KubeCon | CloudNativeCon

Europe 2019

Securing Kubernetes with Trusted Platform Module

Who are we, anyway?

Alex Tcherniakhovski

alextc@google.com

Security Engineer, Google Kubernetes Engine

Andrew Lytvynov

awly@google.com

Software Engineer, Google Kubernetes Engine

What's a Trusted Platform Module (TPM)?

- Crypto coprocessor
- Hardware or software
- Cheap, low-powered
- Spec designed by Trusted Computing Group (TCG)
- Spec versions 1.2 and 2.0

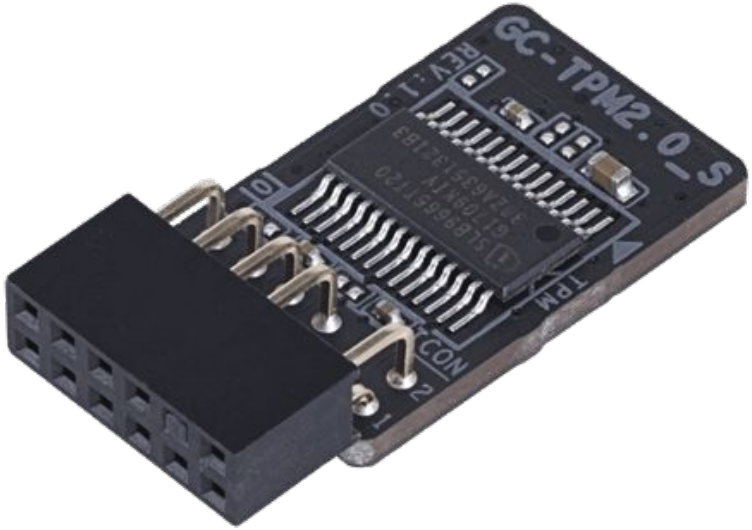


Image source: https://www.gigaparts.com/gigabyte-trusted-platform-module-2-0-gc-tpm2-0-s.html

This talk

Goals:

- sample of TPM capabilities
- match k8s security challenges to TPM capabilities
- fuel exploration by users and sig-auth

Tricky security problems

Node trust bootstrap

- provide kubelet with credentials
- fully automated
- periodic rotation
- protect during Pod or Node compromise

Tricky security problems

First secret problem

- encrypt Secrets at rest
- store encryption key
- protect encryption key

Tricky security problems

Tamper-evident audit logging

- audit access to Secrets
- cryptographically-signed log
- verifiable log
- tamper-evident
- even with full master compromise

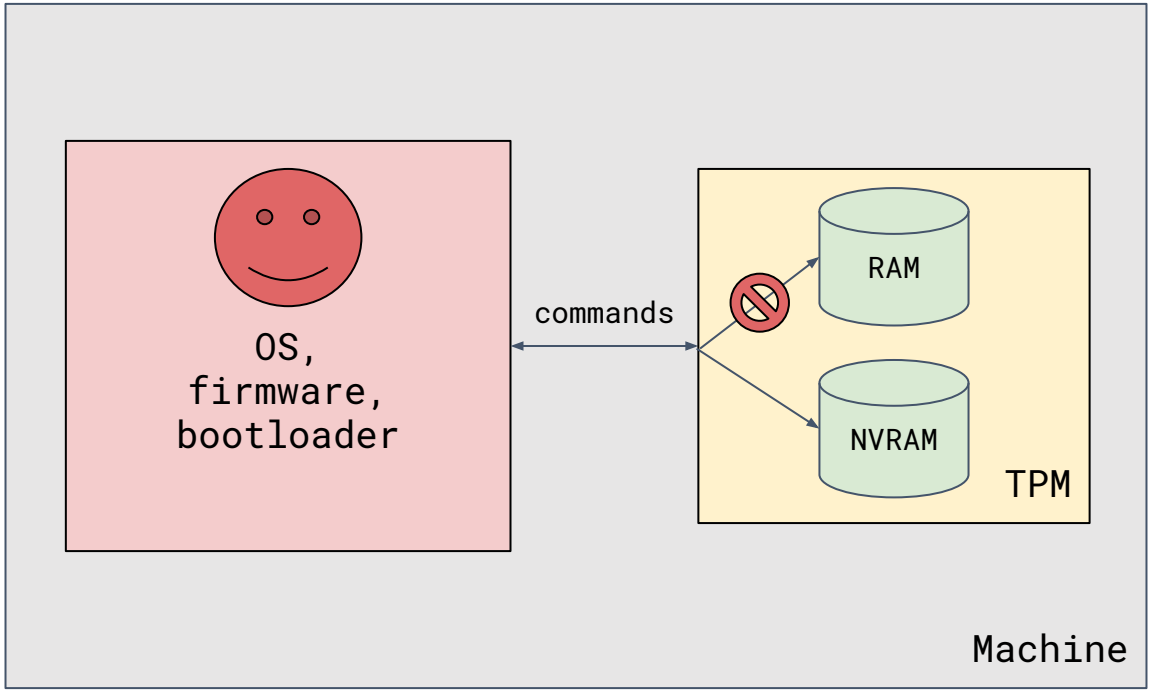
Agenda

1. Trusted Platform Module (TPM) crash course
2. Node trust bootstrap
3. First secret problem
4. Cryptographically protected audit log

1. Trusted Platform Module (TPM) crash course

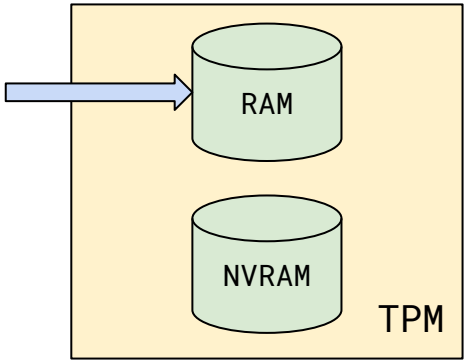
Classic use cases:

- Platform integrity
  - “is this corp machine in an expected state?”
- Disk encryption
  - BitLocker, dm-crypt, etc
  - protect encryption keys
  - verify integrity of bootloader/kernel/drivers


OS,
firmware,
bootloader
commands
RAM
NVRAM
TPM
Machine

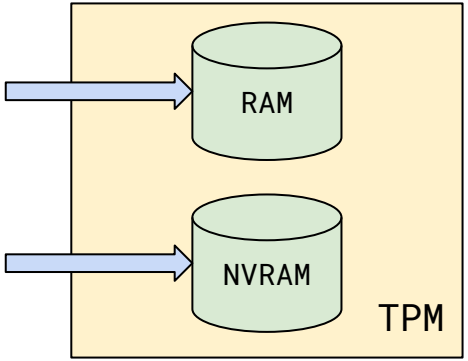
TPM keys

- RSA or ECDSA
- Encryption or signing
- Symmetric or asymmetric
- TPM-bound
  - no exfiltration
  - can export from TPM, but only encrypted
- Used via specialized commands

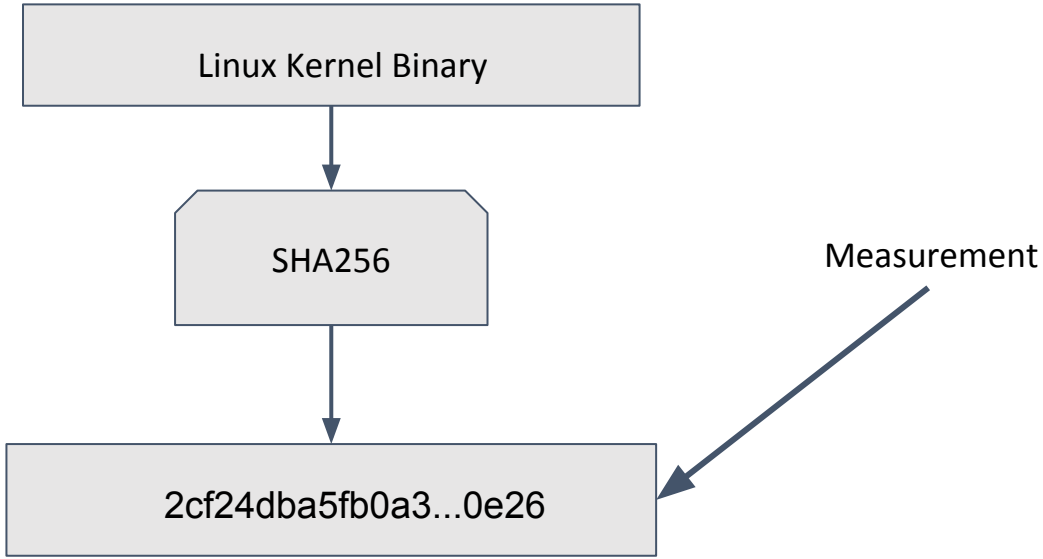


Endorsement Key (EK)

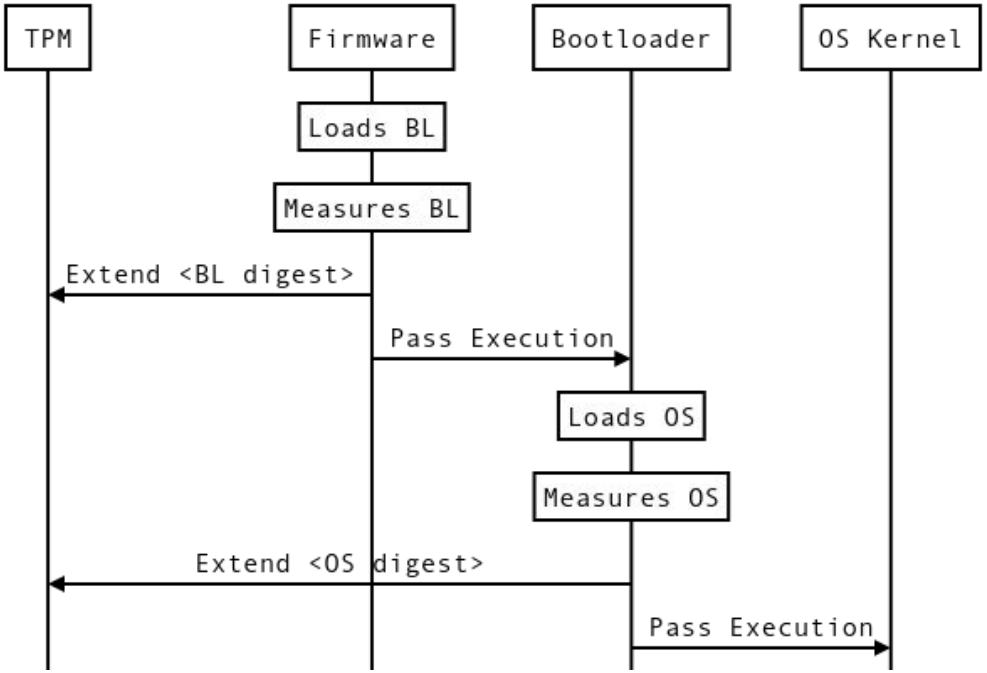
- Key baked into TPM
- Certificate signed by TPM vendor in NVRAM
- Used as machine identity



Measurements/Digests



Measured Boot



source https://google.github.io/tpm-js

Platform Configuration Registers (PCRs)

PCR0 <-- Firmware

...

PCR4 <-- UEFI

PCR5 <-- Partition Table

PCR7 <-- Secure Boot Policy

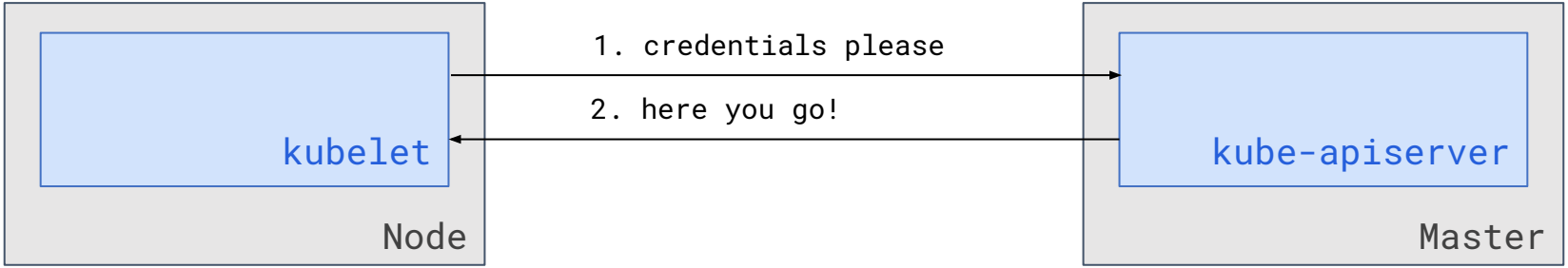
Extend Operation

PCR = HASH (PCR || datanew)

A whole lot more...

- RNG
- key hierarchies
- authorization policies
- certification
- dictionary attack protection
- command audit
- external/transferable keys

2. Node trust bootstrap


1. credentials please
2. here you go!
kubelet
Node
kube-apiserver
Master

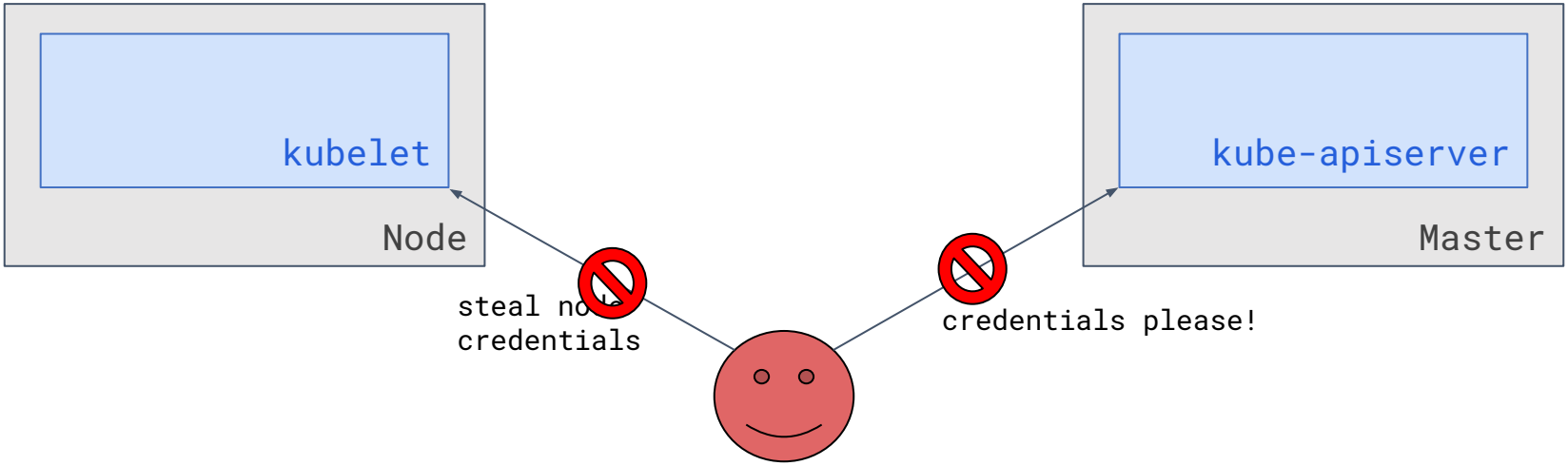
Threat model

Attacker has:

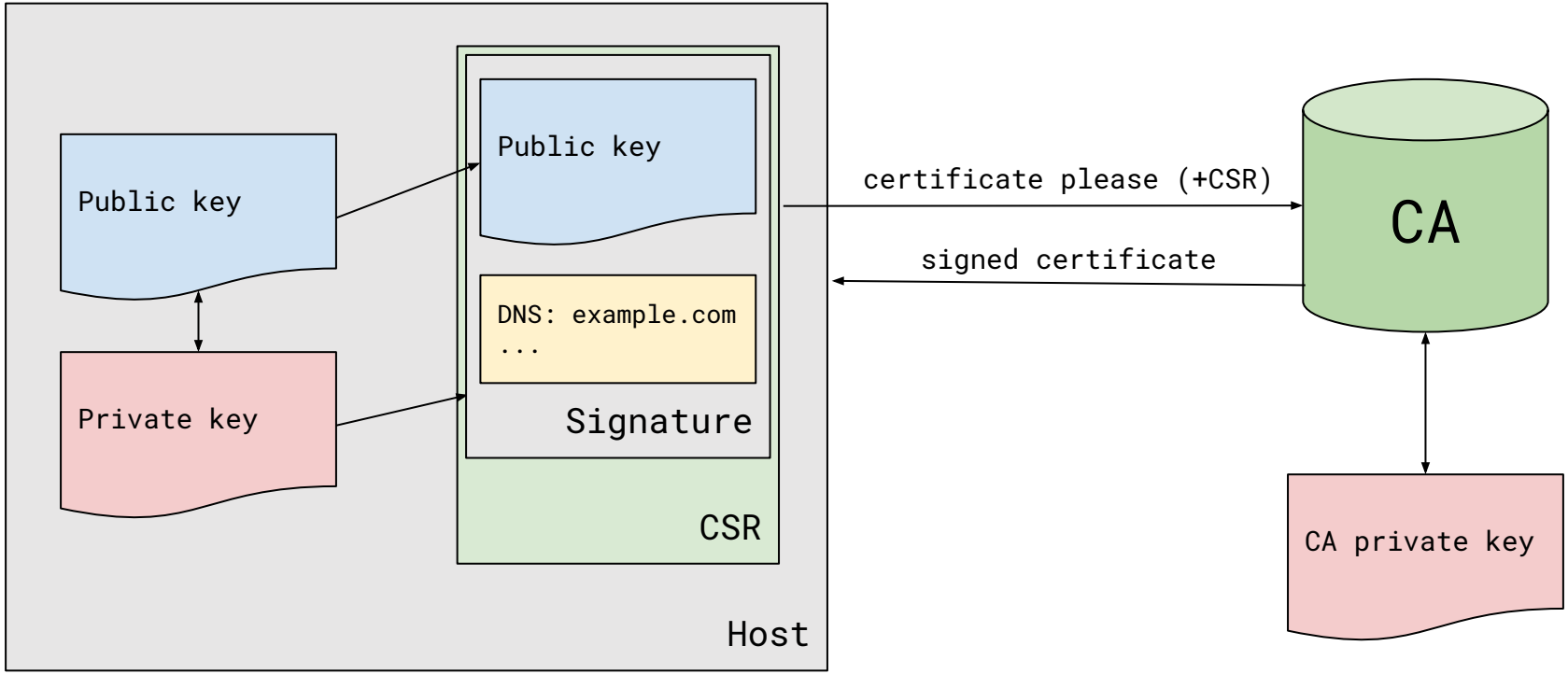
- compromised Pod
- compromised Node

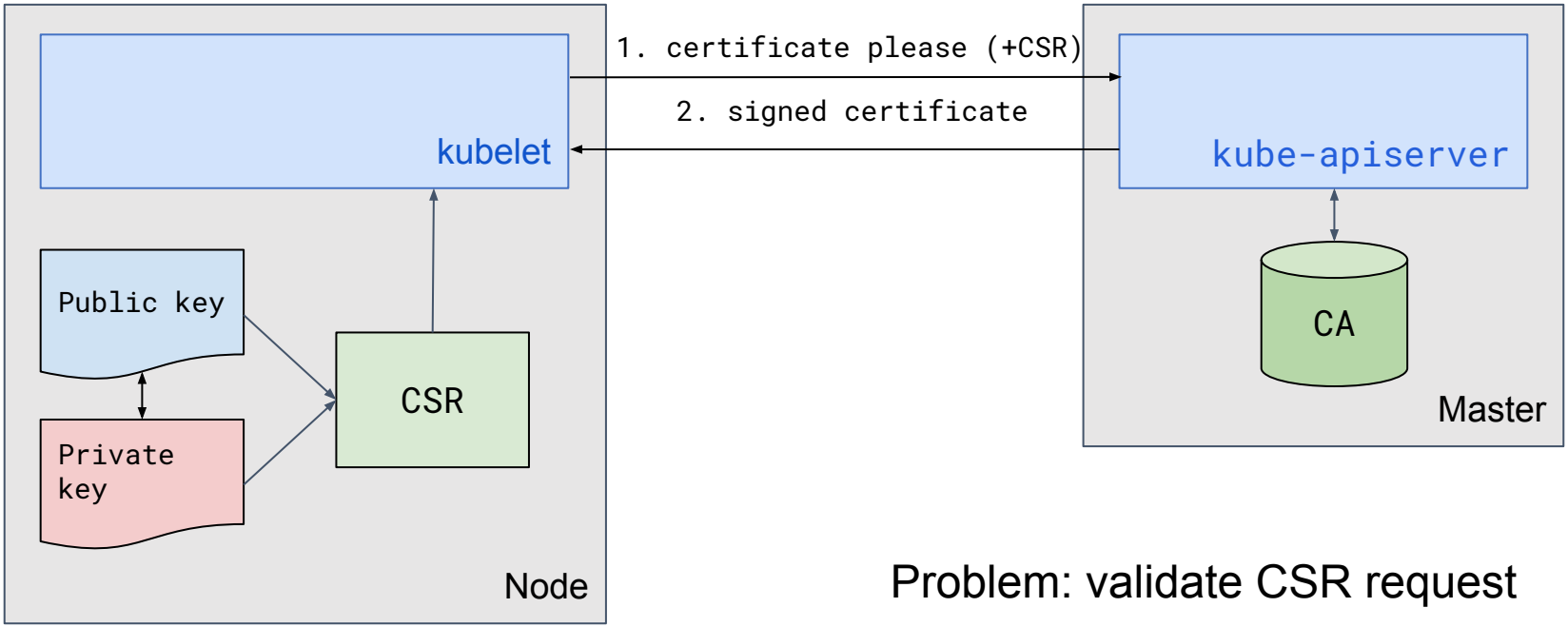
Attacker wants:

- exfiltrate application configs
- exfiltrate application Secrets
- persist access

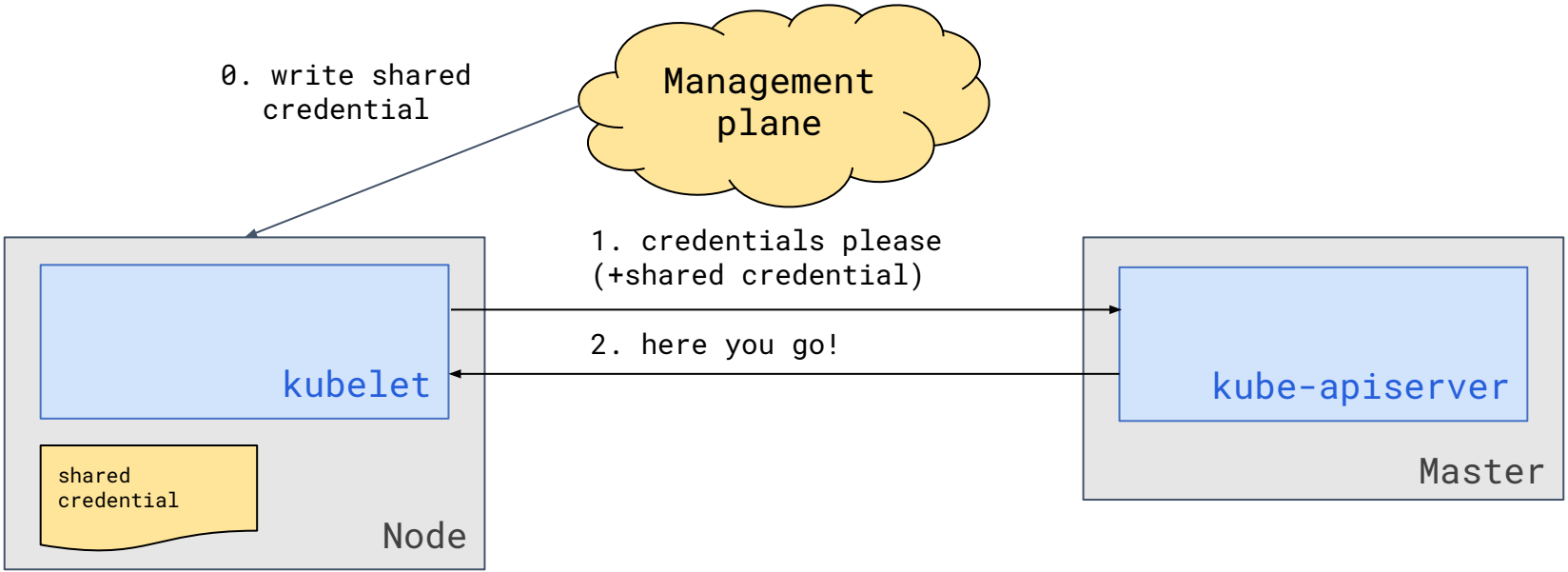

kubelet
Node
kube-apiserver
Master
steal node
credentials
credentials please!

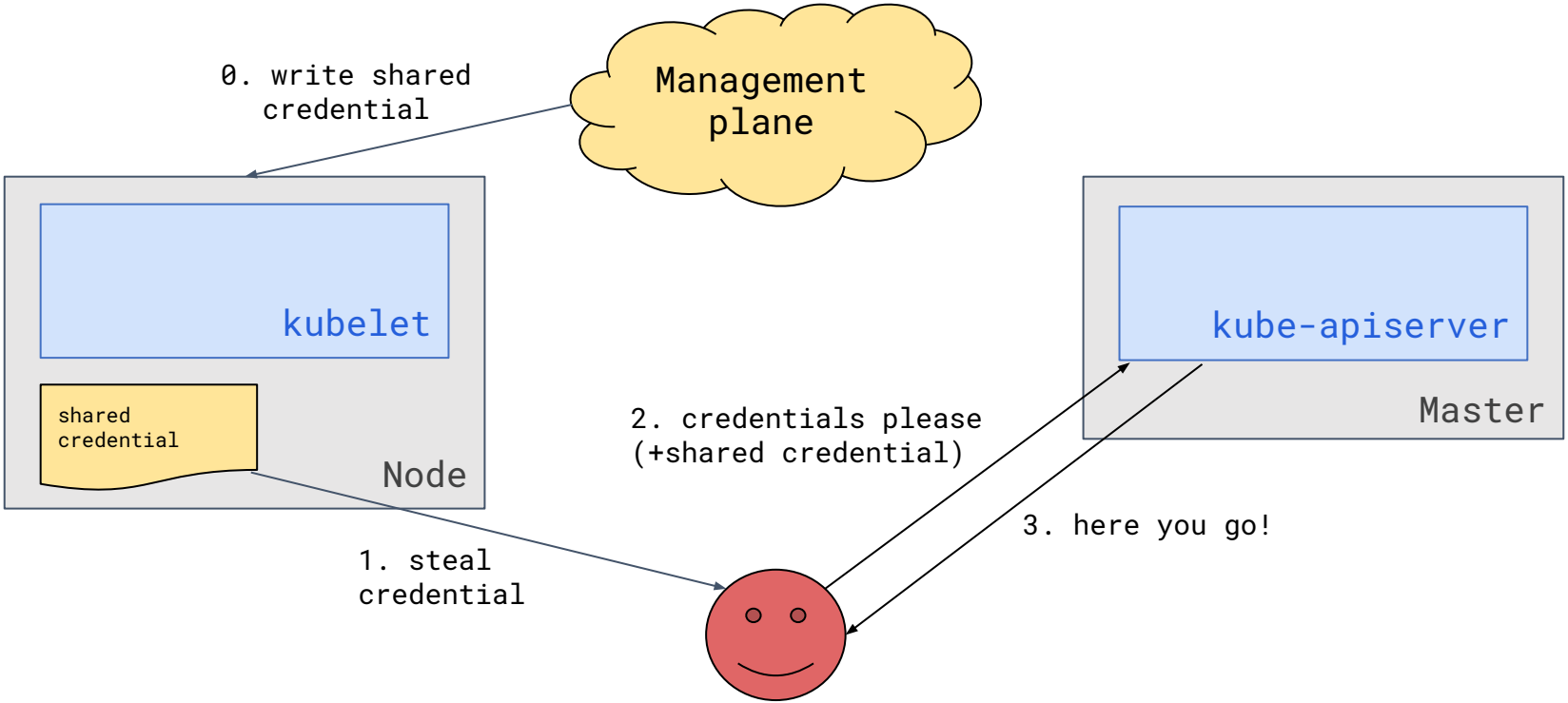
Enter X.509 CSRs and Certificates

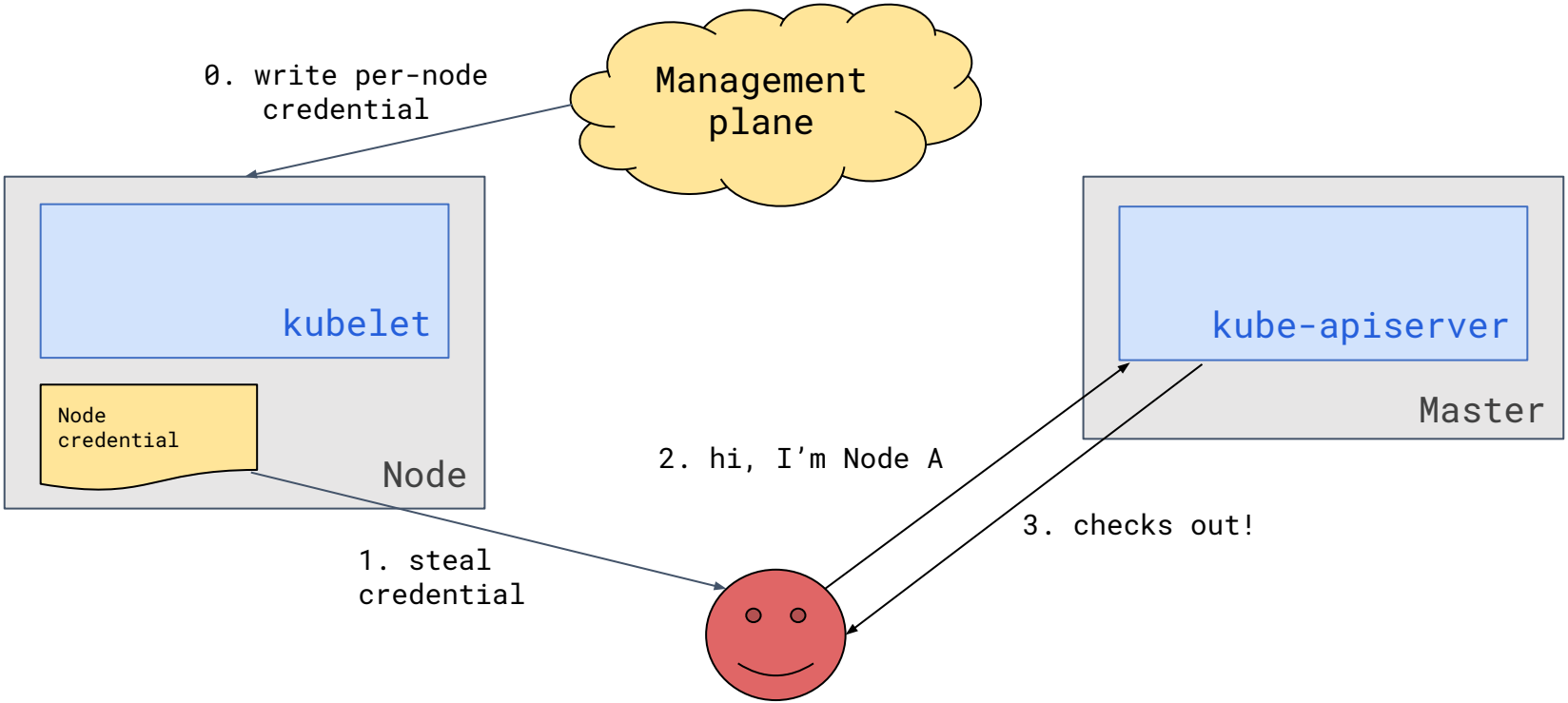

Public key
Private key
Public key
DNS: example.com
...
Signature
CSR
Host
certificate please (+CSR)
signed certificate
CA
CA private key



