
**Secure bootstrapping ③ :**  
The bootstrapping enclave  
attests KMS enclaves



**Microservices  
(KMS clients)**



*Remote  
attestation*



# Secure bootstrapping

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



**Microservices  
(KMS clients)**

*Trusted Host*



**Bootstrapping  
Enclave** 

*Key  
deployment*

*Untrusted Platform*

**KMS  
Enclaves**

*Physical separation*

   
Root key pair  
(Root-of-trust)

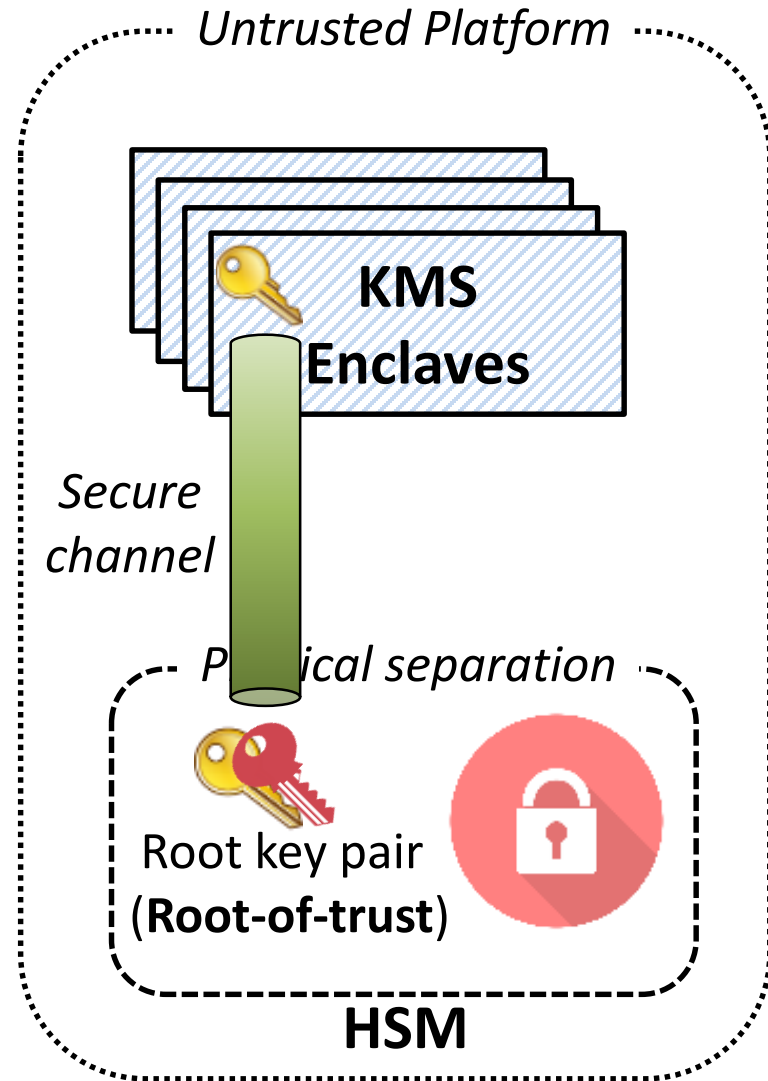
**HSM**

# Secure bootstrapping

**Secure bootstrapping ⑤ :**  
The KMS enclaves attest the HSM and build secure channels



**Microservices  
(KMS clients)**



# Secure bootstrapping

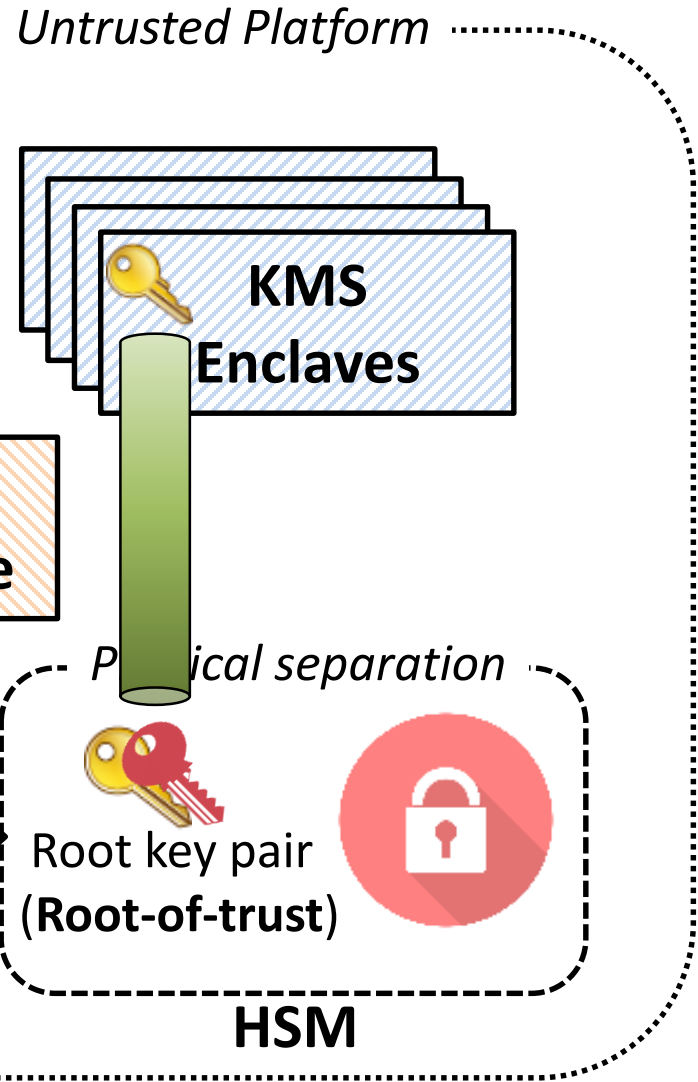
**Secure bootstrapping :**  
A fake enclave cannot build a secure channel with the HSM



