



KubeCon



CloudNativeCon

Europe 2019

Securing Kubernetes with Trusted Platform Module

Who are we, anyway?



KubeCon



CloudNativeCon

Europe 2019

Alex Tcherniakhovski

alextc@google.com

Security Engineer, Google
Kubernetes Engine

Andrew Lytvynov

awly@google.com

Software Engineer, Google
Kubernetes Engine

This talk

Goals:

- sample of TPM capabilities
- theoretical applications
- fuel exploration by users and sig-auth

Non-goals:

- demo real implementations
- pitch cloud provider features
- pitch existing sig-auth projects

Tricky security problems



KubeCon



CloudNativeCon

Europe 2019

Node trust bootstrap

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

Tricky security problems



KubeCon



CloudNativeCon

Europe 2019

First secret problem

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

Tricky security problems



KubeCon



CloudNativeCon

Europe 2019

Cryptographic audit logging

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

Agenda



KubeCon



CloudNativeCon

Europe 2019

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

What's a TPM?



KubeCon



CloudNativeCon

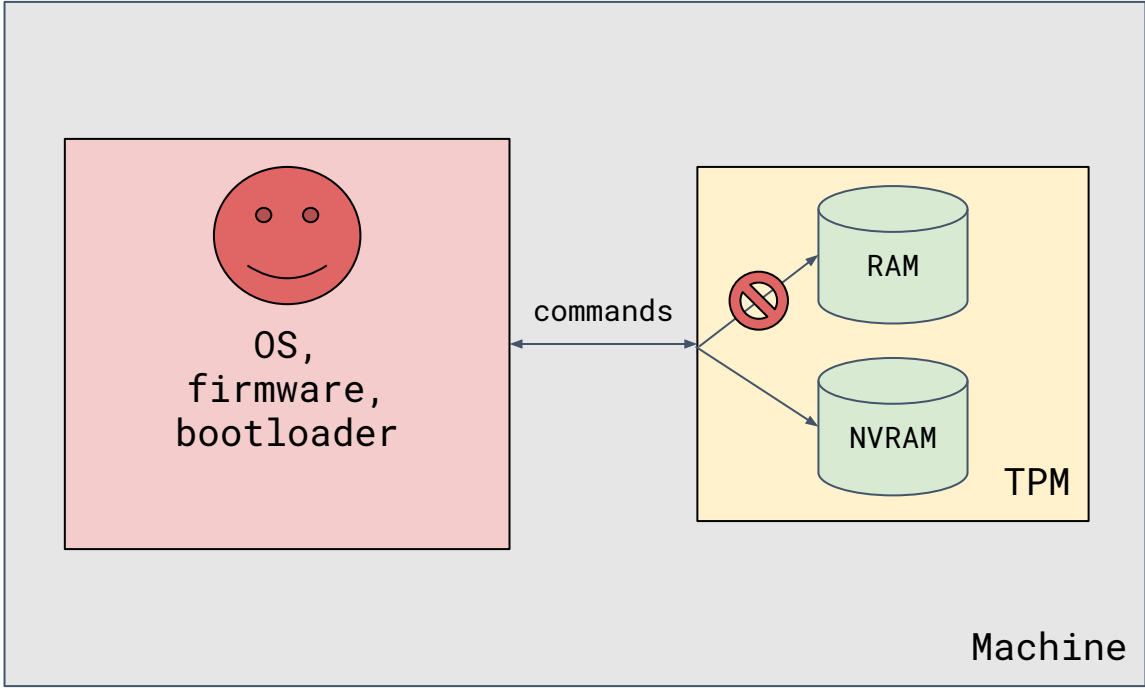
Europe 2019

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



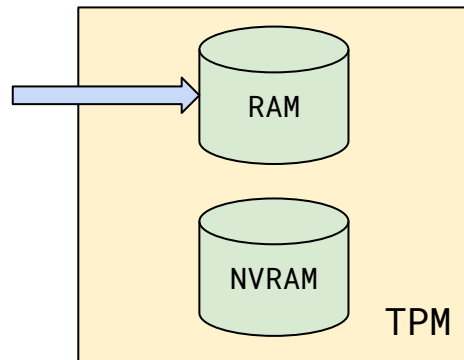
Classic use cases:

- 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



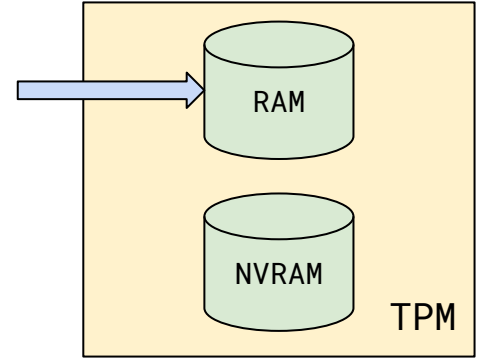
TPM keys

- RSA or ECDSA
- Encryption or signing
- Symmetric or asymmetric
- TPM-bound
 - no exfiltration
 - can export from TPM, but only encrypted
- Used via specialized commands
- Can be persisted, but usually flushed and re-created on demand