Problem: validate CSR request

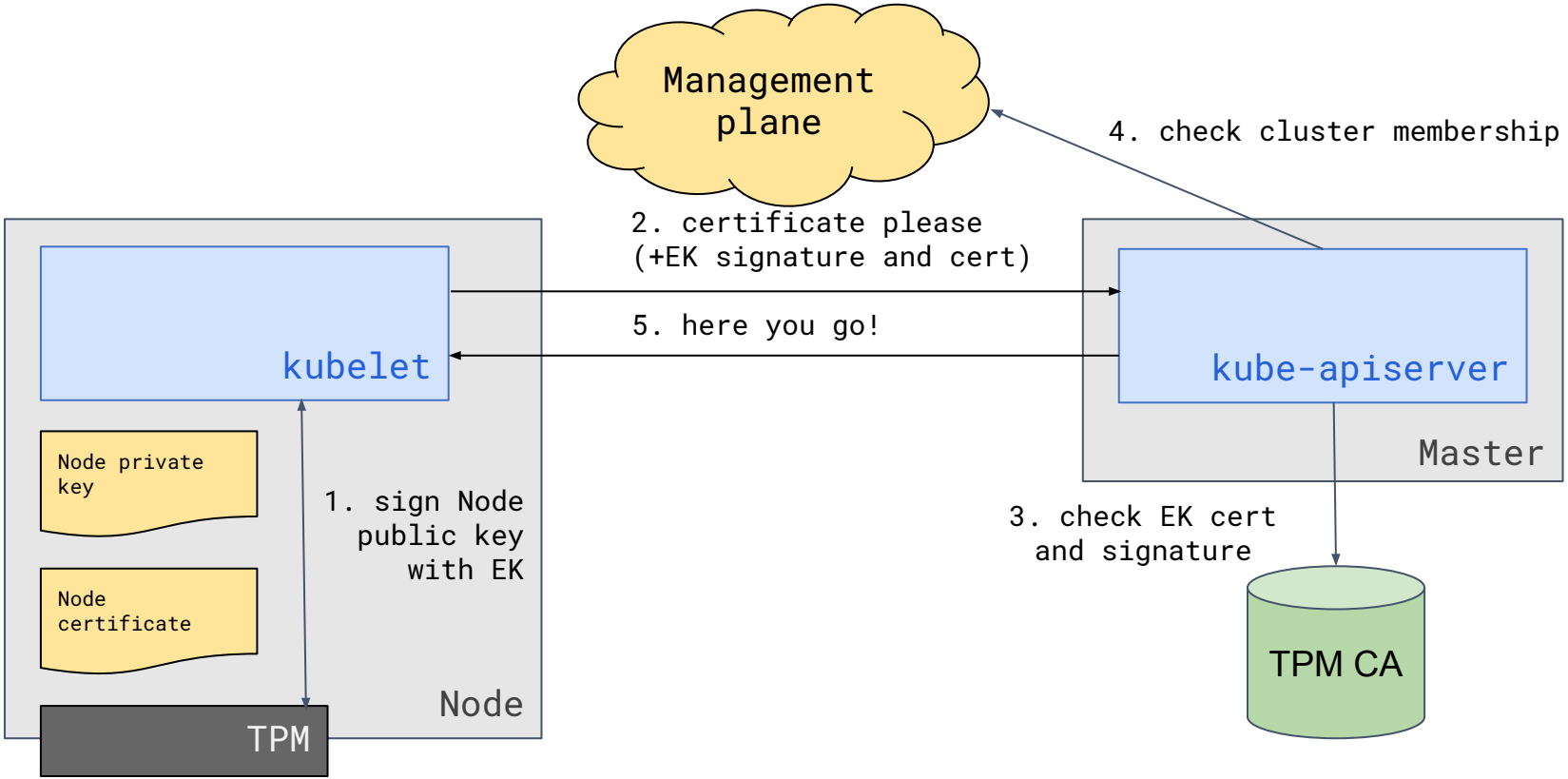

0. write shared
credential
Management
plane
1. credentials please
(+shared credential)
2. here you go!
kubelet
shared
credential
Node
kube-apiserver
Master


0. write shared credential
Management plane
kubelet
shared credential
Node
kube-apiserver
Master
2. credentials please (+shared credential)
3. here you go!
1. steal credential


0. write per-node credential
Management plane
kubelet
Node credential
Node
kube-apiserver
Master
2. hi, I'm Node A
3. checks out!
1. steal credential

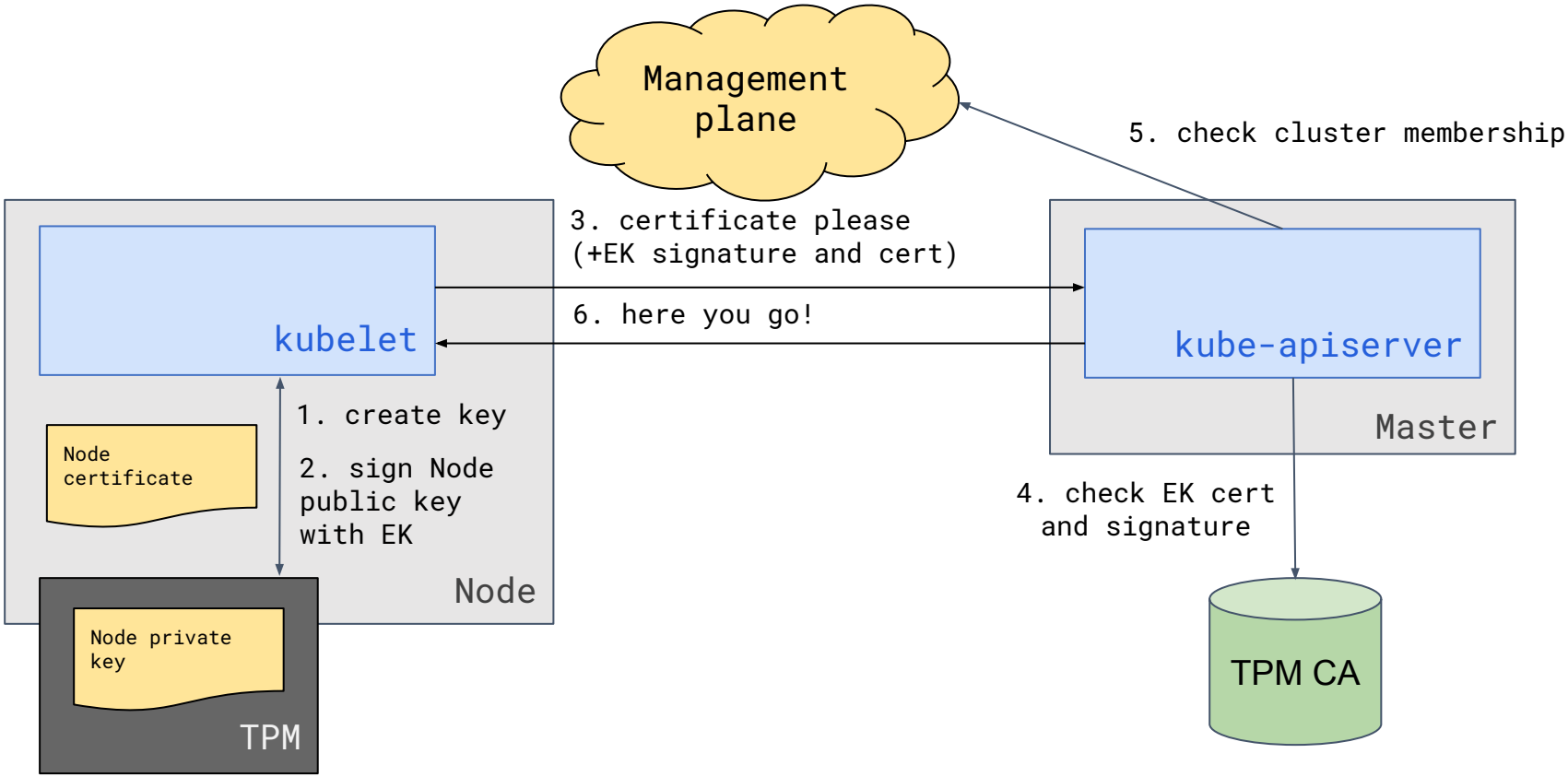
Let’s use a TPM!

EK as proof of machine identity


Management plane
4. check cluster membership
2. certificate please
(+EK signature and cert)
5. here you go!
kubelet
kube-apiserver
Master
Node private key
Node certificate
1. sign Node public key with EK
3. check EK cert and signature
TPM CA
Node
TPM

But what about exfiltration of the Node credential after provisioning?

Put it in a TPM!


Management plane
5. check cluster membership
3. certificate please
(+EK signature and cert)
6. here you go!
kubelet
kube-apiserver
Master
1. create key
2. sign Node public key with EK
Node certificate
Node
Node private key
TPM
4. check EK cert and signature
TPM CA

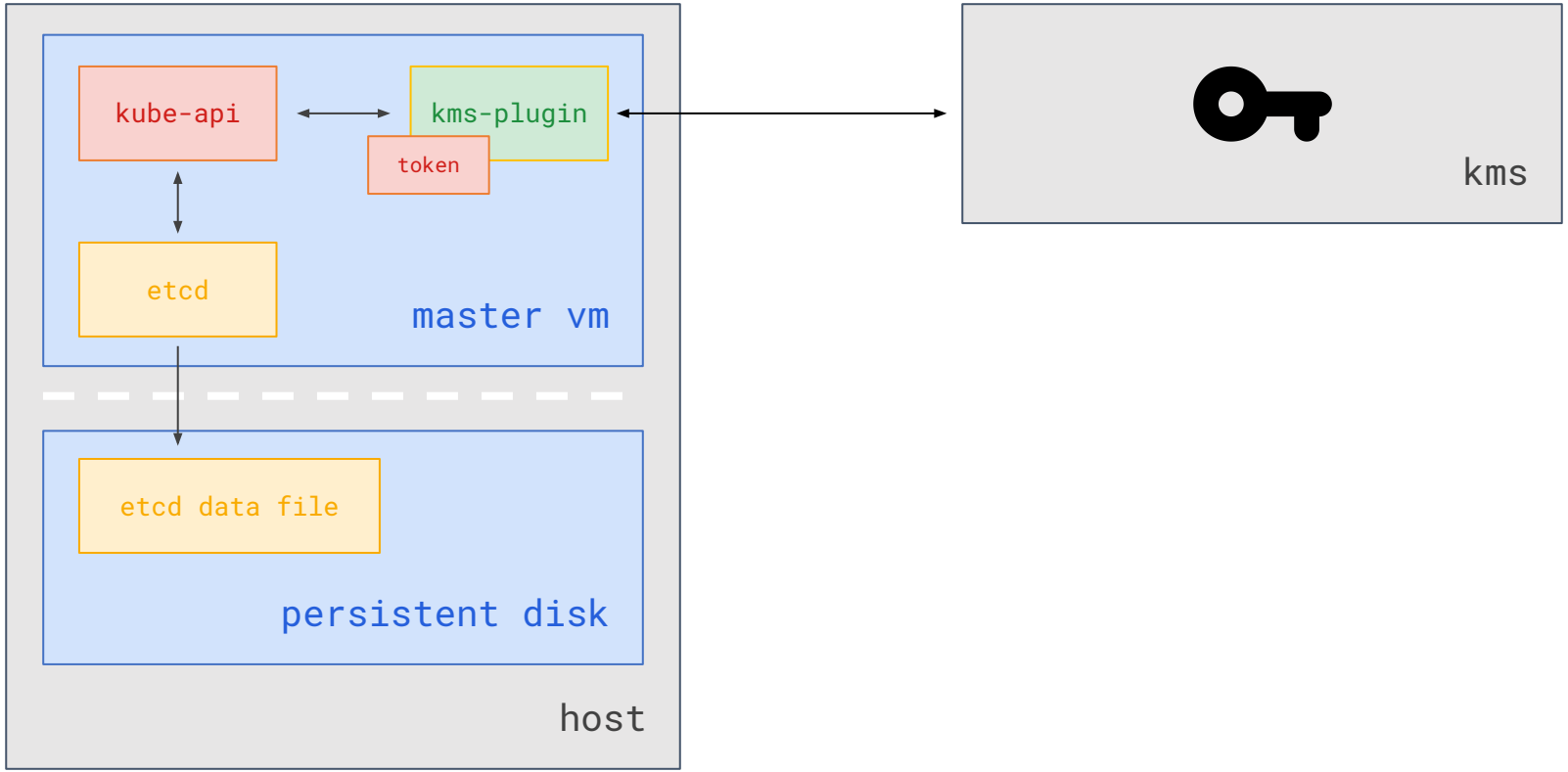
Not 100% solution

Attacker can still use Node credential via RCE on the Node.