**Microservices  
(KMS clients)**

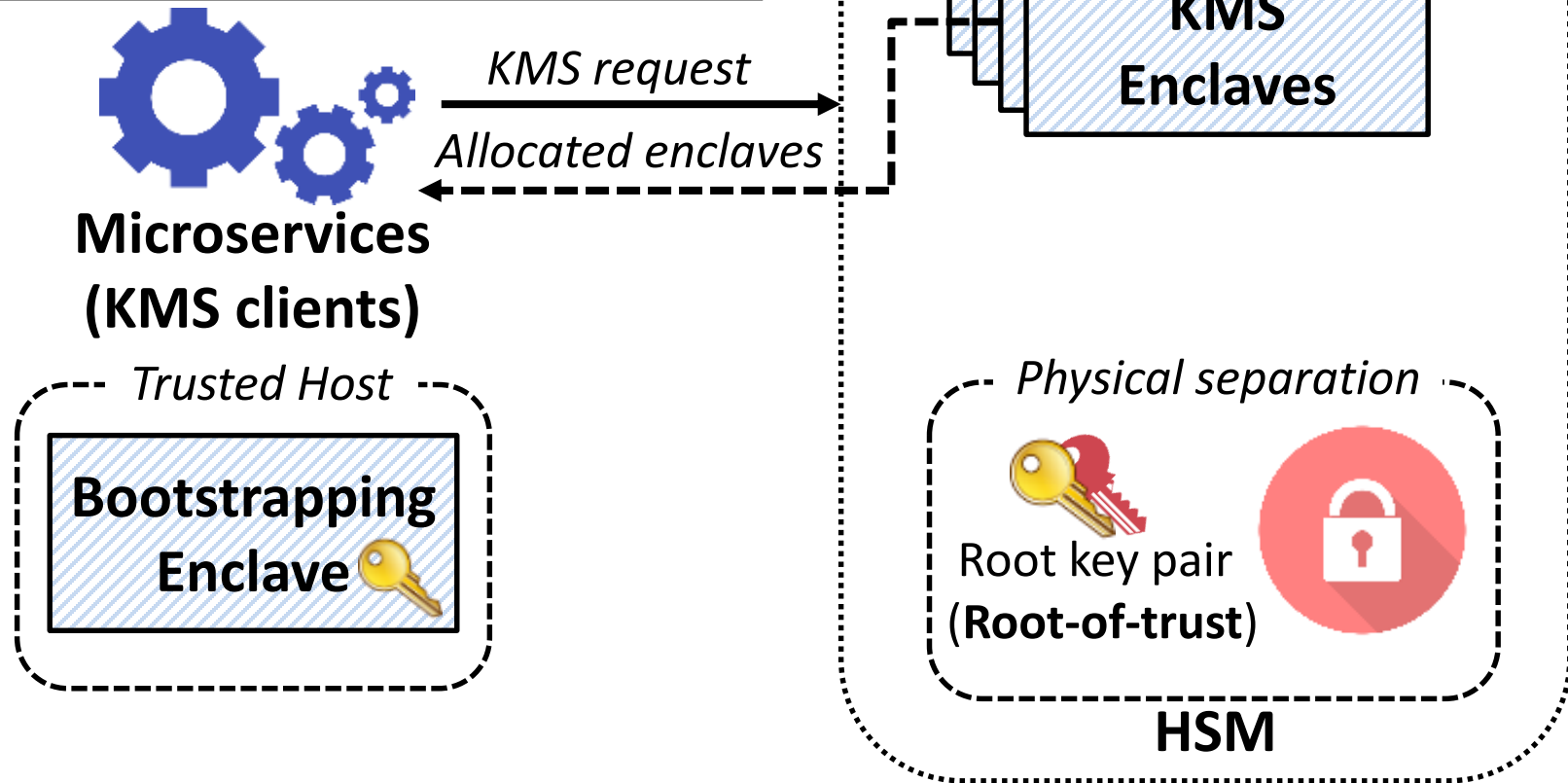


*Remote  
attestation*



# Attestation on SGX Instances

**Attestation on enclaves ① :**  
When the client first request 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.



**Microservices  
(KMS clients)**

*Trusted Host*

**Bootstrapping  
Enclave** 


*Remote  
attestation*

*Untrusted Platform*

**KMS  
Enclaves**

*Secure  
channel*

*Physical separation*

**Root key pair  
(Root-of-trust)** 

**HSM**



# Attestation on SGX Instances

## Attestation on enclaves ③ :

Also, the client performs remote attestation to verify the KMS enclave.