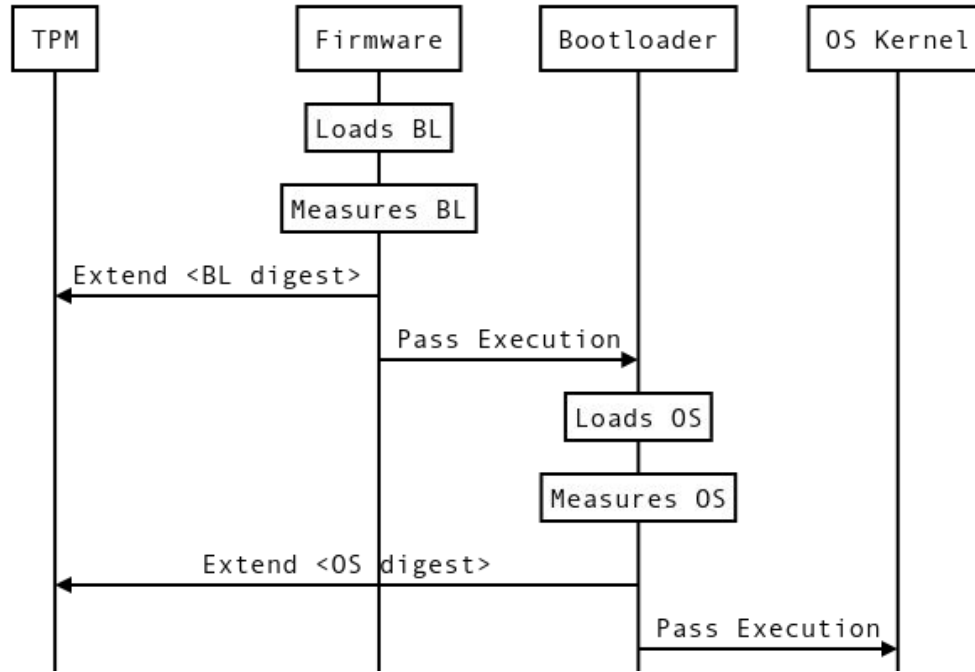


Platform Configuration Registers (PCRs)

- Hash value of a chain of events
- Same sequence of events - same PCR value
- TPM2_Quote signs current value with a key
 - used to remotely prove PCR state

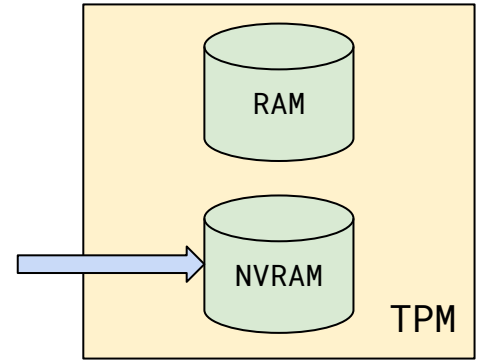


PCR and Measured Boot



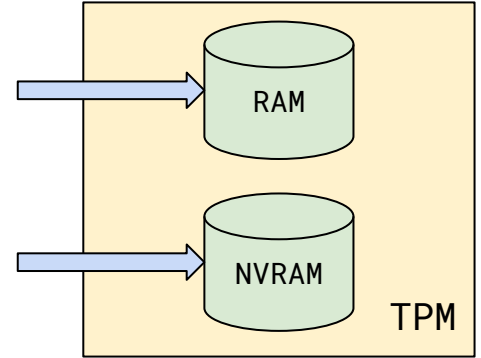
NVRAM (non-volatile RAM)

- Persistent memory
- Small capacity
- Not secure on its own
 - Encrypt (seal) valuable data with a TPM key to protect
- Binary data, counters, locks



Endorsement Key (EK)

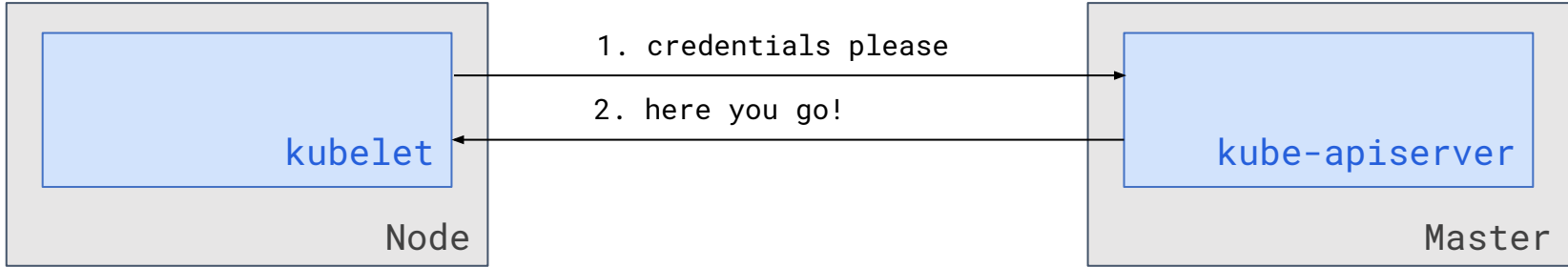
- Key baked into TPM
- Certificate signed by TPM vendor in NVRAM
- Used as machine identity



A whole lot more...