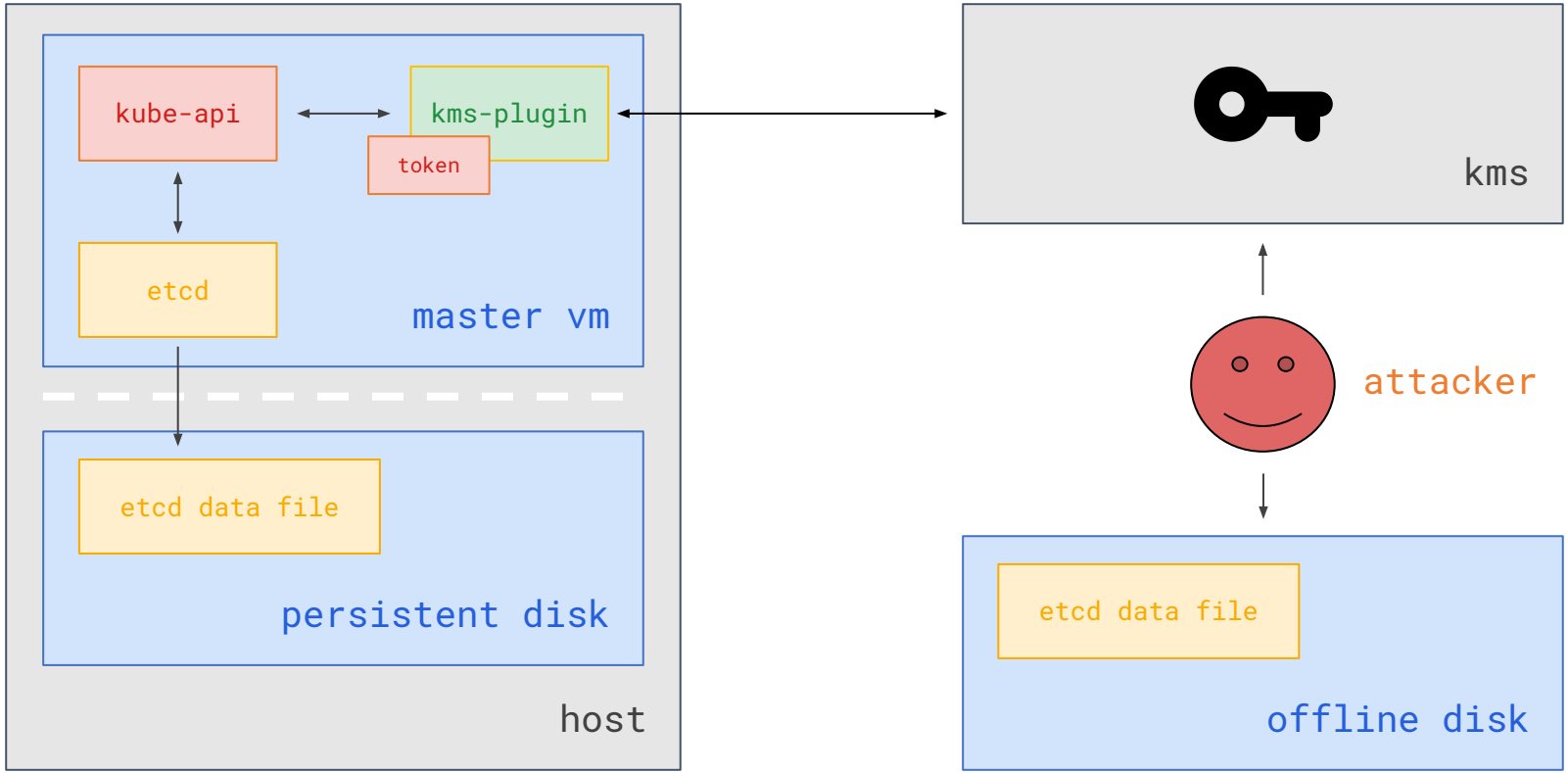
But things are in a much better state!

- requires constant Node access
- mitigated after patching vuln
- use industry standard for trust bootstrap