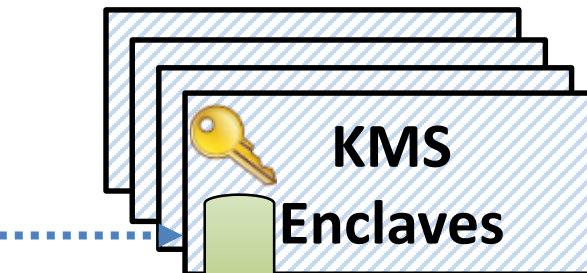
**Microservices  
(KMS clients)**

*Trusted Host*

**Bootstrapping  
Enclave** 

*Remote attestation*

*Untrusted Platform*



**KMS  
Enclaves**

*Secure  
channel*

*PKCS#11  
API calls*

*Physical separation*



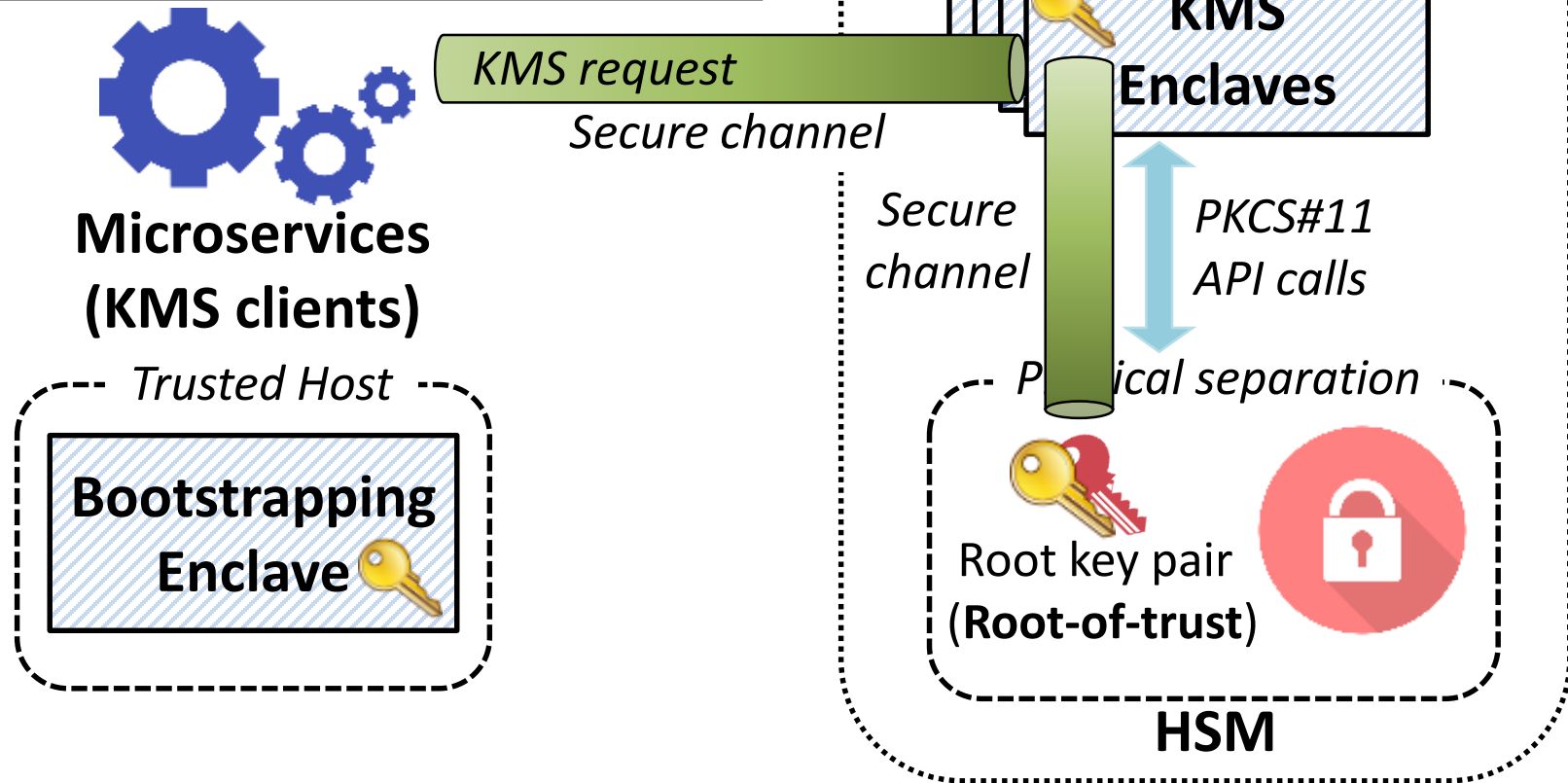
**Root key pair  
(Root-of-trust)**



**HSM**

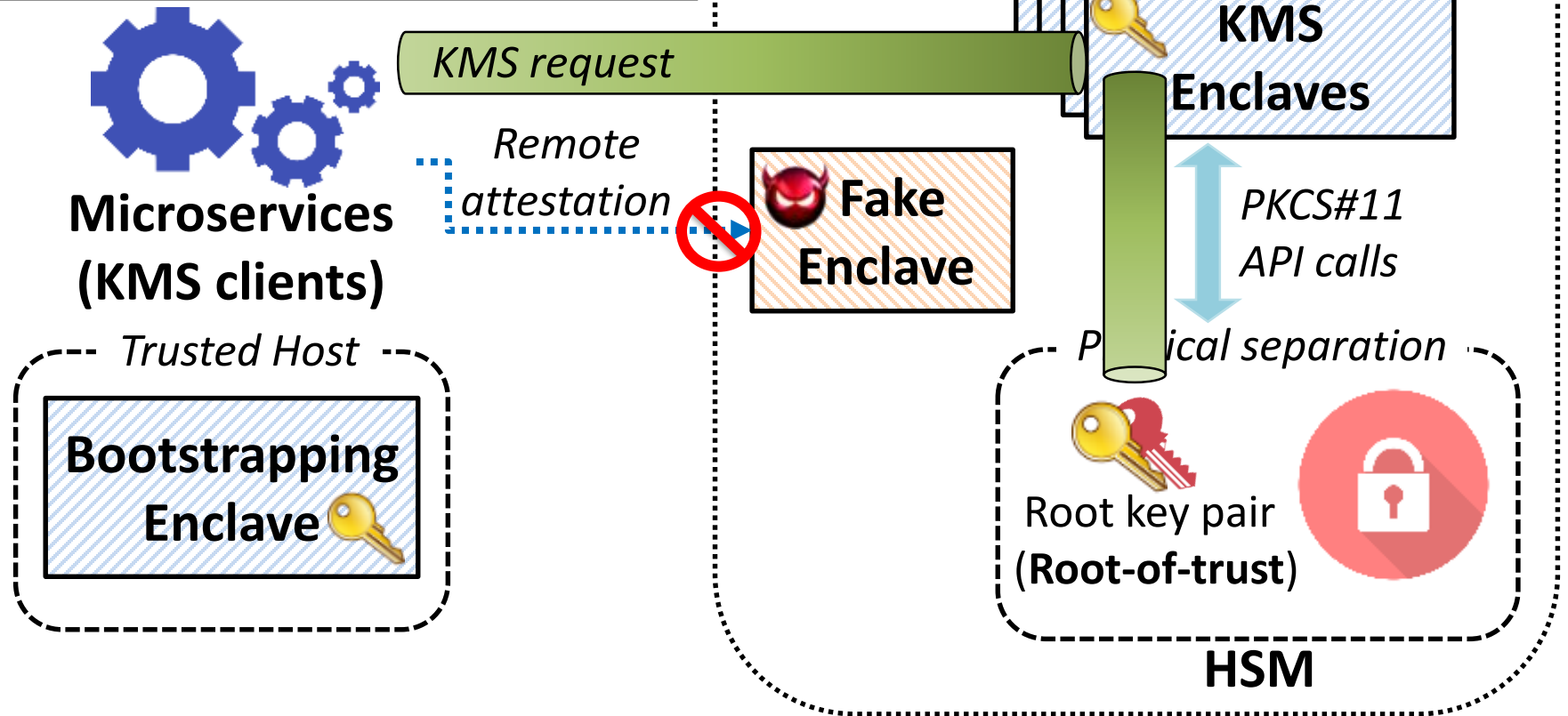
# Attestation on SGX Instances

**Attestation on enclaves ④ :**  
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



# Hierarchical Design for Scaling

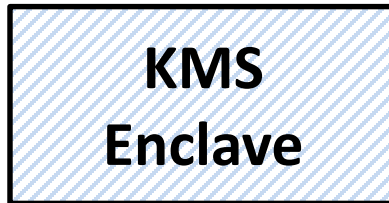
---

**KMS  
requests**

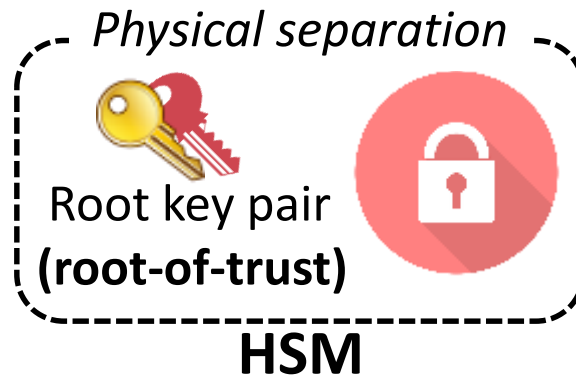


**Microservices  
(KMS clients)**

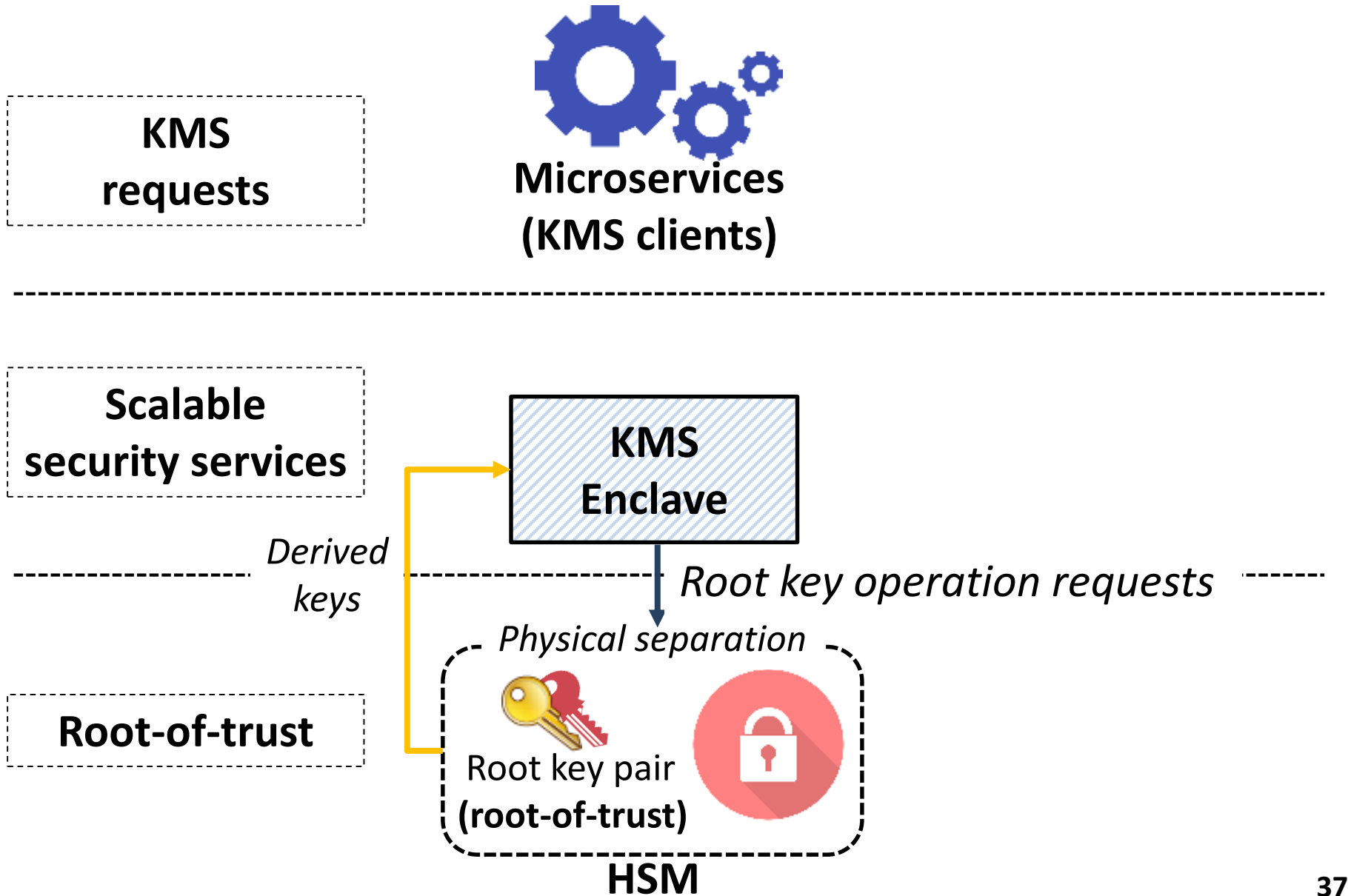
**Scalable  
security services**



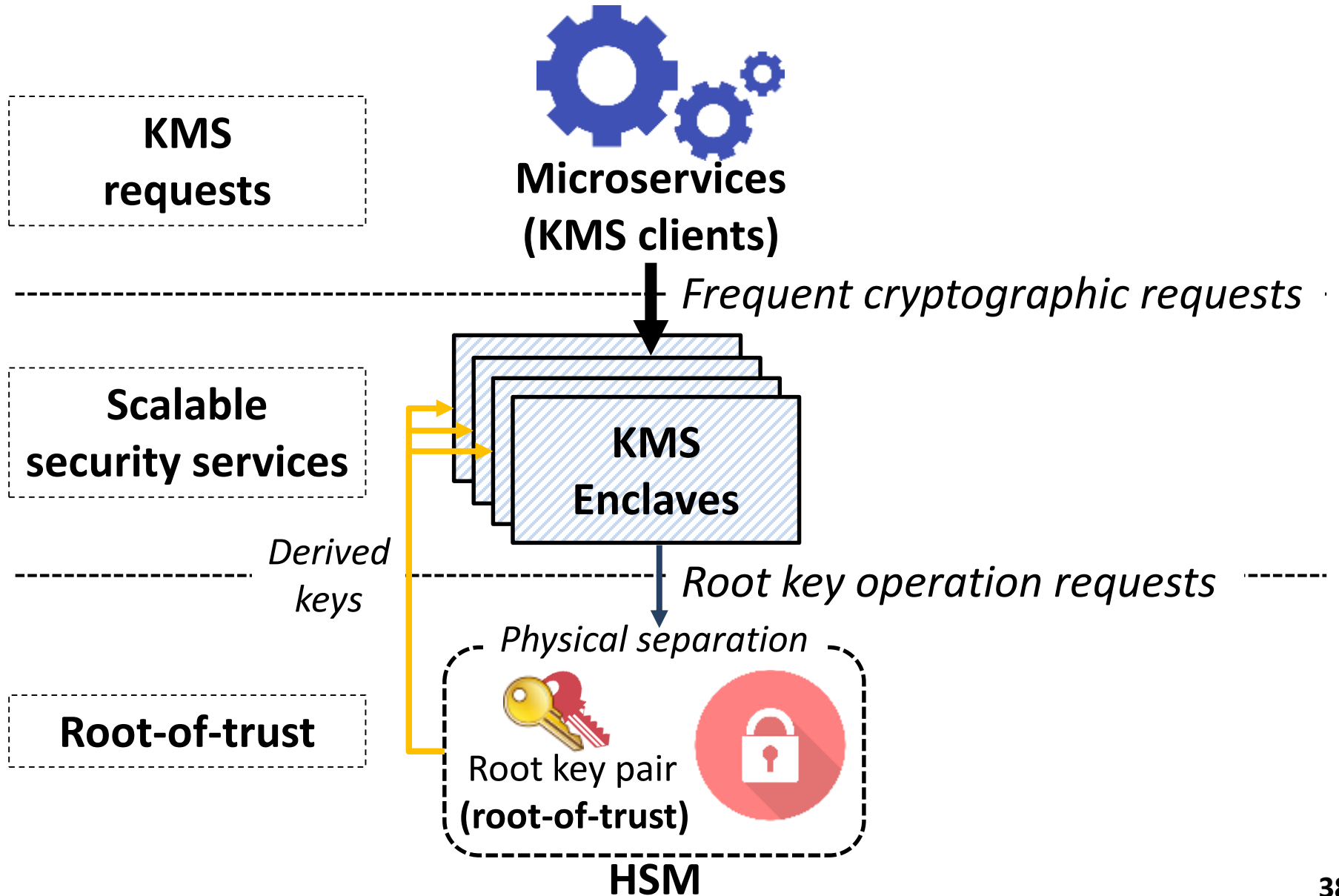
**Root-of-trust**



# Hierarchical Design for Scaling



# Hierarchical Design for Scaling

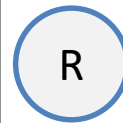


# JSON Web Token (JWT) for Microservice

---



**JWT client**



: Refresh token  
(Lifetime: few hours)



: Access token  
(Lifetime: more than a week)

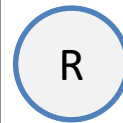
*JWT auth server*



# JSON Web Token (JWT) for Microservice



**JWT client**



: Refresh token  
(Lifetime: few hours)



: Access token  
(Lifetime: more than a week)



*Refresh token request*

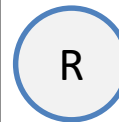
*JWT auth server*



# JSON Web Token (JWT) for Microservice



**JWT client**



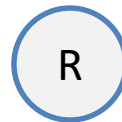
: Refresh token  
(Lifetime: few hours)



: Access token  
(Lifetime: more than a week)