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

2. Node trust bootstrap



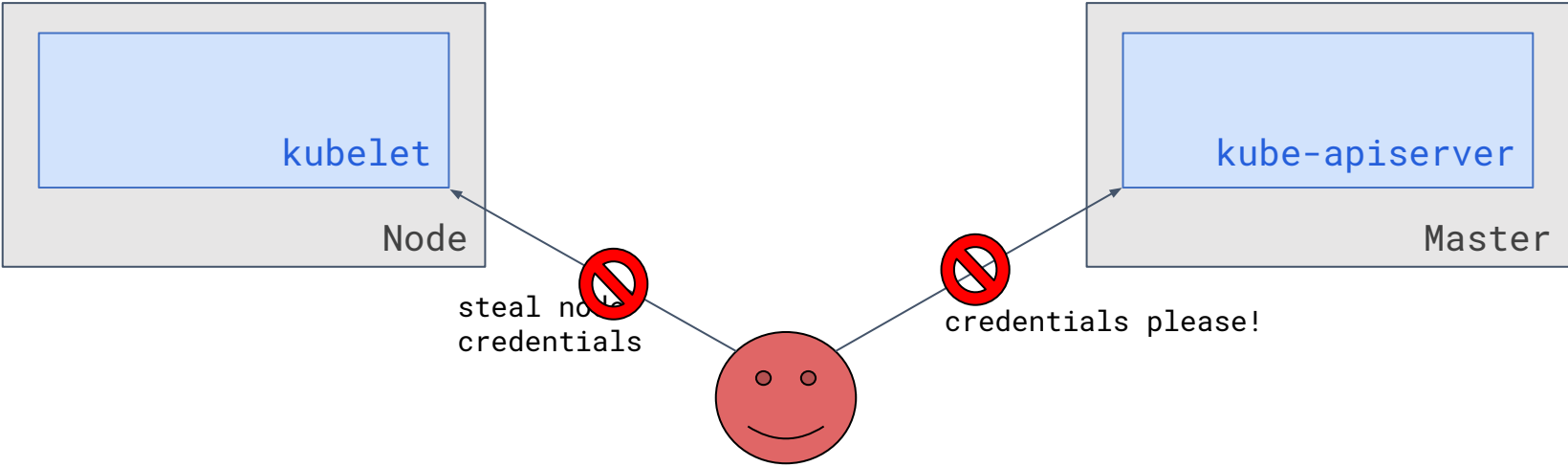
Threat model

Attacker has:

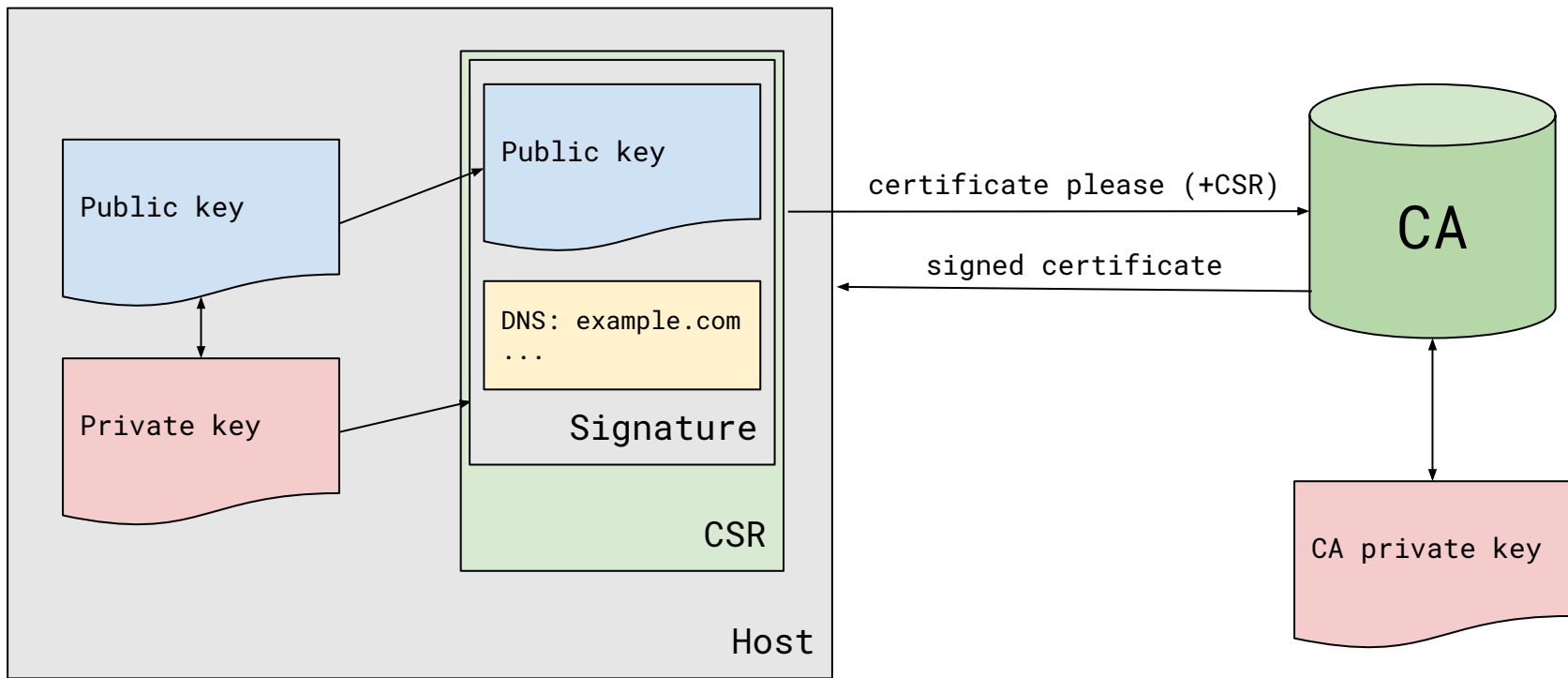
- compromised Pod
- compromised Node

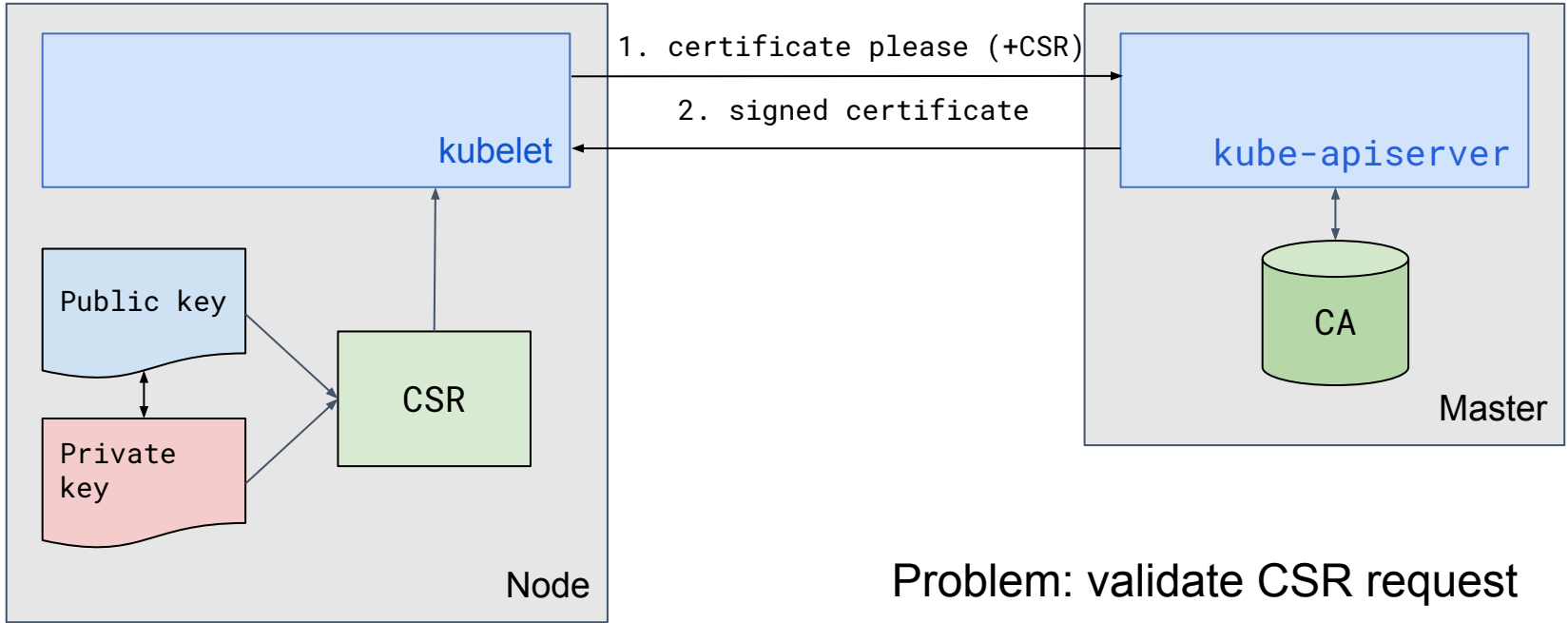
Attacker wants:

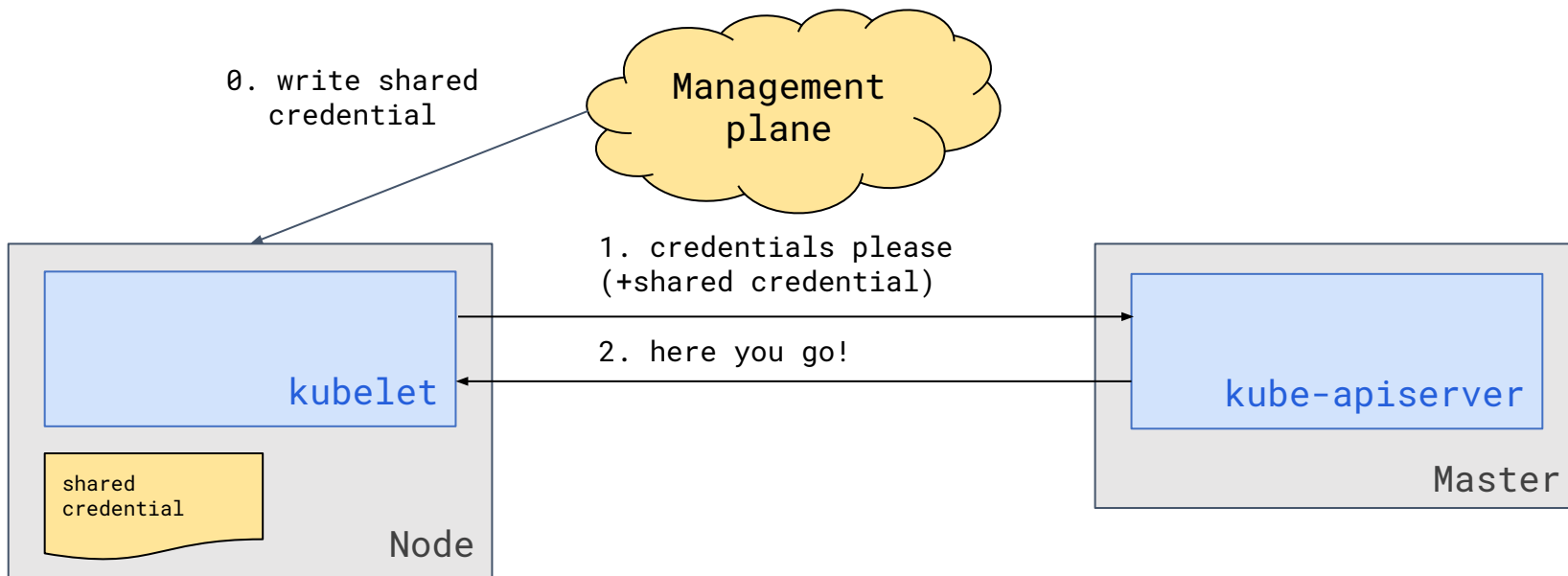
- exfiltrate application configs
- exfiltrate application Secrets
- persist access