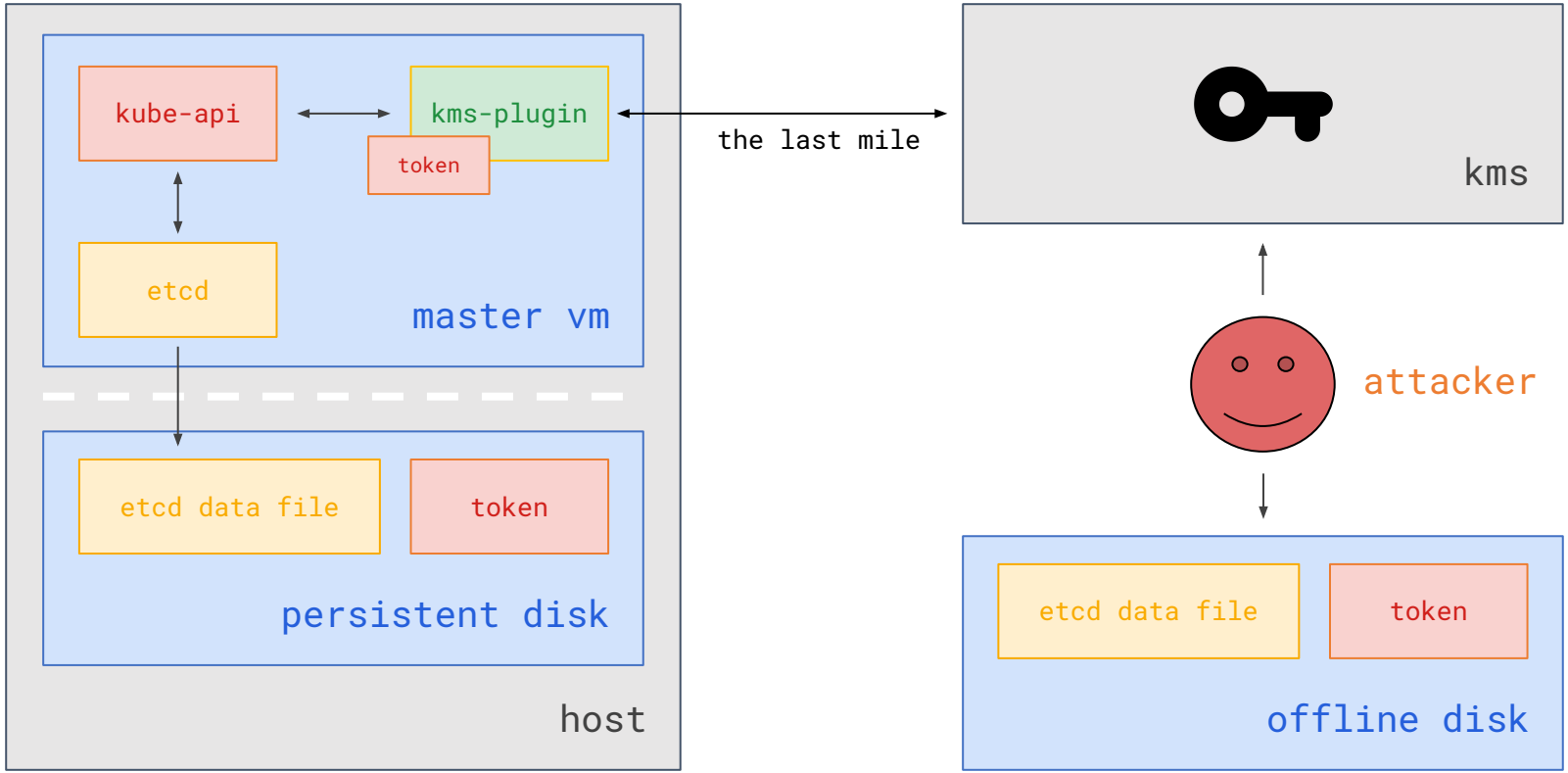
3. Solving the first secret problem


kube-api
kms-plugin
token
etcd
master vm
etcd data file
persistent disk
host
kms

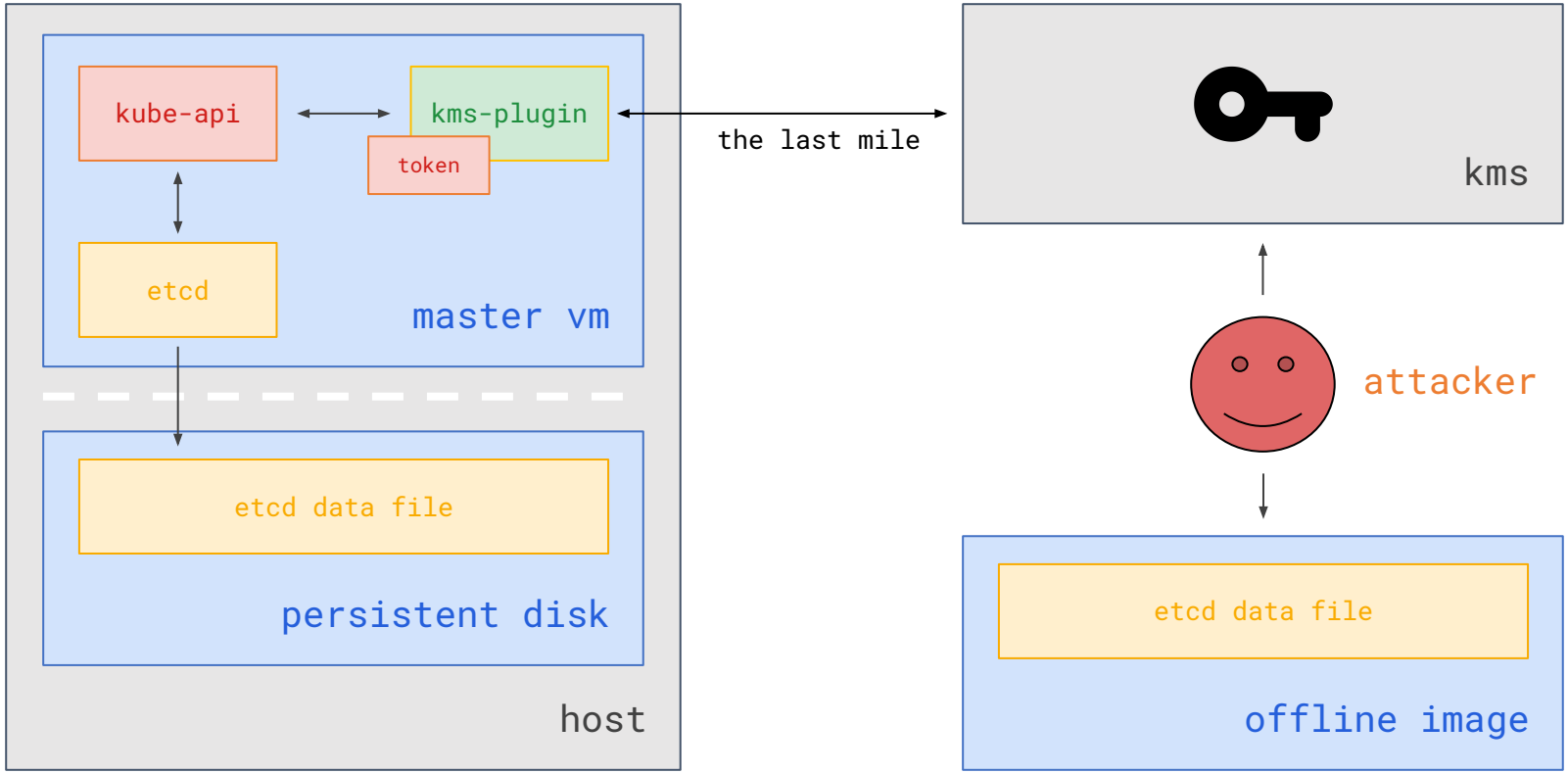
KMS Plugin



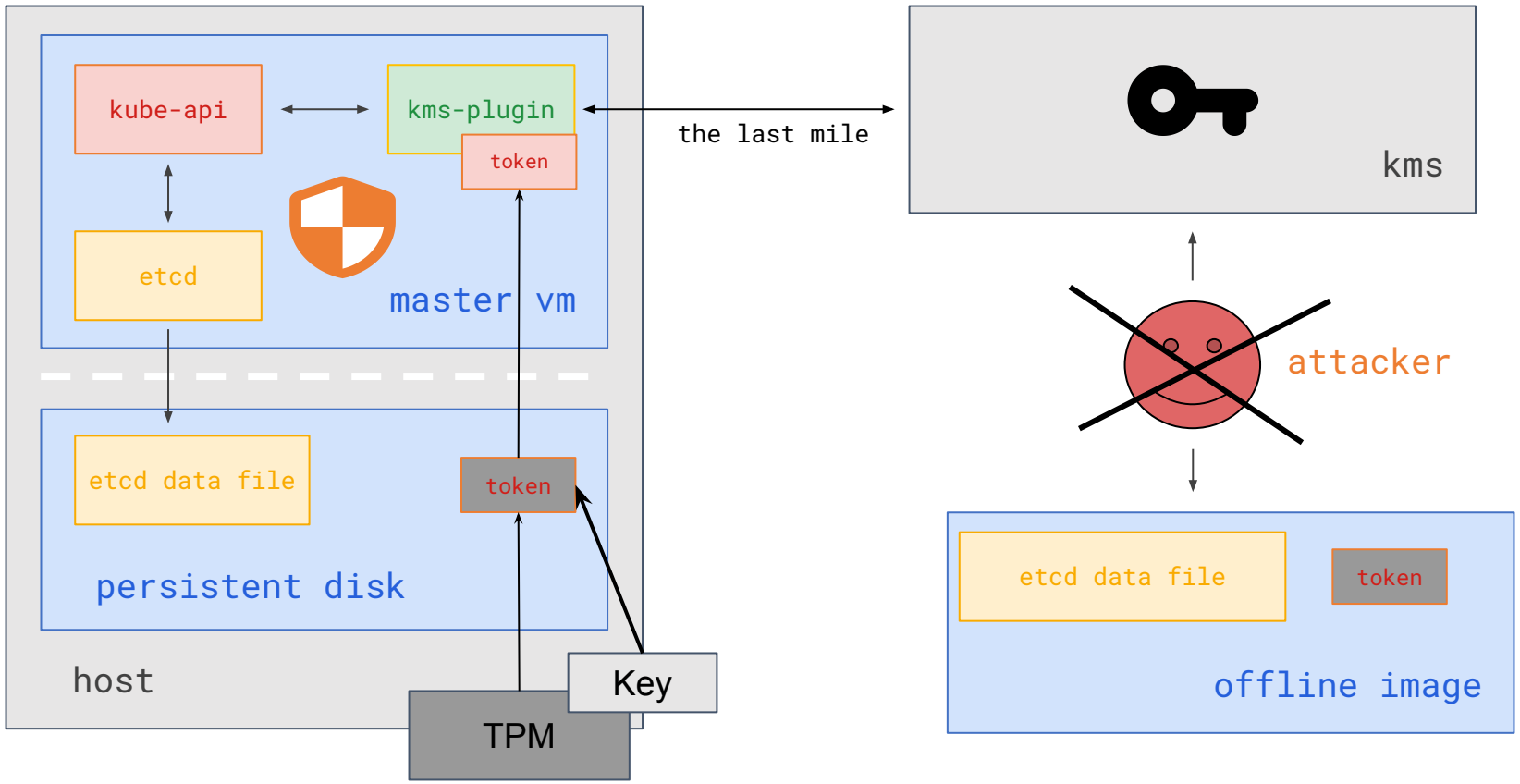
Threat Model



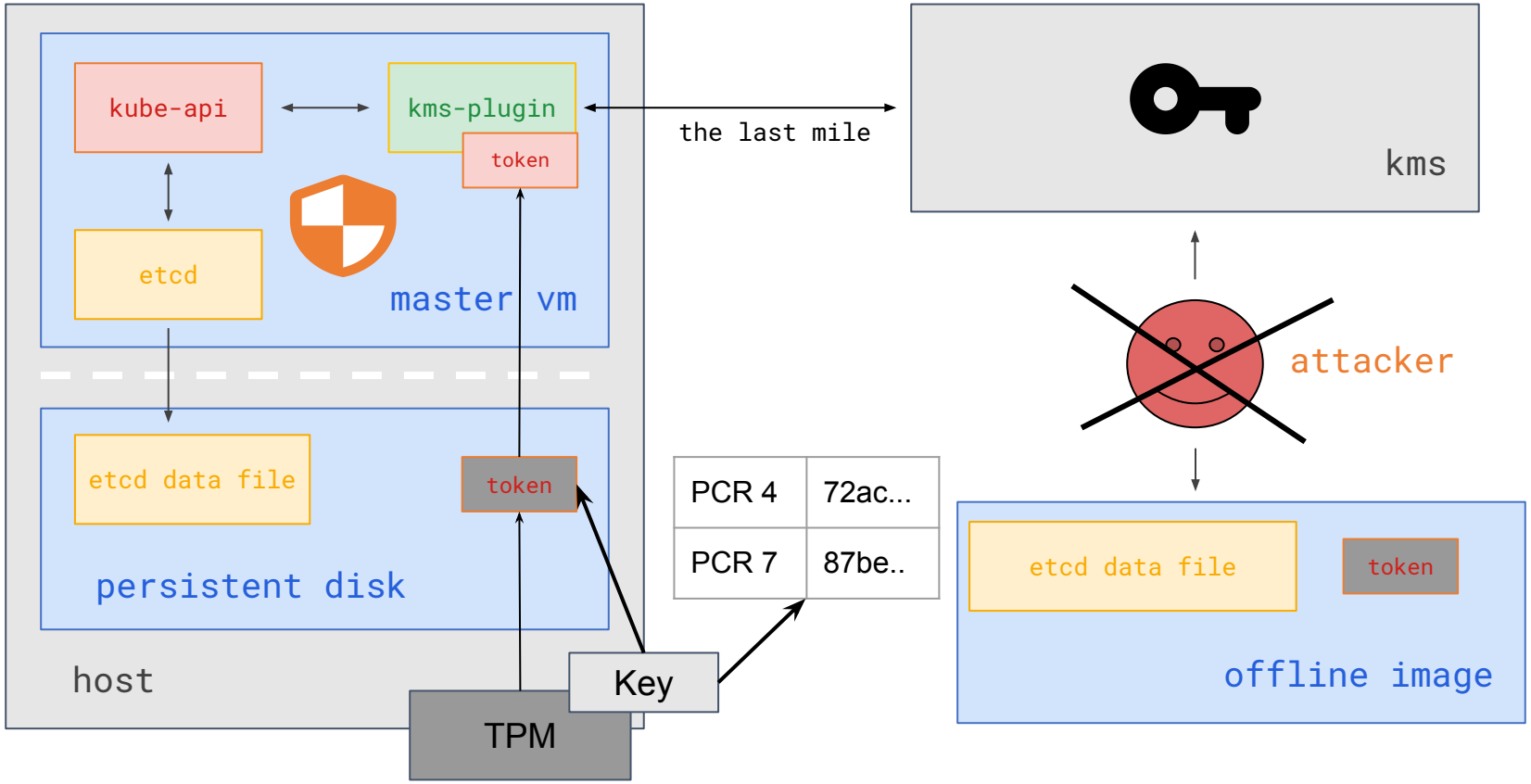
The last-mile problem



Goal: Do **NOT** get access to keys



Solution: Seal KMS Credential to TPM



Apply: PCR Policy

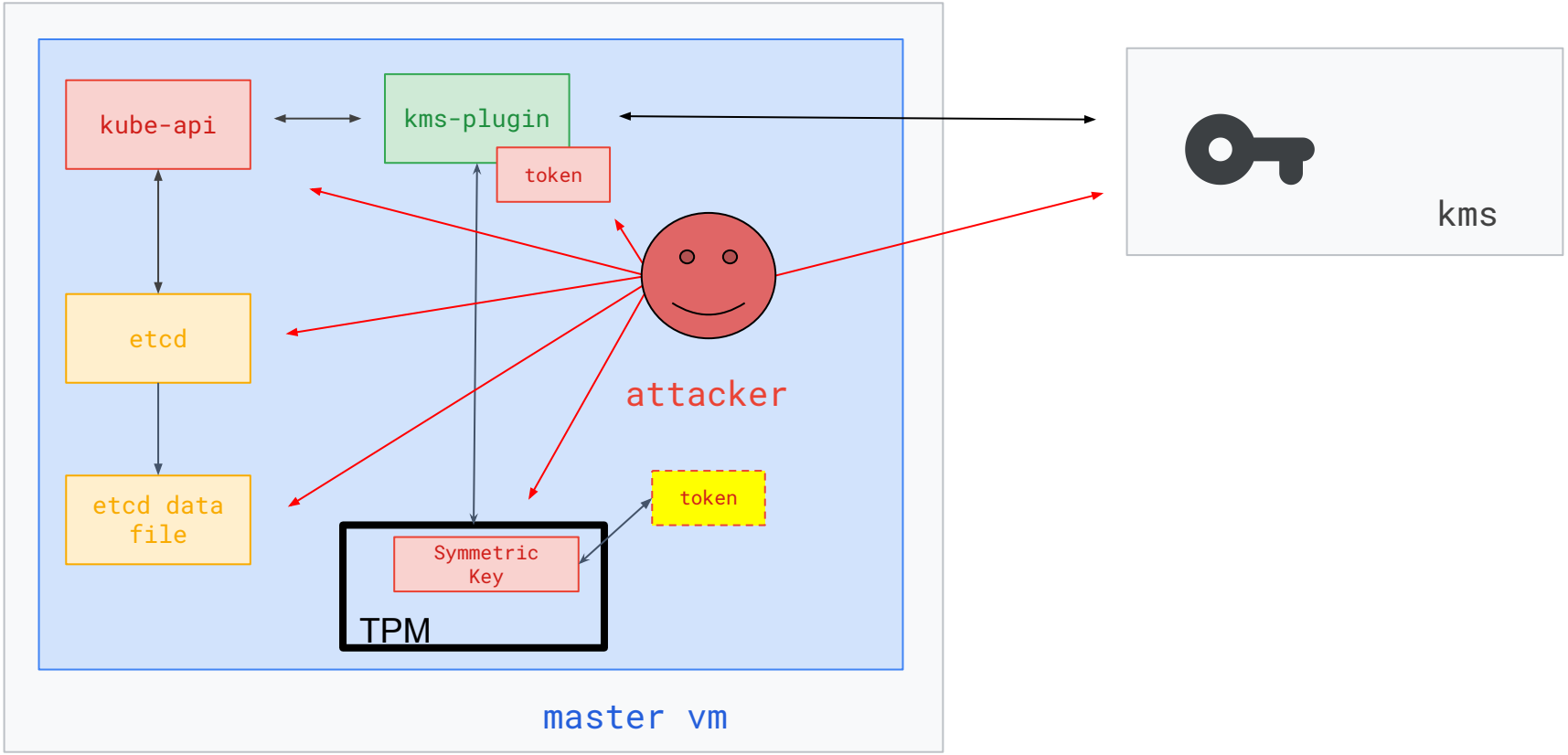
Practical Use

- Embed into your applications: https://github.com/google/go-tpm/tree/master/examples/tpm2-seal-unseal

- Script via an Init container: https://github.com/tpm2-software/tpm2-tools

4. Tamper-evident audit logs*

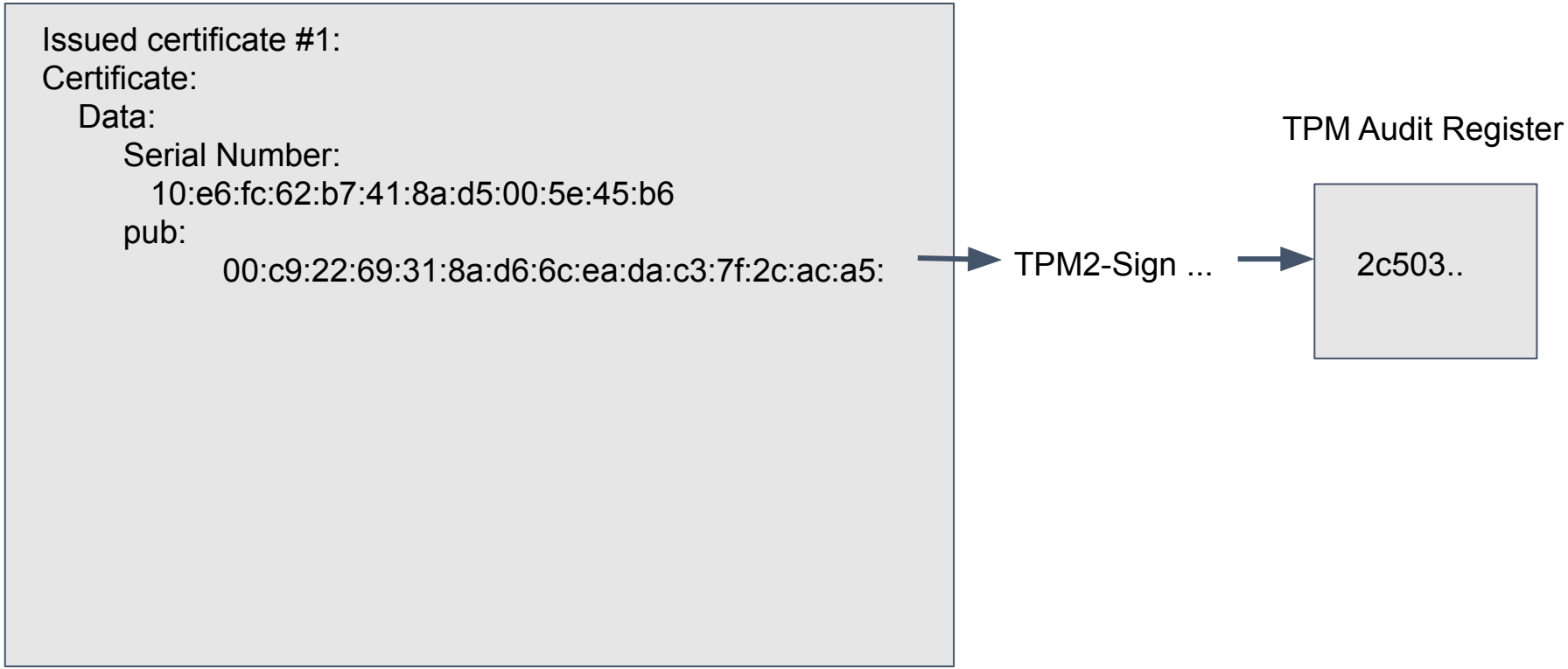
* WIP



Threat Model

$$audit_{new} = \mathbf{H}_{auditAlg}(audit_{old} \,||\, inputHash \,||\, outputHash)$$

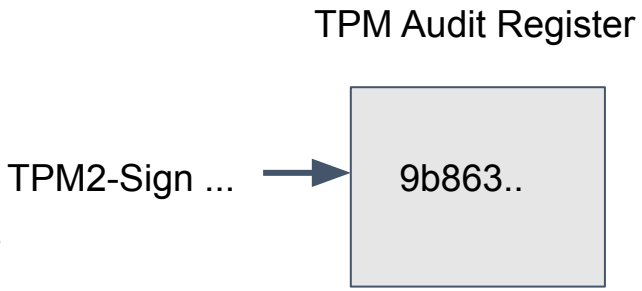
Auditing TPM Commands



Logs are stored externally

```
Issued certificate #1:
Certificate:
    Data:
        Serial Number:
            10:e6:fc:62:b7:41:8a:d5:00:5e:45:b6
        pub:
                00:c9:22:69:31:8a:d6:6c:ea:da:c3:7f:2c:ac:a5:


Issued certificate #2:
Certificate:
    Data:
        Serial Number:
            19:e8:fc:62:b7:41:8a:d5:00:5e:45:b9
        pub:
                00:c7:22:79:31:8a:d6:6c:ea:da:c3:7f:2c:ac:a5:
```



Logs are stored externally

Issued certificate #1:
….
Issued certificate #2
...

Issued certificate #3:
Certificate:
Data:
Serial Number:
20:e6:fc:32:b7:41:8a:d5:00:5e:45:b9
pub:
00:a7:229:79:31:8a:d6:6c:ea:da:c3:7f:2c:ac:a5:

TPM Audit Register

TPM2-Sign ... → 8b300..

Attacker gets a certificate

```
Issued certificate #1:
Certificate:
    Data:
        Serial Number:
            10:e6:fc:62:b7:41:8a:d5:00:5e:45:b6
        pub:
                00:c9:22:69:31:8a:d6:6c:ea:da:c3:7f:2c:ac:a5:

Issued certificate #2:
Certificate:
    Data:
        Serial Number:
            19:e8:fc:62:b7:41:8a:d5:00:5e:45:b9
        pub:
                00:c7:22:79:31:8a:d6:6c:ea:da:c3:7f:2c:ac:a5:
```

TPM Audit Register

TPM2-Sign ... → **8b300**..

Logs are examined externally

TPM2_GetCommandAuditDigest
8b300

Issued certificate #1:
Certificate:
Data:
Serial Number:
10:e6:fc:62:b7:41:8a:d5:00:5e:45:b6
pub:
00:c9:22:69:31:8a:d6:6c:ea:da:c3:7f:2c:ac:a5:

Issued certificate #2:
Certificate:
Data:
Serial Number:
19:e8:fc:62:b7:41:8a:d5:00:5e:45:b9
pub:
00:c7:22:79:31:8a:d6:6c:ea:da:c3:7f:2c:ac:a5:

TPM2_GetCommandAuditDigest
9b863...

TPM2-Sign ...

TPM Audit Register

9b863...

Logs are examined externally

TPM2_GetCommandAuditDigest

Issued certificate #1:
Certificate:
Data:
Serial Number:
10:e6:fc:62:b7:41:8a:d5:00:5e:45:b6
pub:
00:c9:22:69:31:8a:d6:6c:ea:da:c3:7f:2c:ac:a5:

Issued certificate #2:
Certificate:
Data:
Serial Number:
19:e8:fc:62:b7:41:8a:d5:00:5e:45:b9
pub:
00:c7:22:79:31:8a:d6:6c:ea:da:c3:7f:2c:ac:a5:

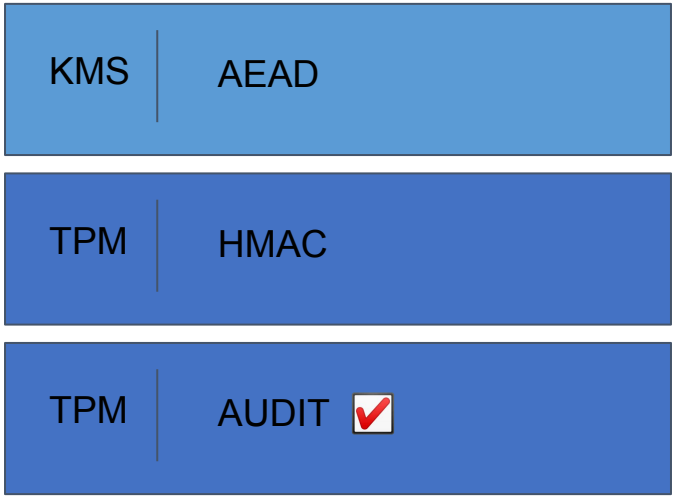
TPM2-Sign ...