↓ Refresh token request

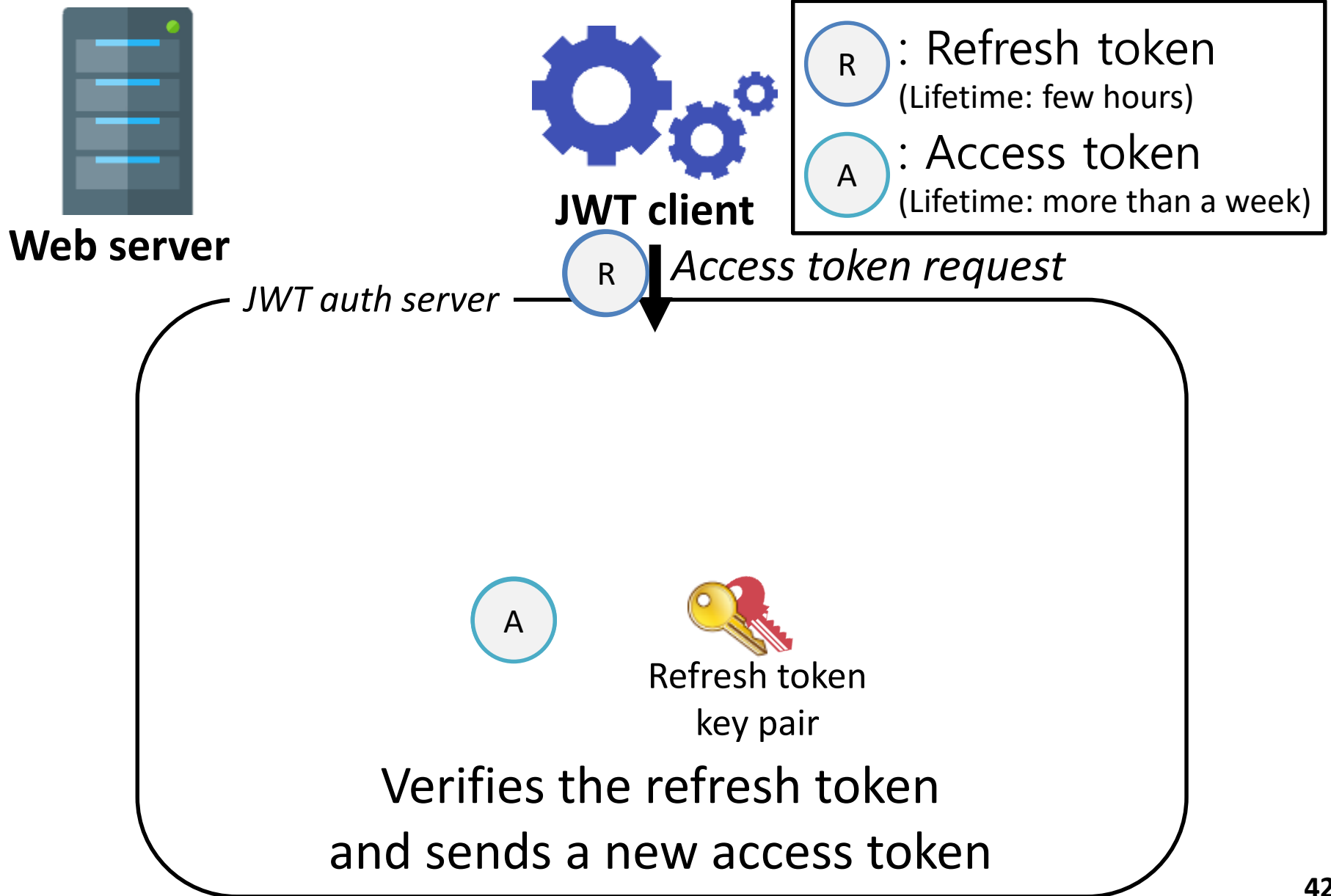
JWT auth server



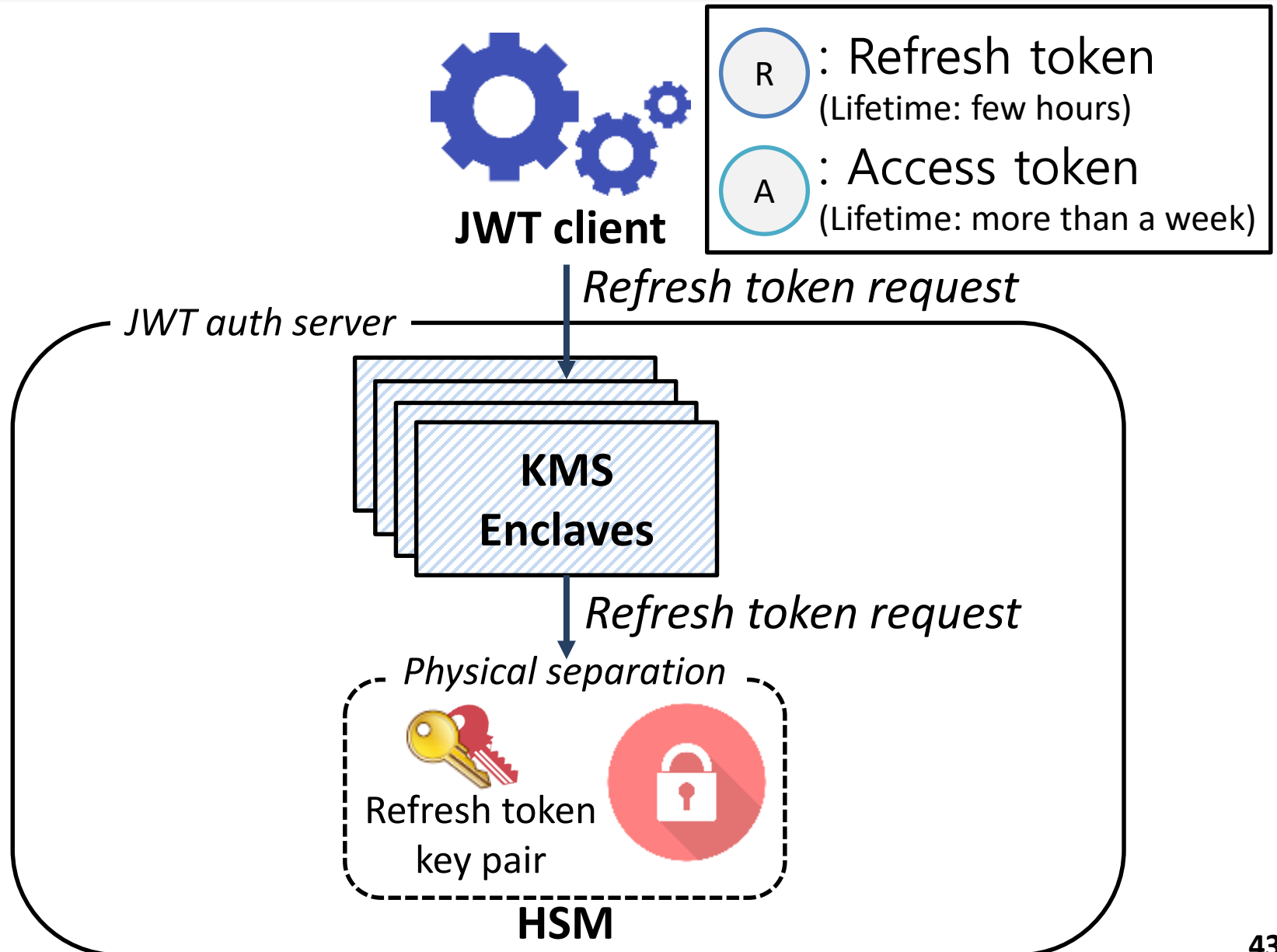
Refresh token  
key pair

Creates and signs the refresh token

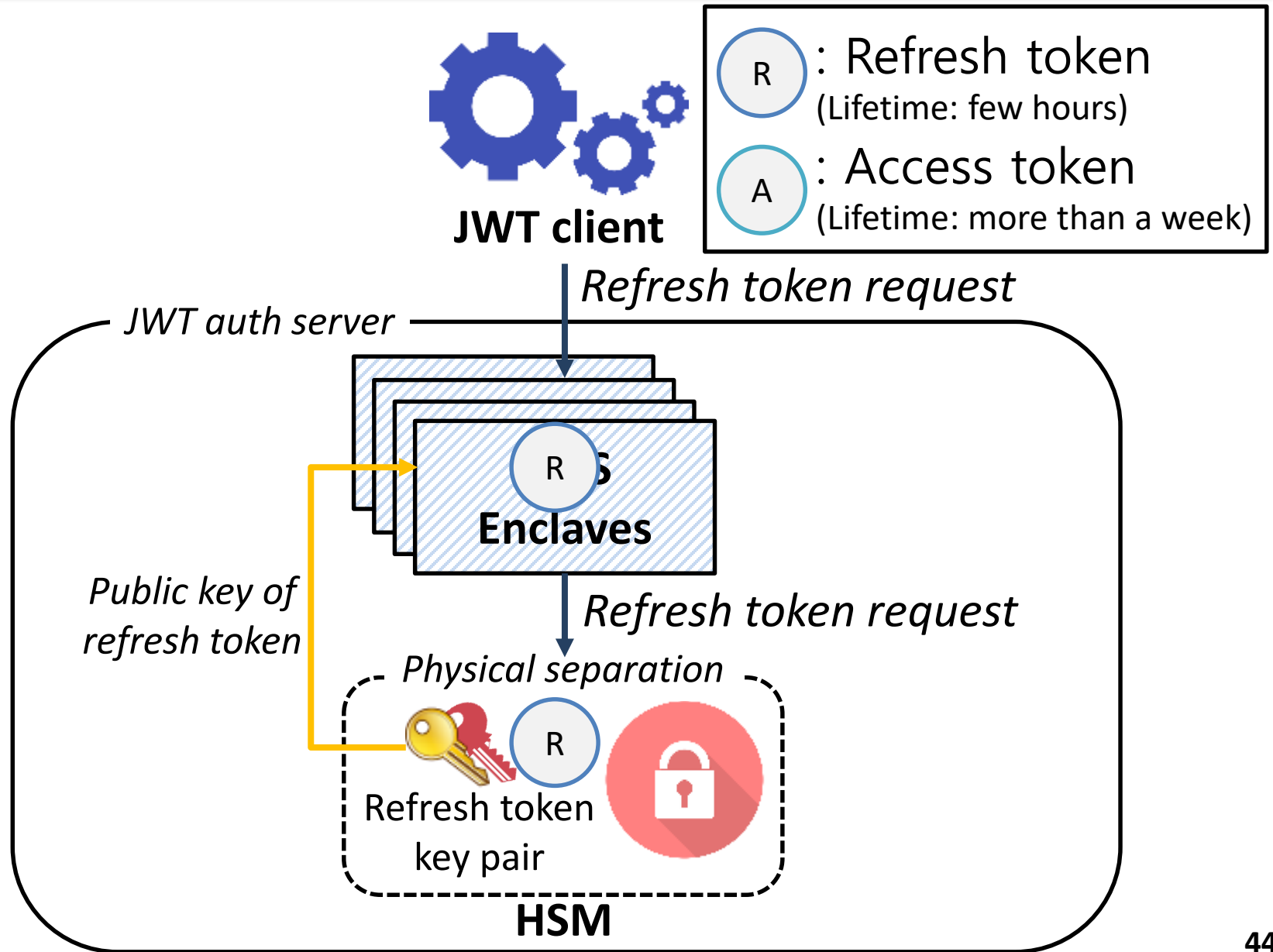