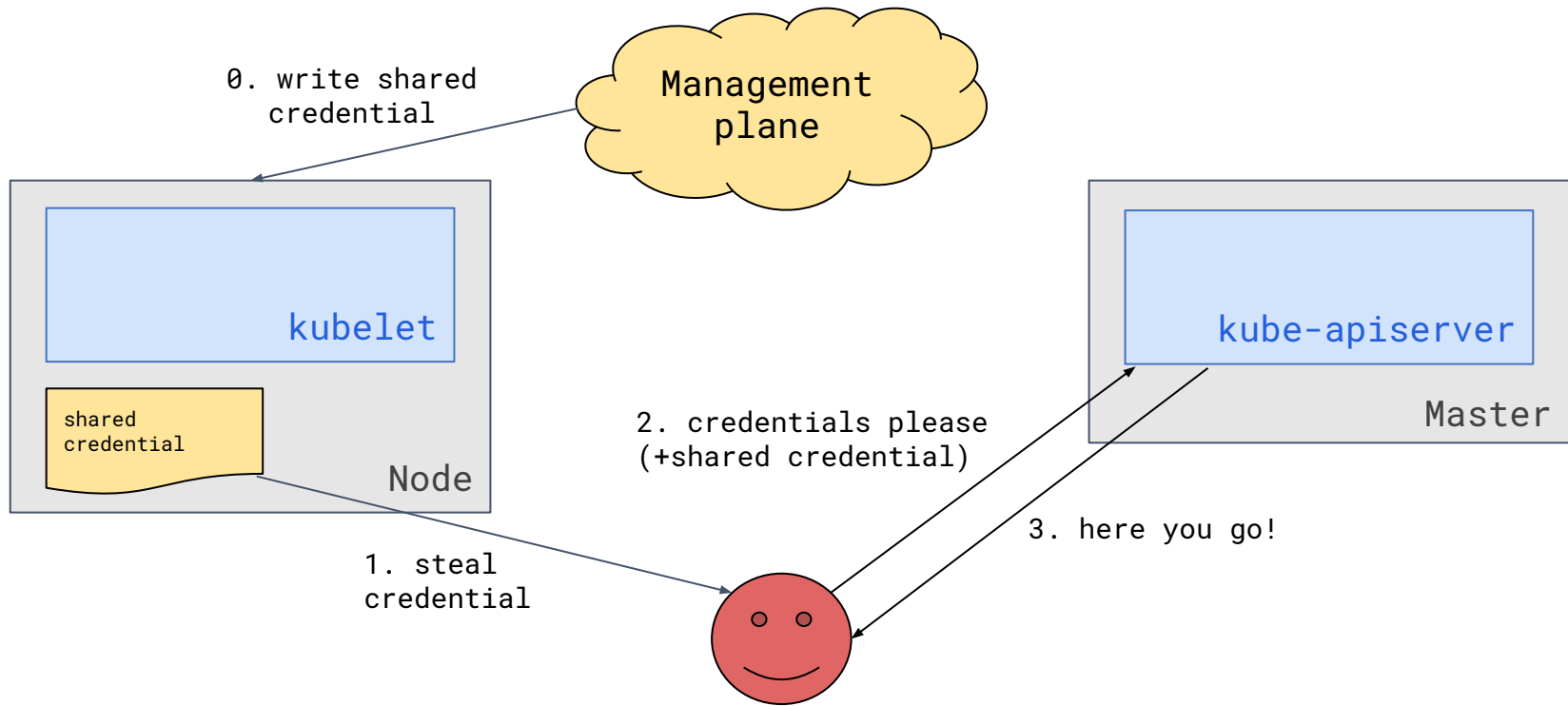


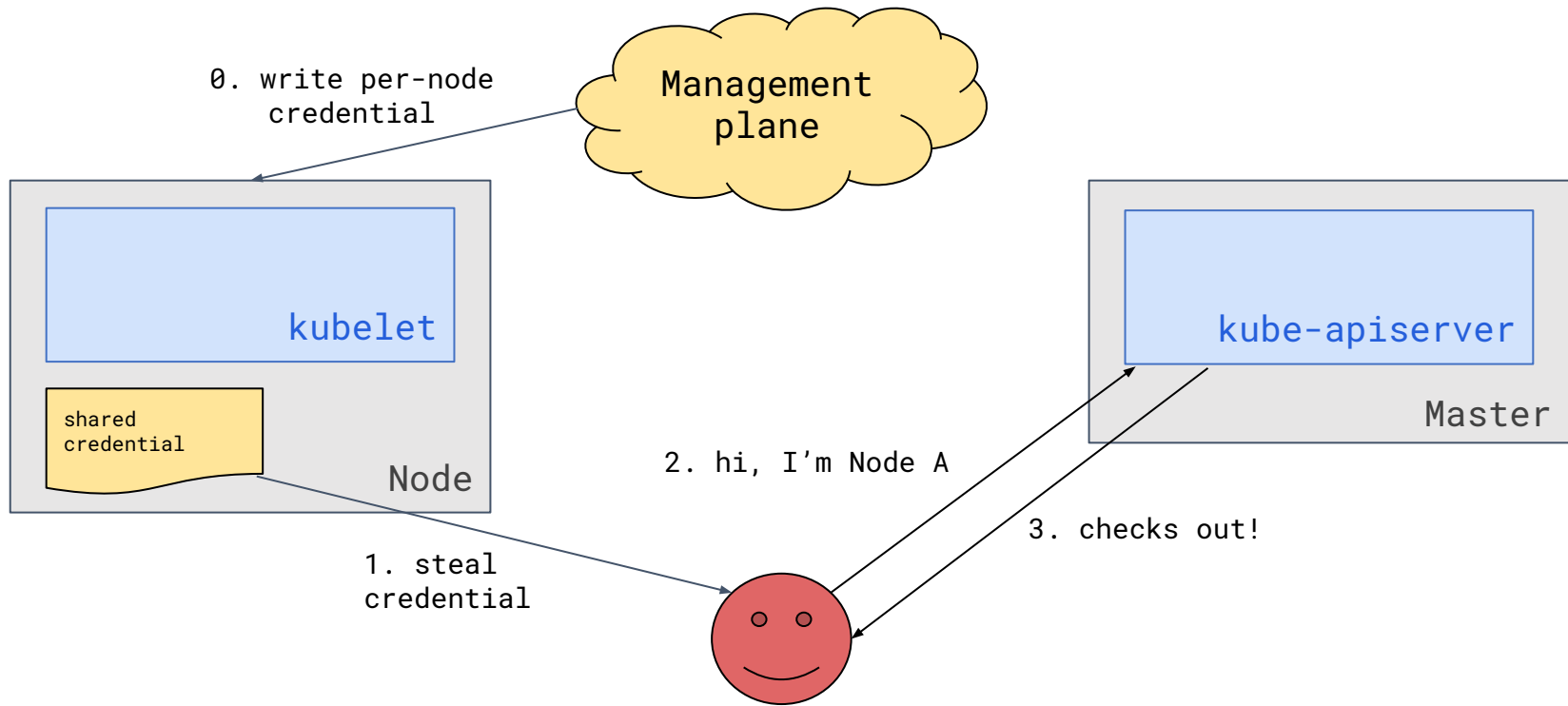
Enter X.509 CSRs and Certificates