TPM Audit Register

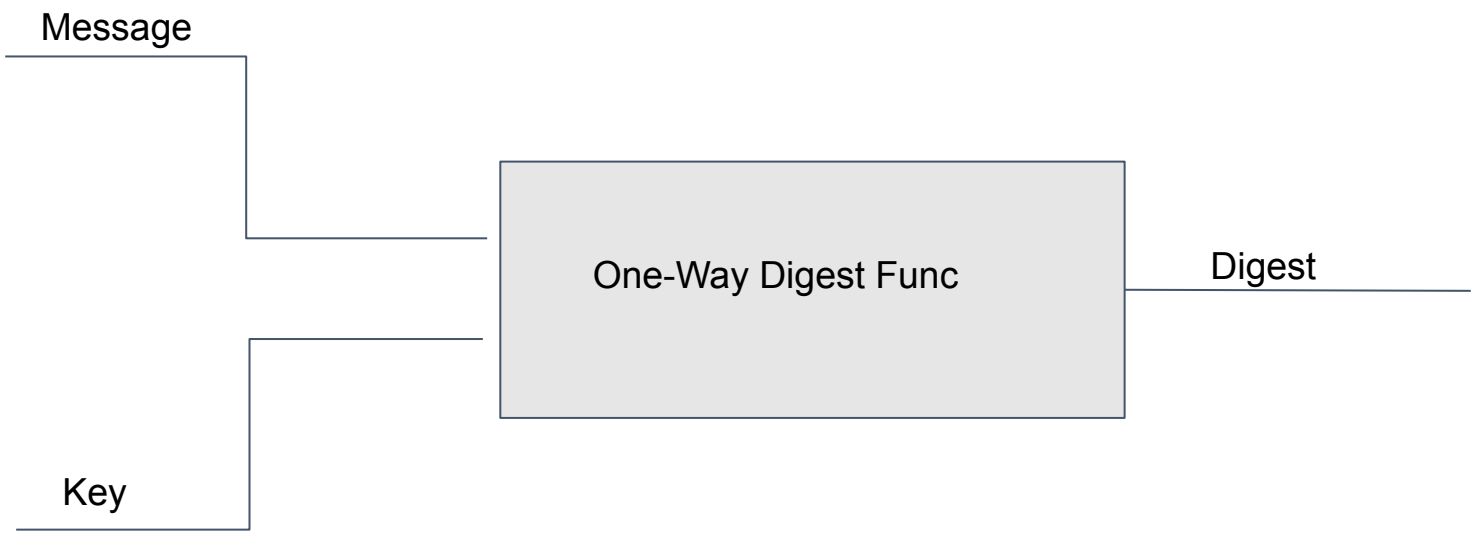
Num Of Logs = 2
...

What if attackers reset and replay

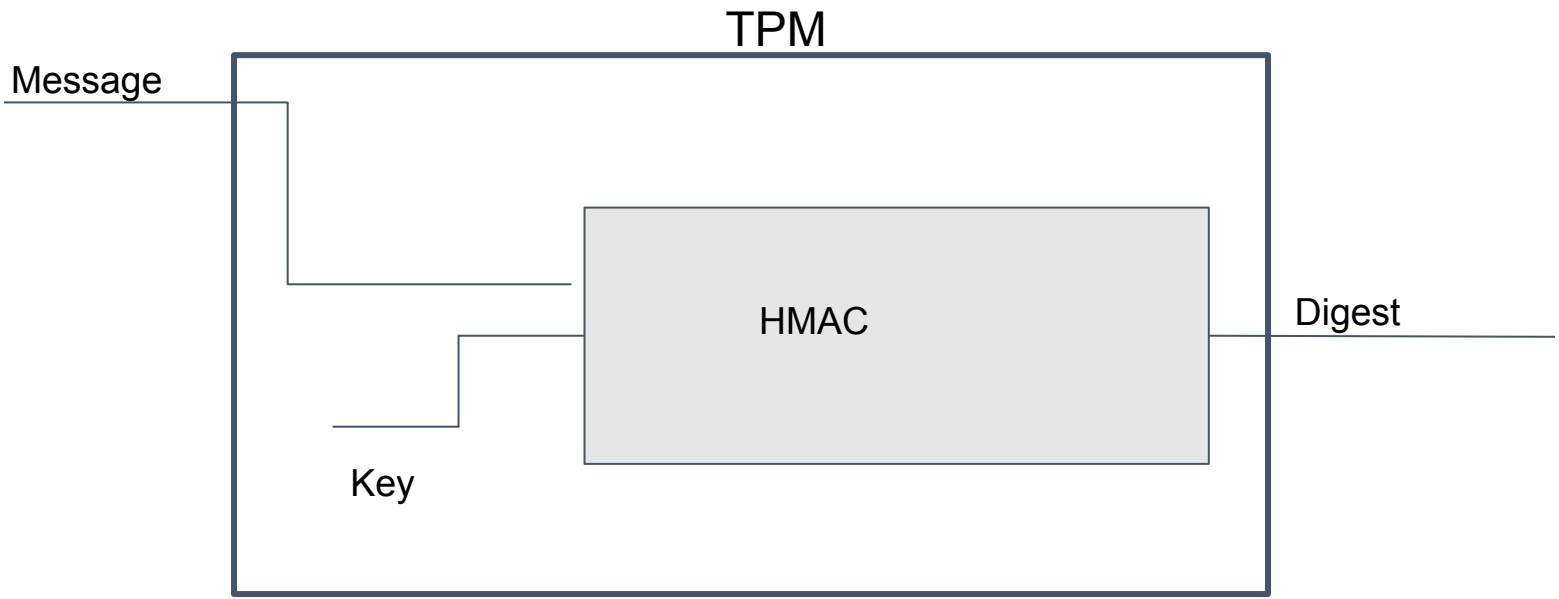
Are we done?



Building Blocks



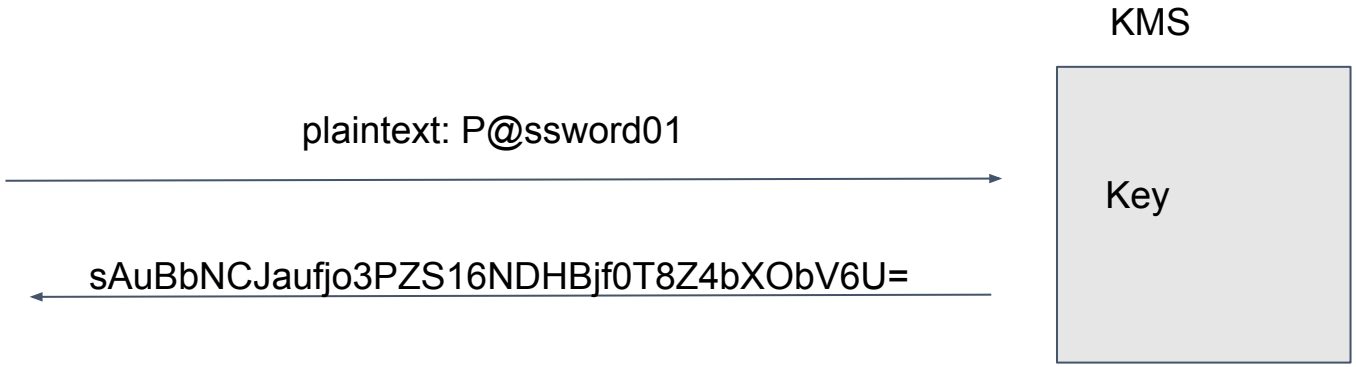
Hash-Based Message Authentication Code



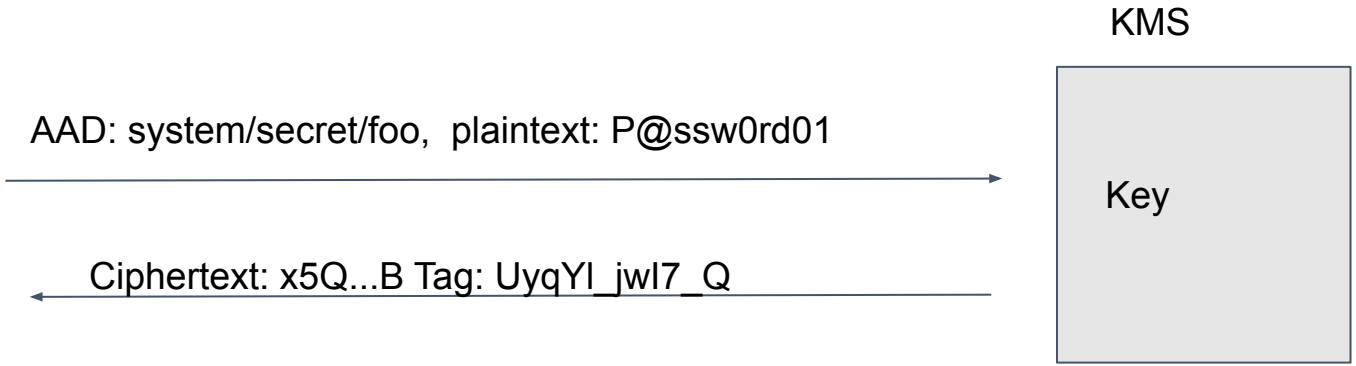
TPM2_HMAC

KMS	AEAD
TPM	HMAC ✔
TPM	AUDIT ✔

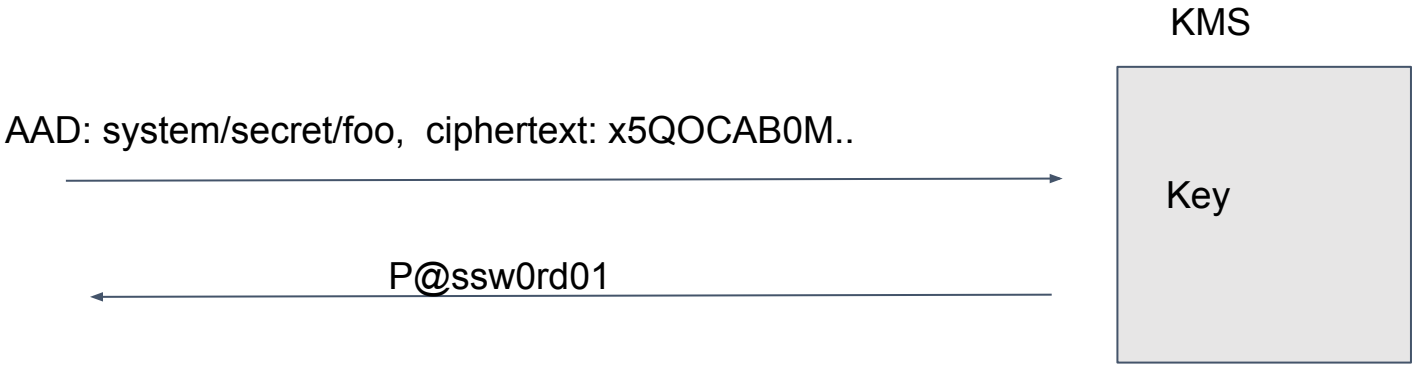
Building Blocks



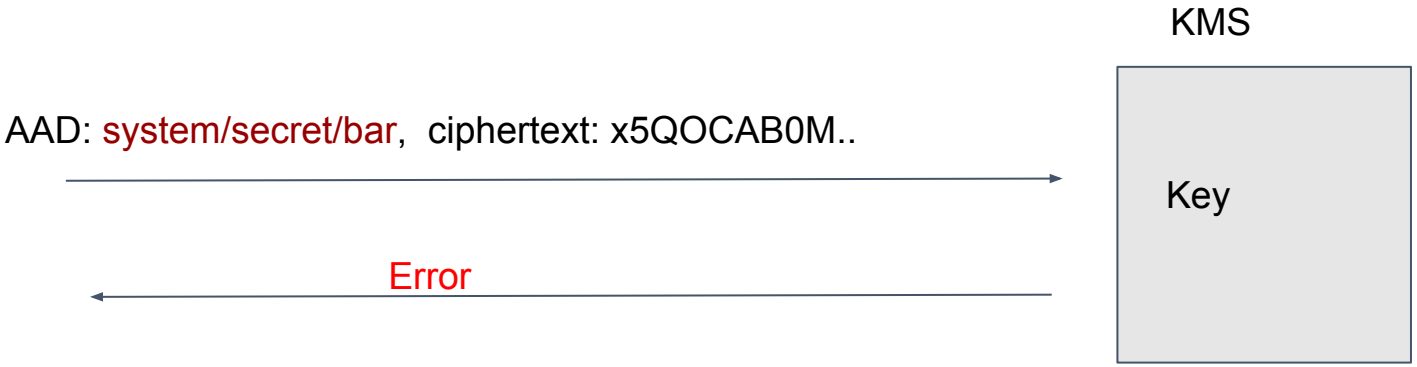
Symmetric Encryption



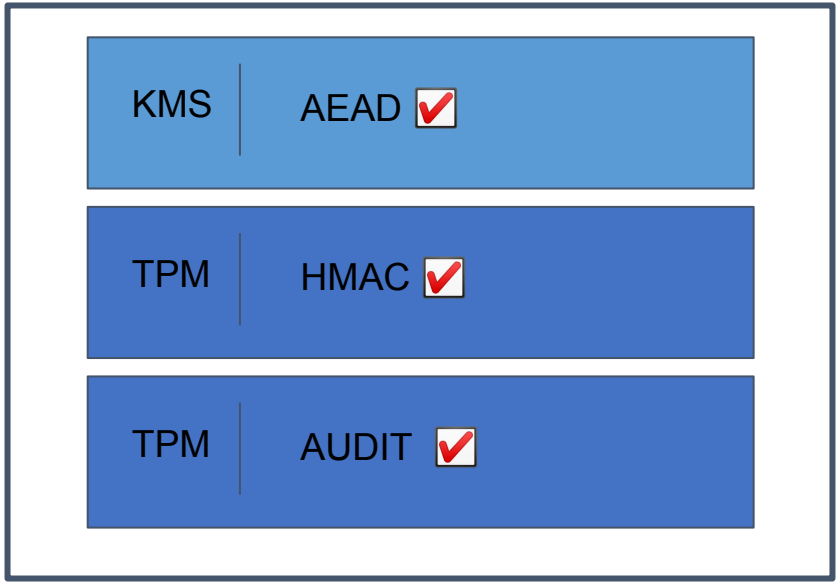
AEAD Encryption



AEAD Encryption



AEAD - AAD must match



Building Blocks

Additional Auth Data	Plaintext
TPM2_HMAC (system/my-dba-pwd)	P@ssw0rd01

Use TPM2_HMAC to generate AAD

Attacks not covered

- Reading directly from kube-apiserver cache
- Reading KEK from kms-plugin cache
- Waiting for a request from a legitimate user and intercepting the response

Summary

When not to use TPMs

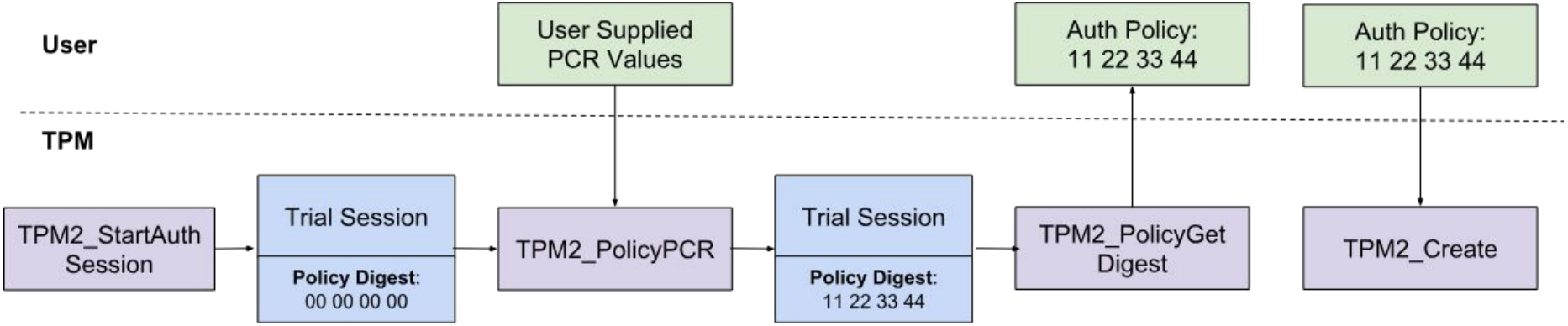
- Performance-sensitive crypto (unless virtual)
- Bulk encryption
- As a substitute for physical security, it is tamper-resistant not tamper-proof