# JSON Web Token (JWT) for Microservice



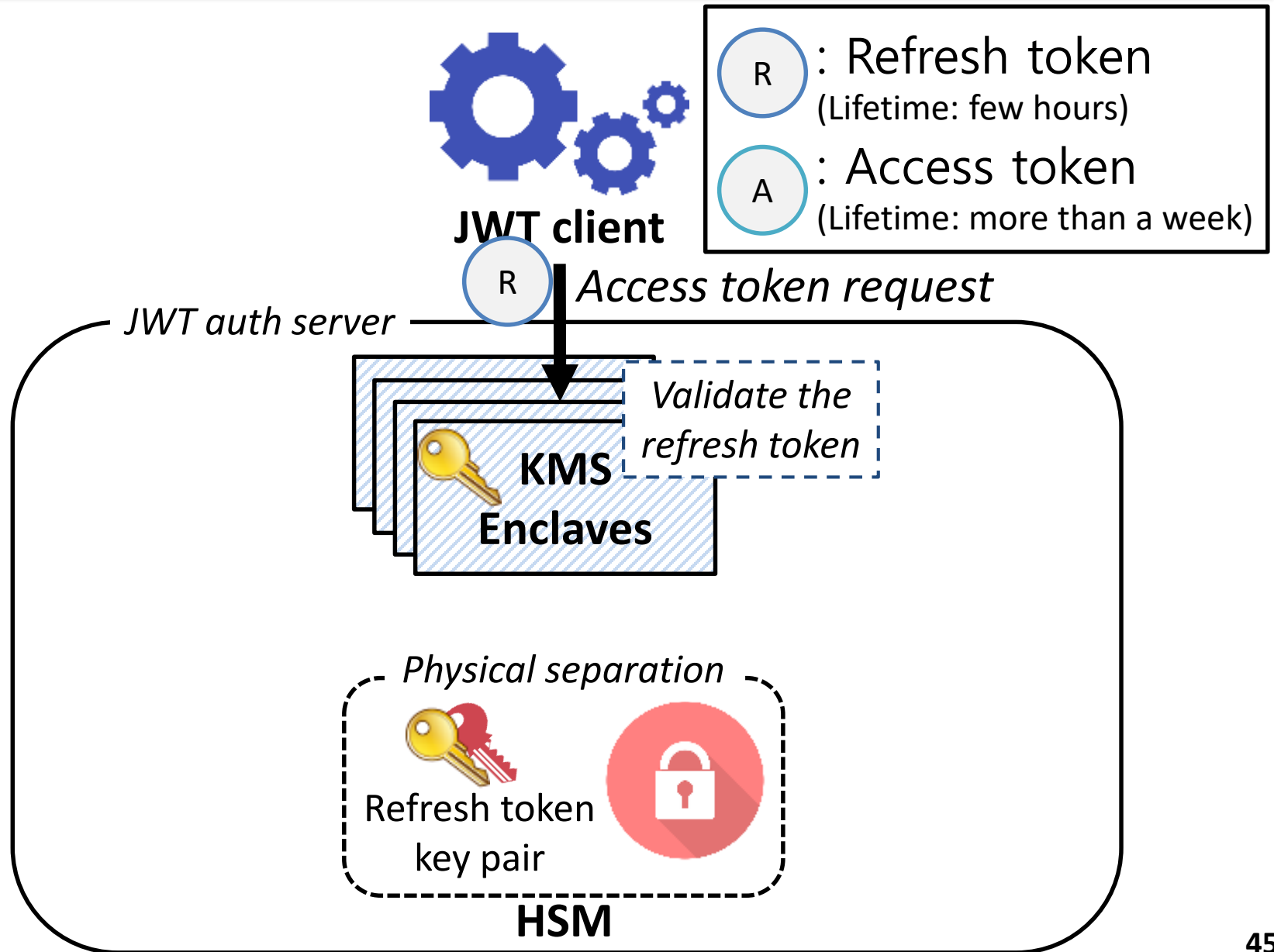
# Application Case Study : JWT Management



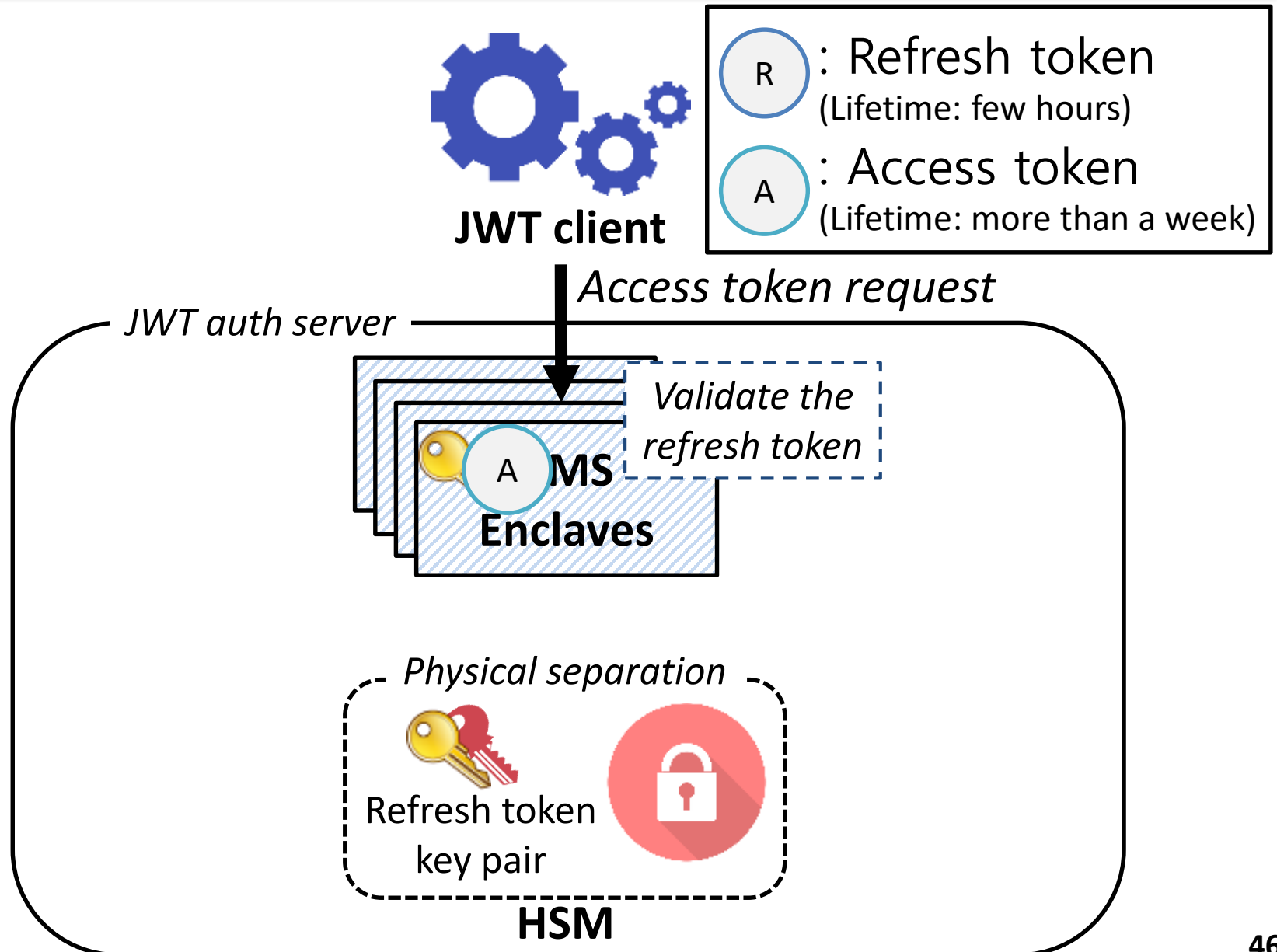
# Application Case Study : JWT Management