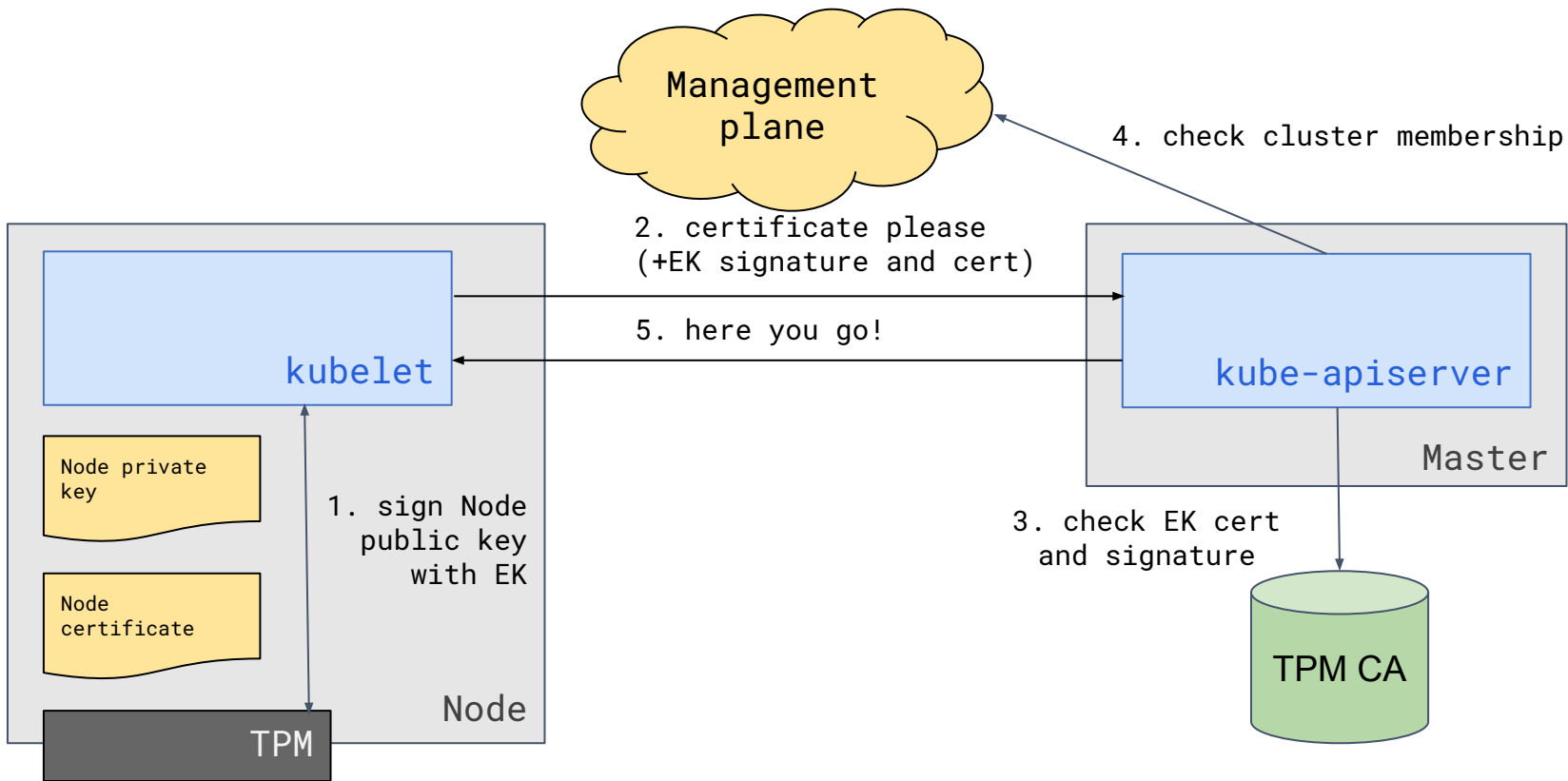






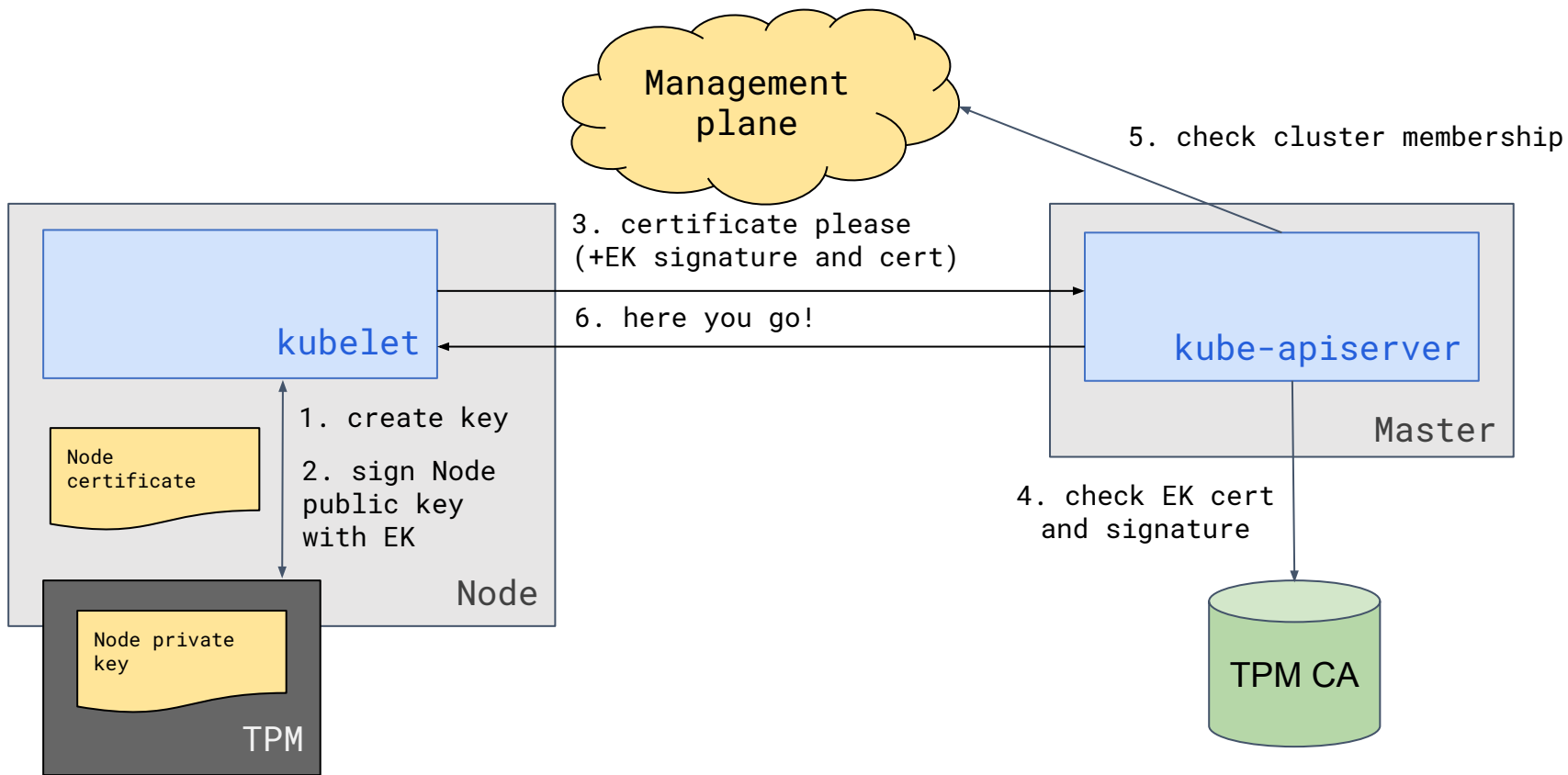
Let's use a TPM!