References

- TPM 2.0 specification
- Turtles All the Way Down: Managing Kubernetes Secrets
- Securing Kubernetes Secrets
- Continuous Tamper-proof Logging using TPM2.0
- Cryptographic Support for Secure Logs on Untrusted Machines
- go-tpm library
- TPM2 Tools
- K8S KMS Plugin for Google CloudKMS

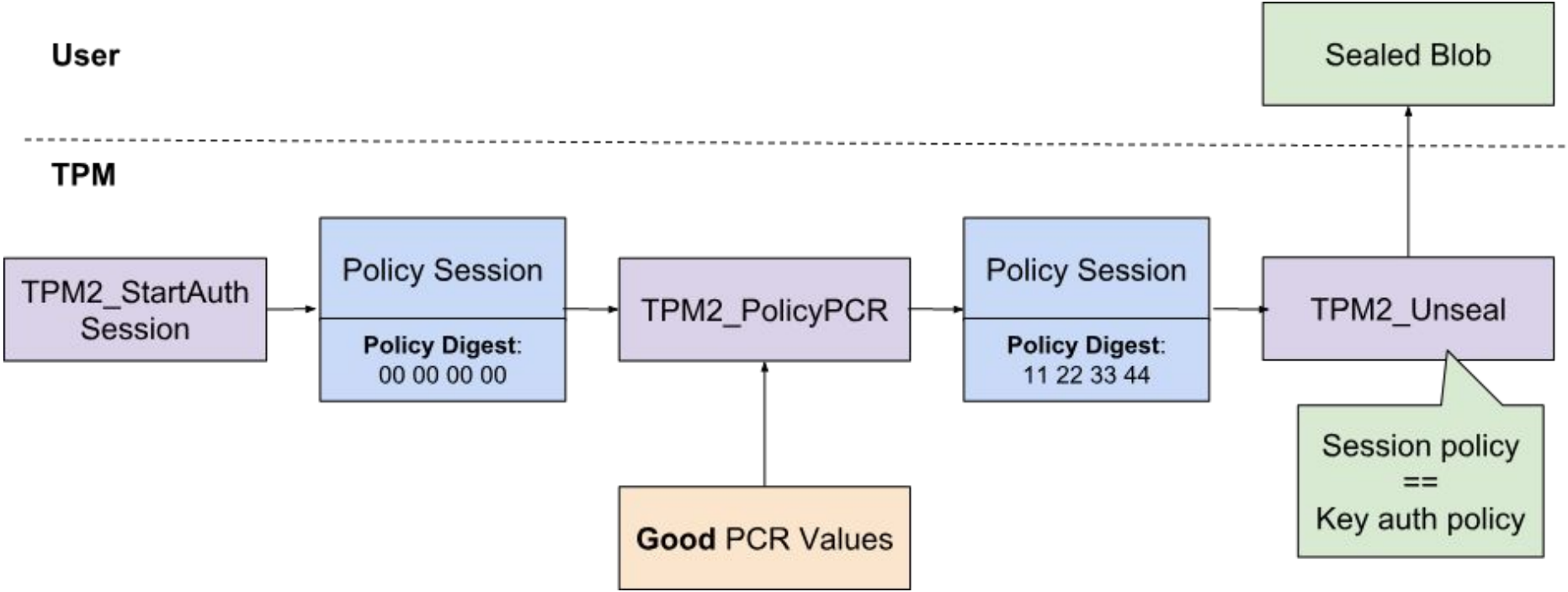
Backup Slides

Sealing to PCR Values



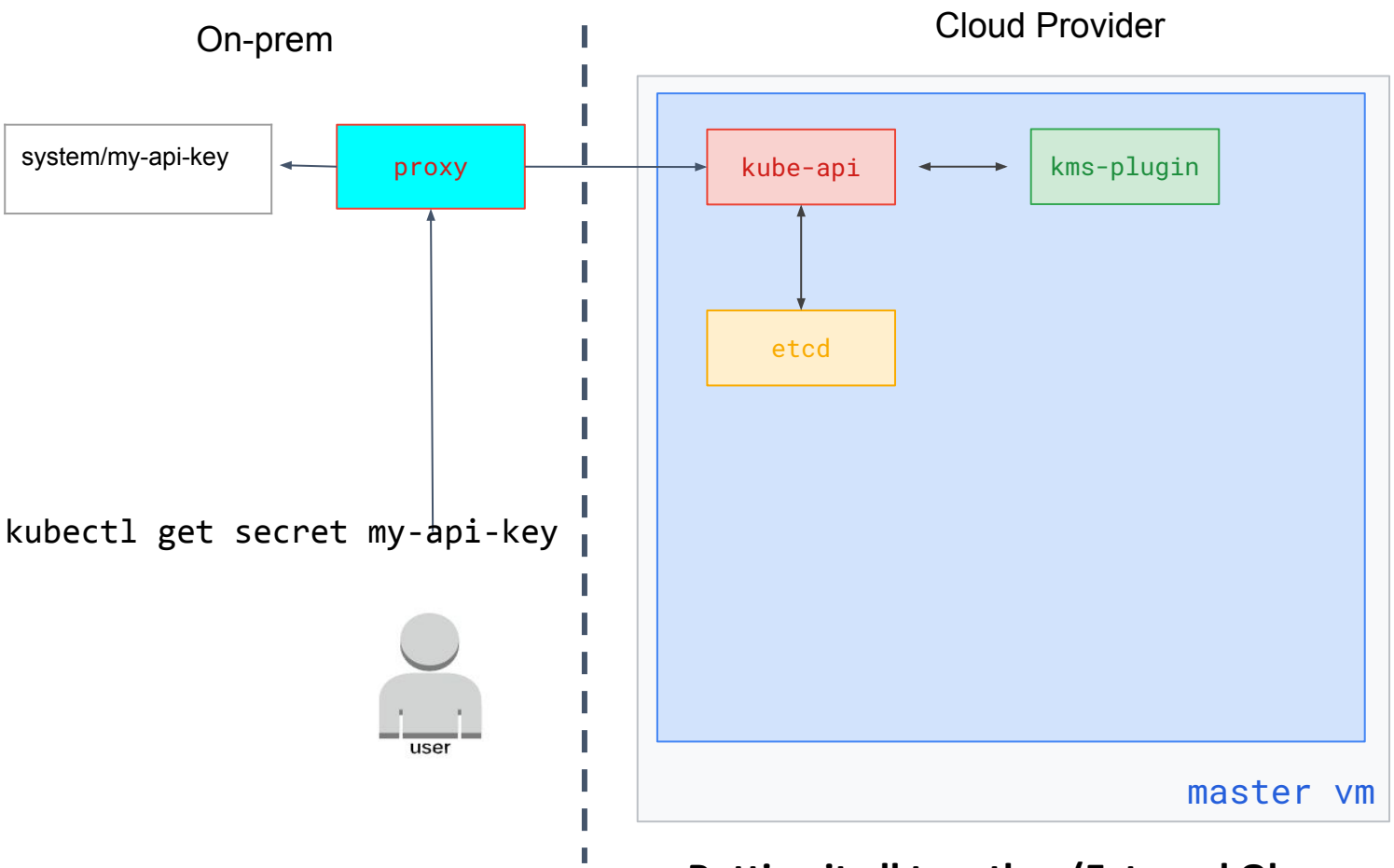
source https://google.github.io/tpm-js

Unsealing

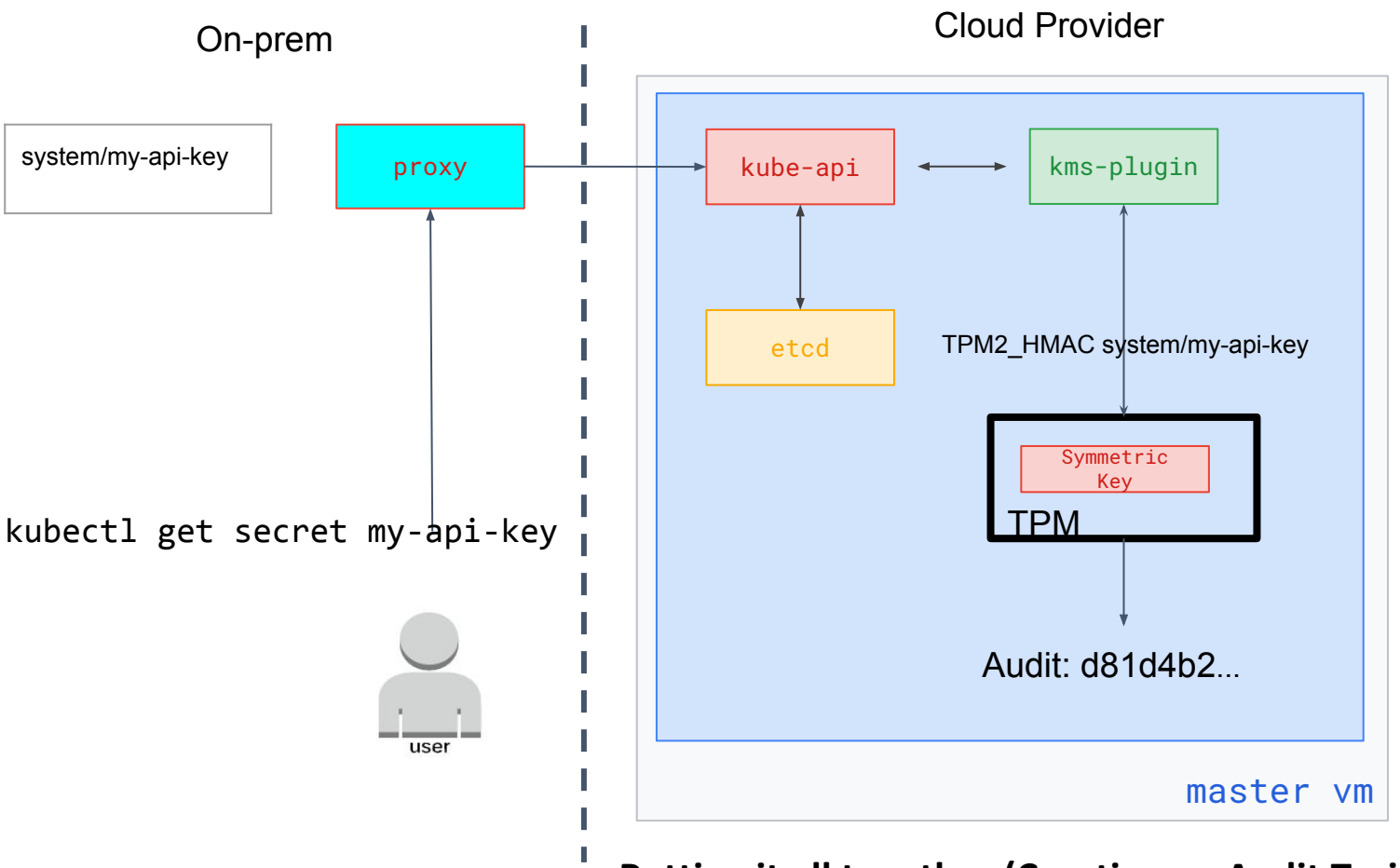


source https://google.github.io/tpm-js

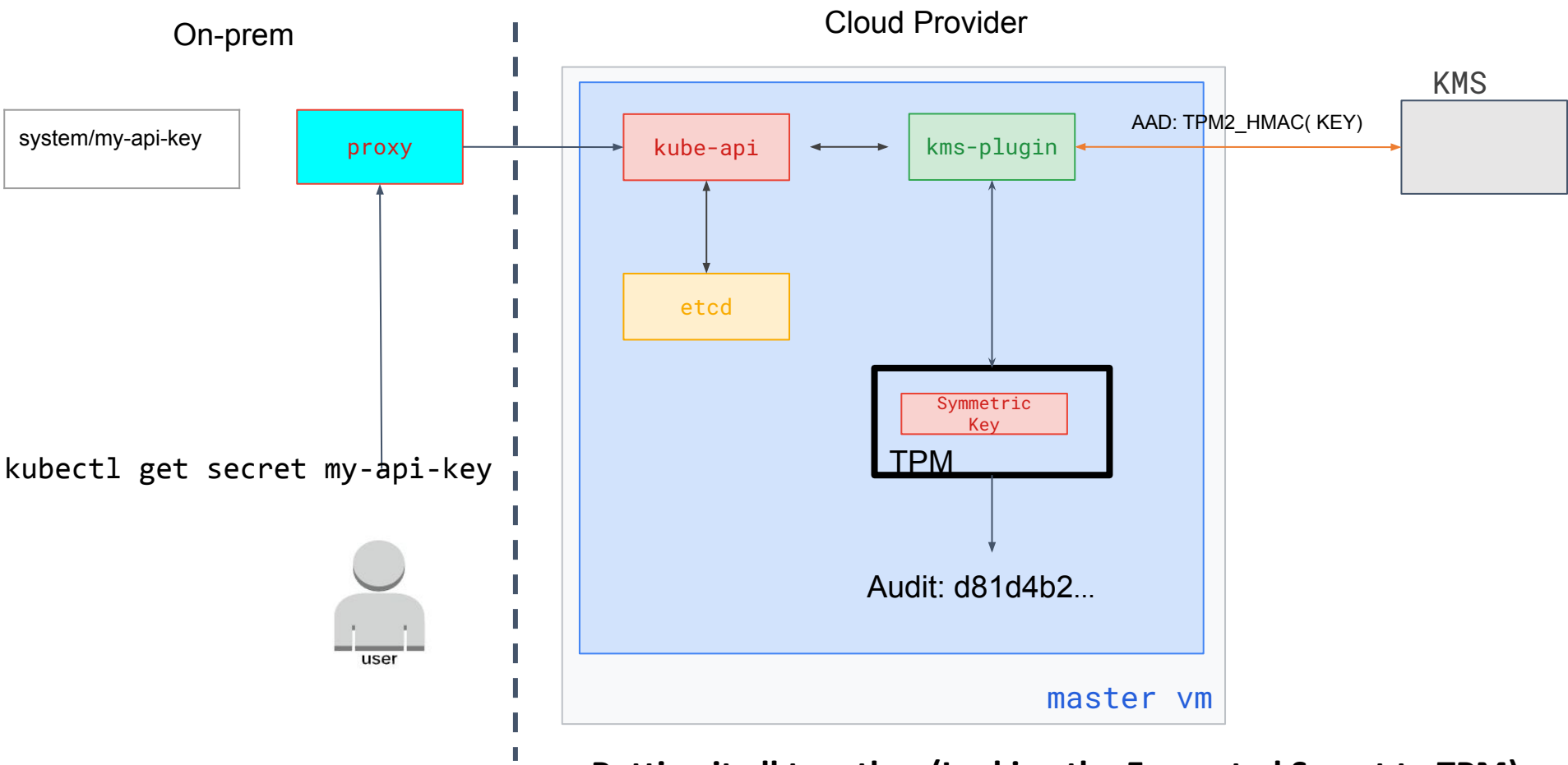
Summarizing Tamper Evident Loging