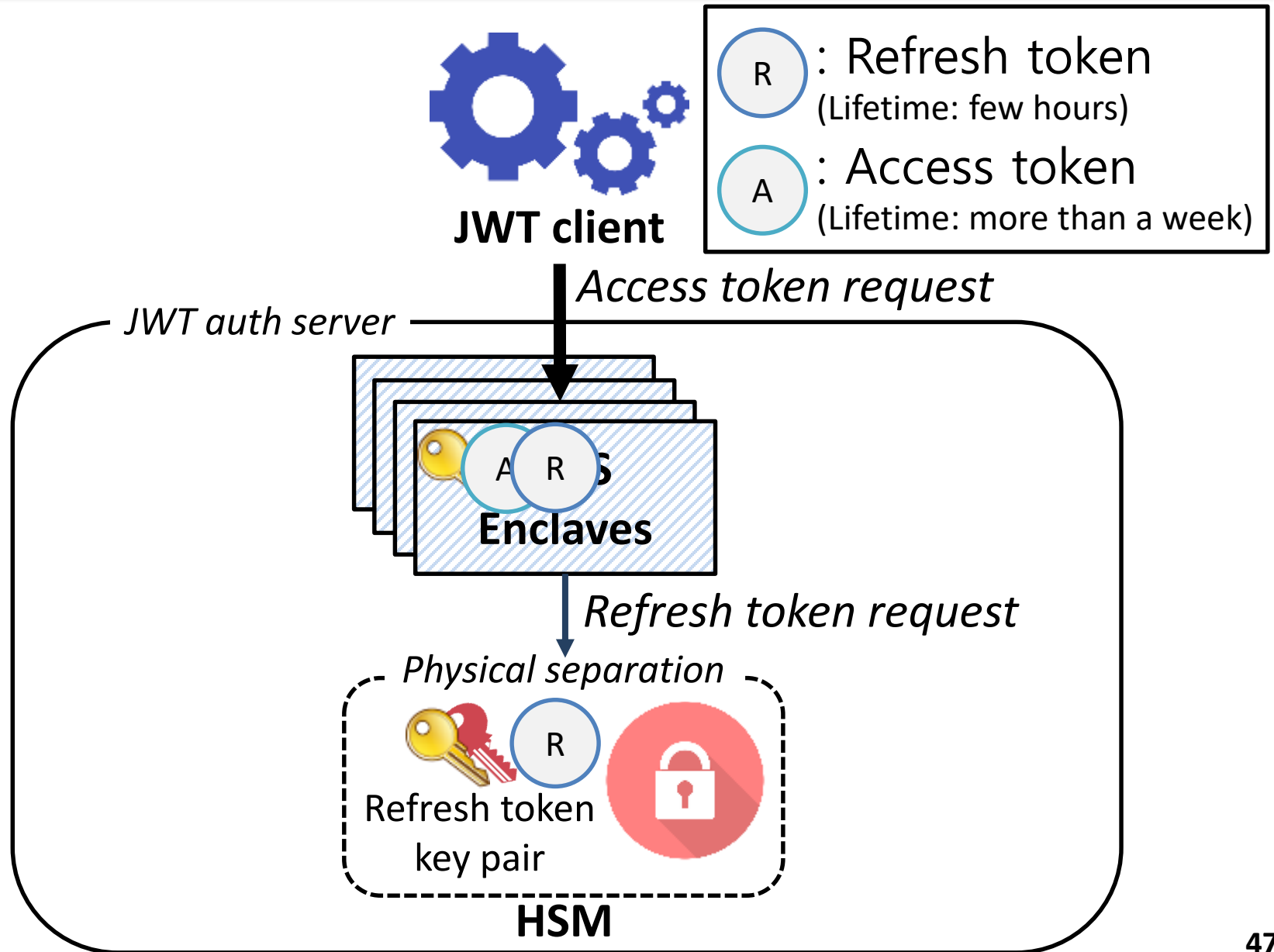
# Application Case Study : JWT Management



# Application Case Study : JWT Management



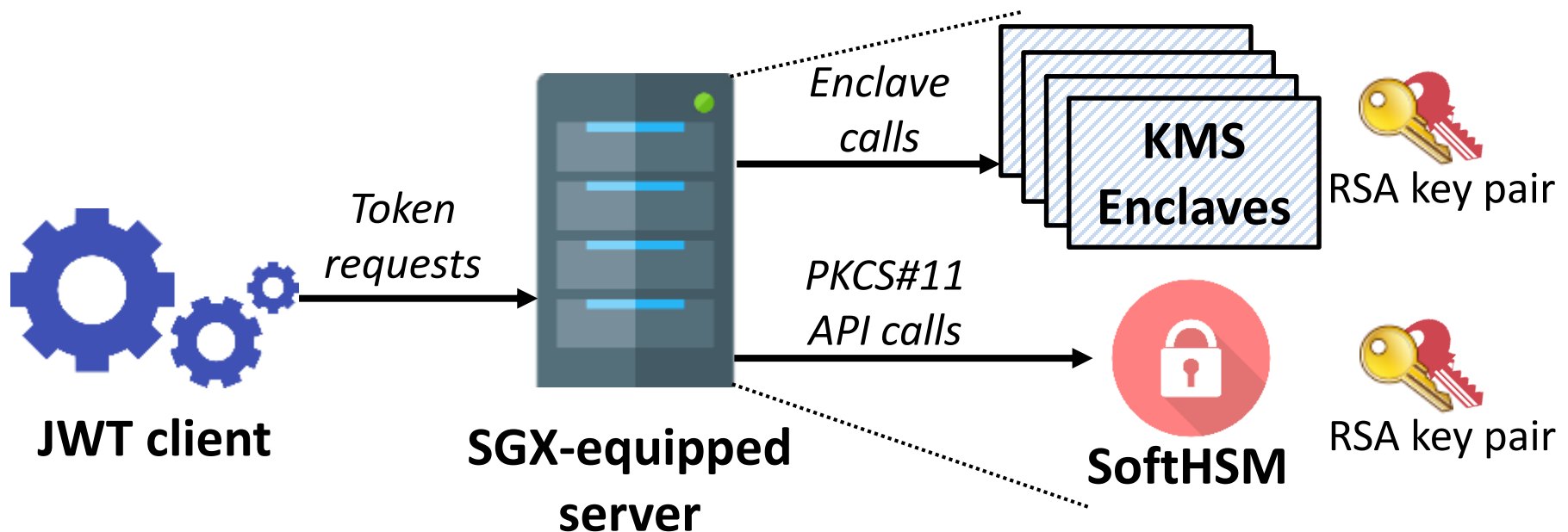
# Application Case Study : JWT Management



# Preliminary Evaluation

- **Environment setup**

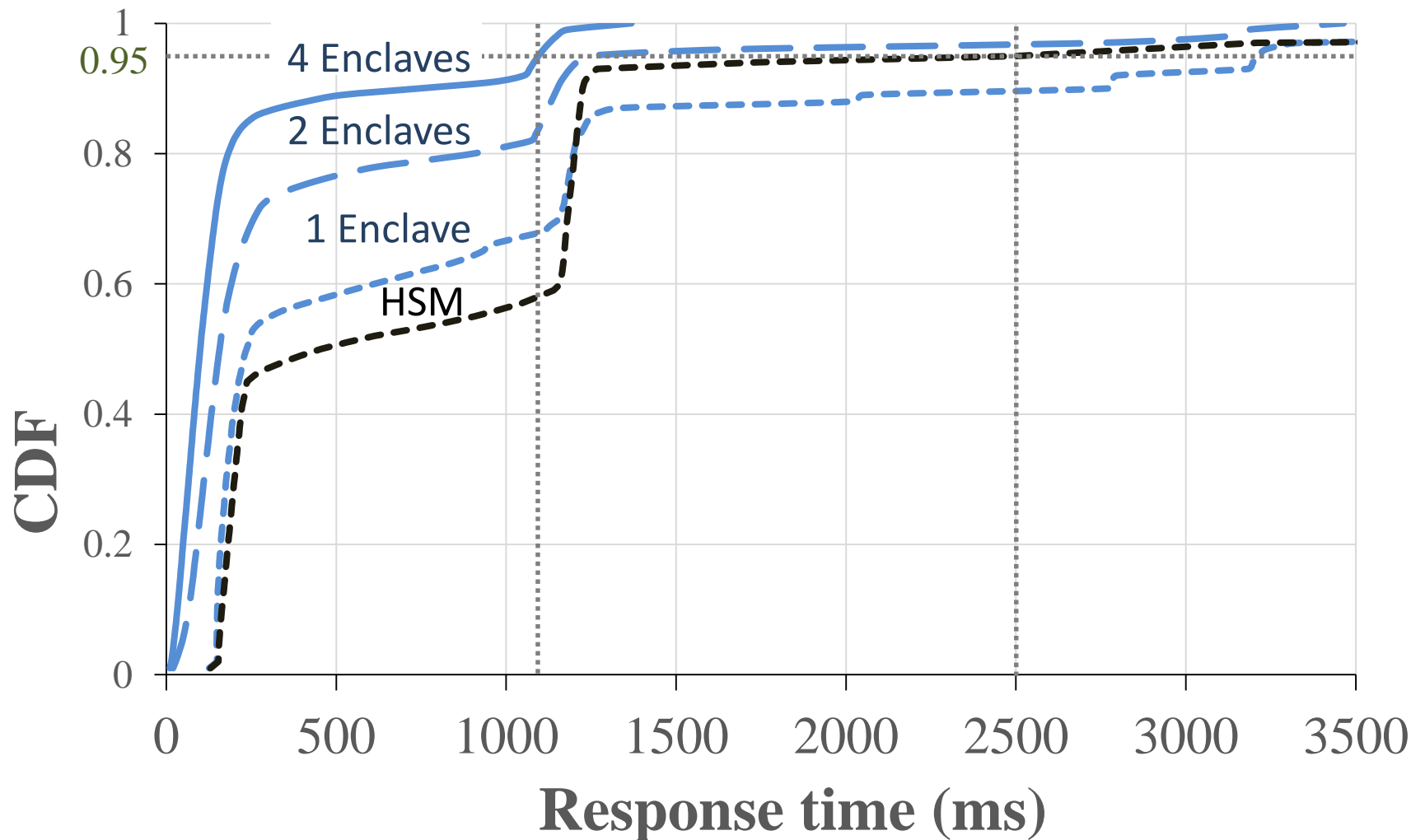
- CPU: Quad-core Intel Xeon E3-1280 v6 (SGX-enabled)
- Intel SGX Linux SDK version 2.5
- We use SoftHSM to emulate an HSM device.
- Each enclave and HSM performs the same SHA-256 with RSA-2048 signing