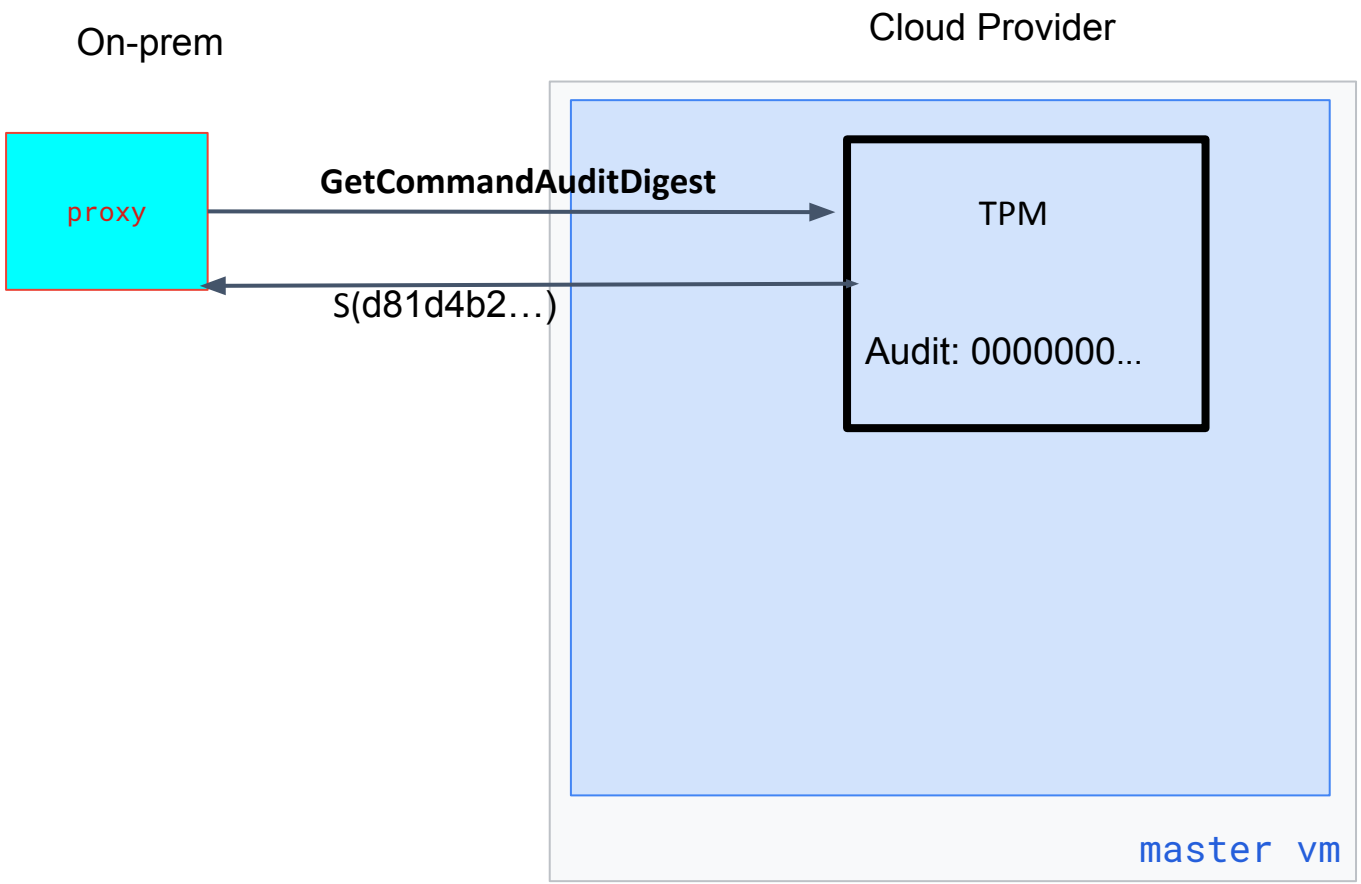
Putting it all together (External Observer - Proxy)



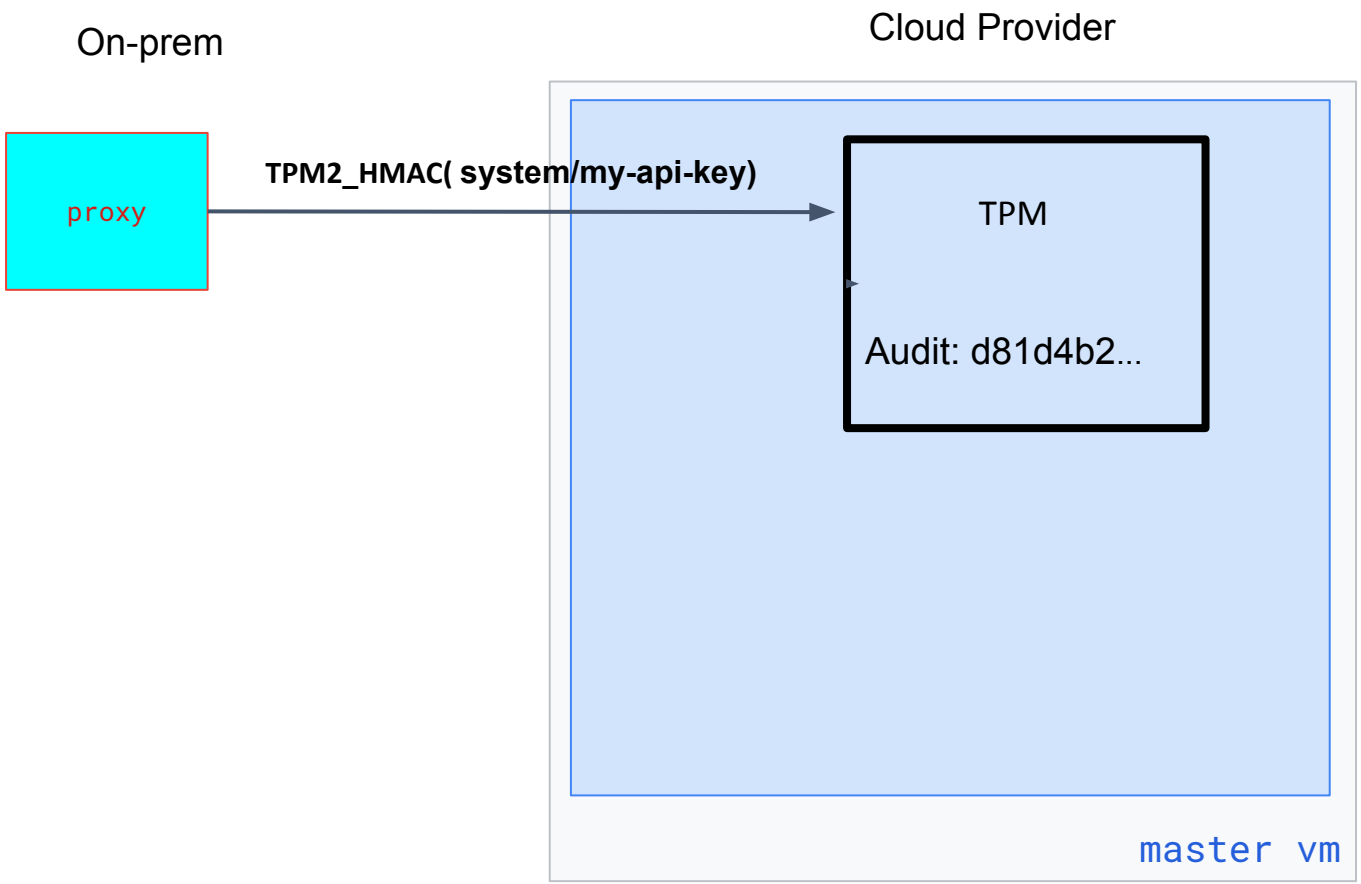
Putting it all together (Creating an Audit Trail)



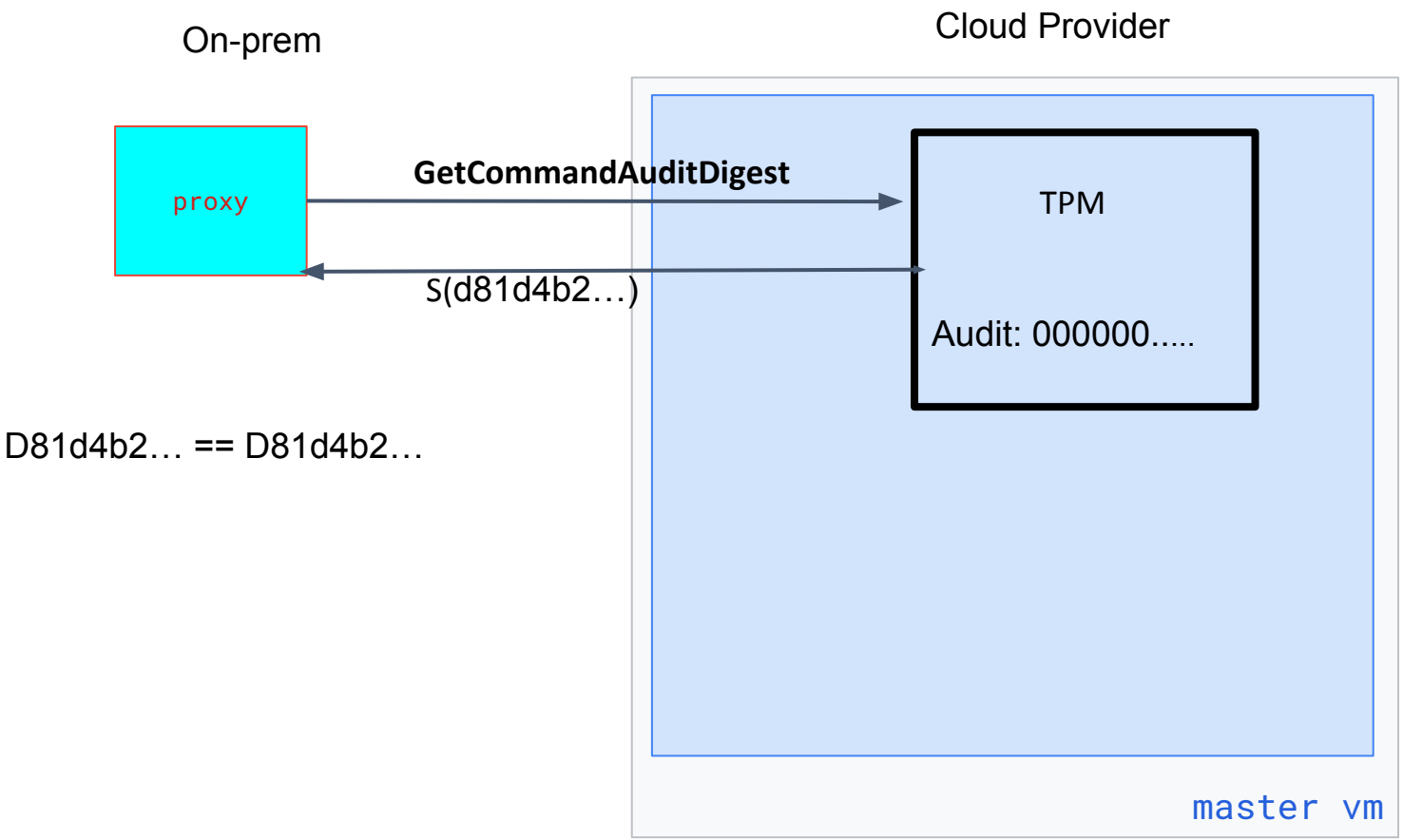
Putting it all together (Locking the Encrypted Secret to TPM)



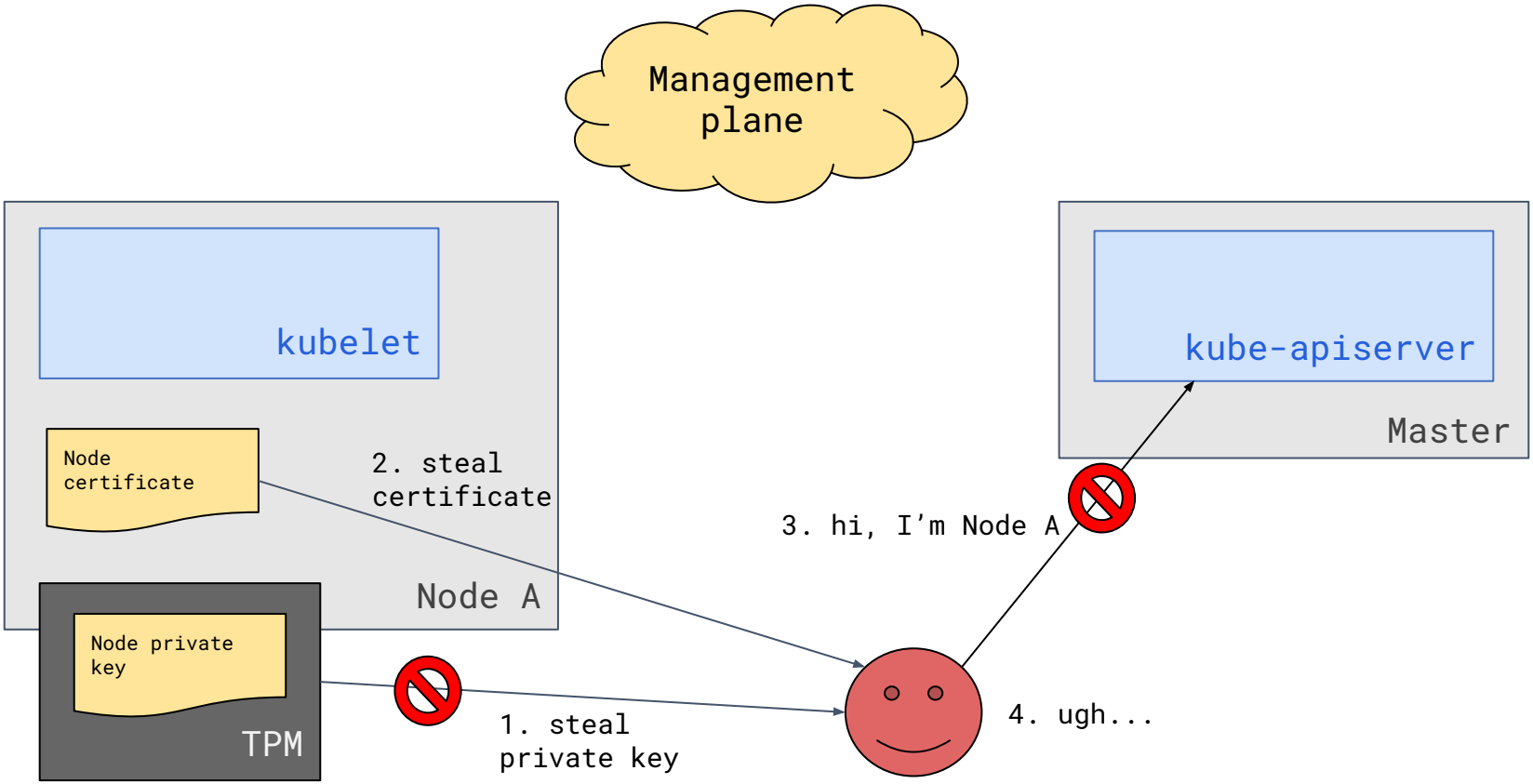
Validation - Get Signed Value of the Audit Register



Validation - Replay TPM2_HMAC commands



Validation - Get Signed Value of the Audit Register


Management plane
kubelet
Node certificate
2. steal certificate
Node A
Node private key
TPM
1. steal private key
kube-apiserver
Master
3. hi, I'm Node A
4. ugh...