# Preliminary Evaluation: Latency Improvement

- Scaling out KMS enclaves for latency improvement



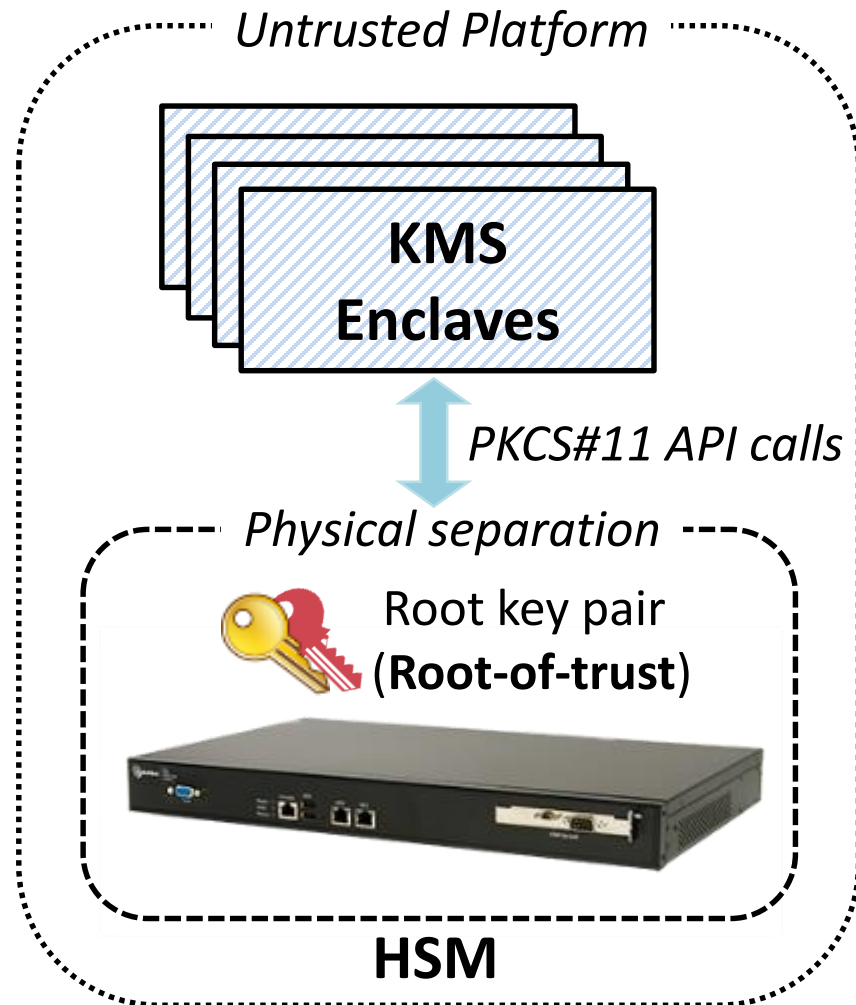
# Preliminary Evaluation: Cost-effective Scaling

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

\*tps = transactions per second

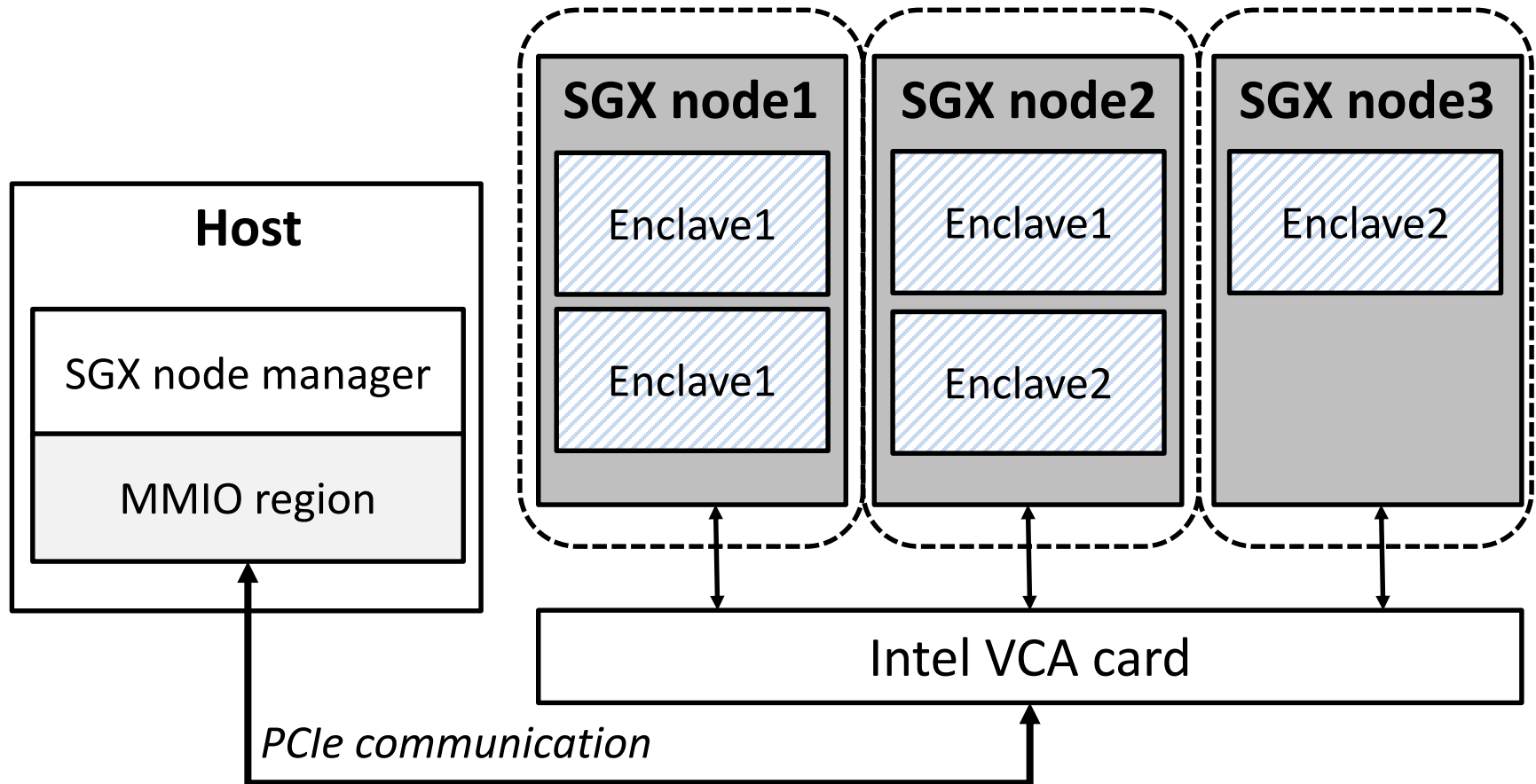
# Future work

- Evaluation with a real HSM device



# Future work

- Physical separation by Intel VCA (SGX card)



# 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**