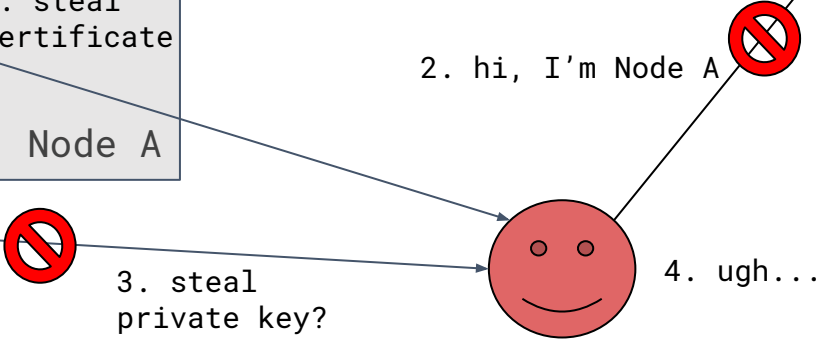
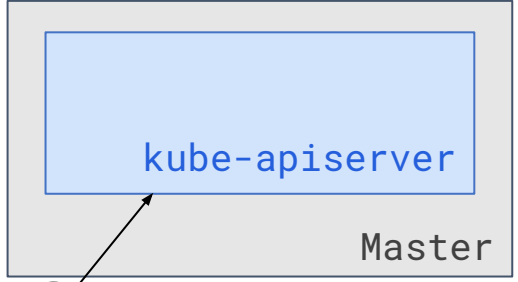
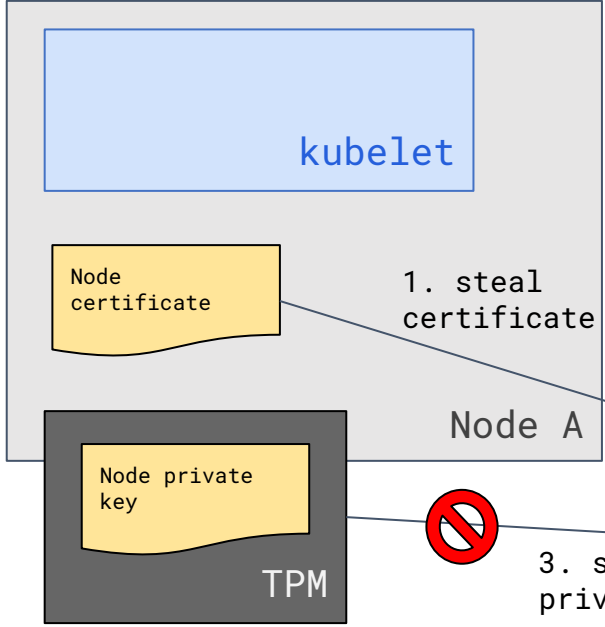
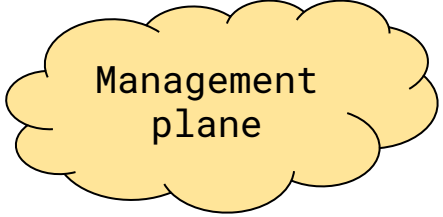
EK as proof of machine identity



But what about exfiltration of the Node credential after provisioning?

Put it in a TPM!





TPM

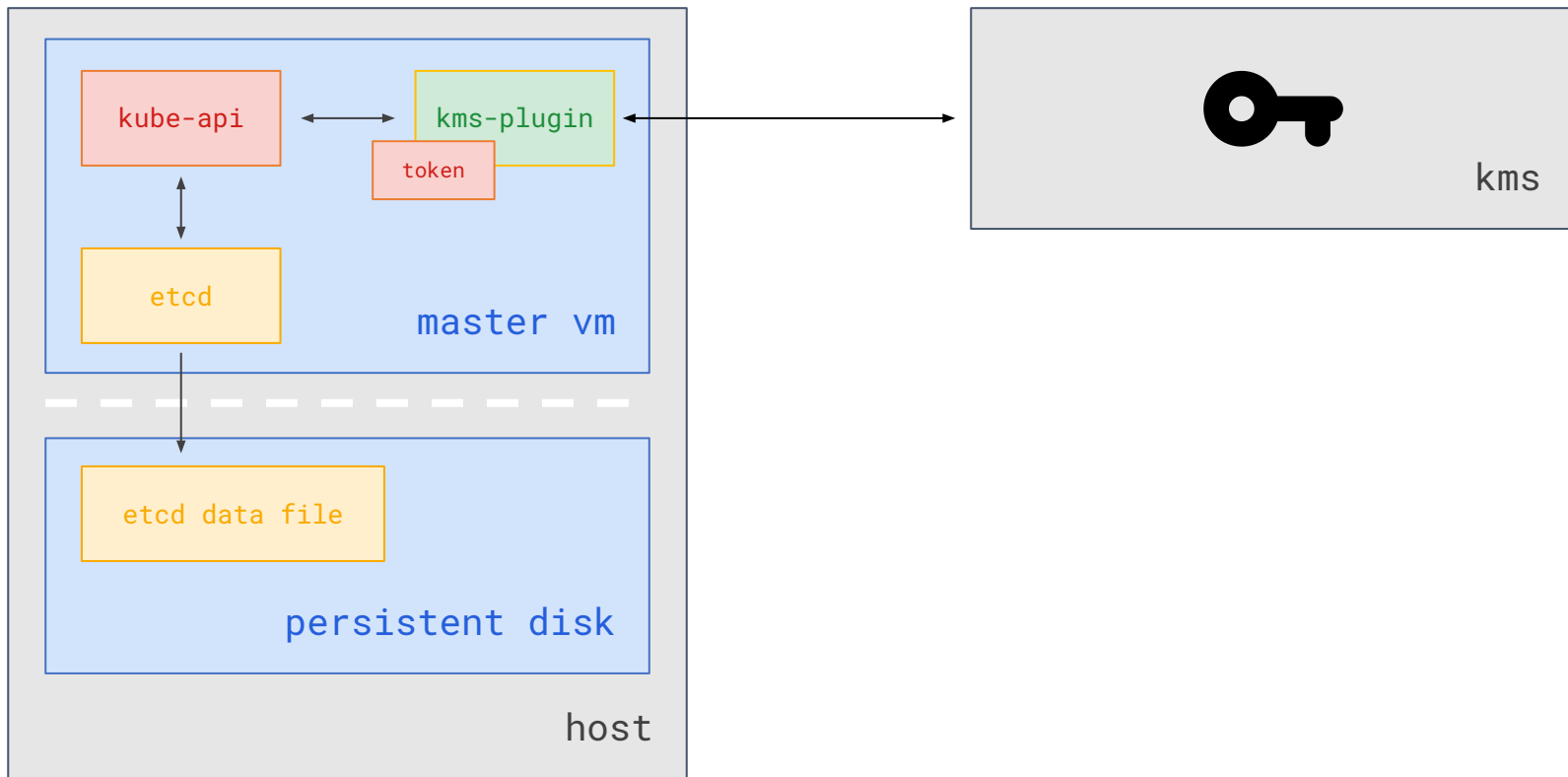
Not 100% solution

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

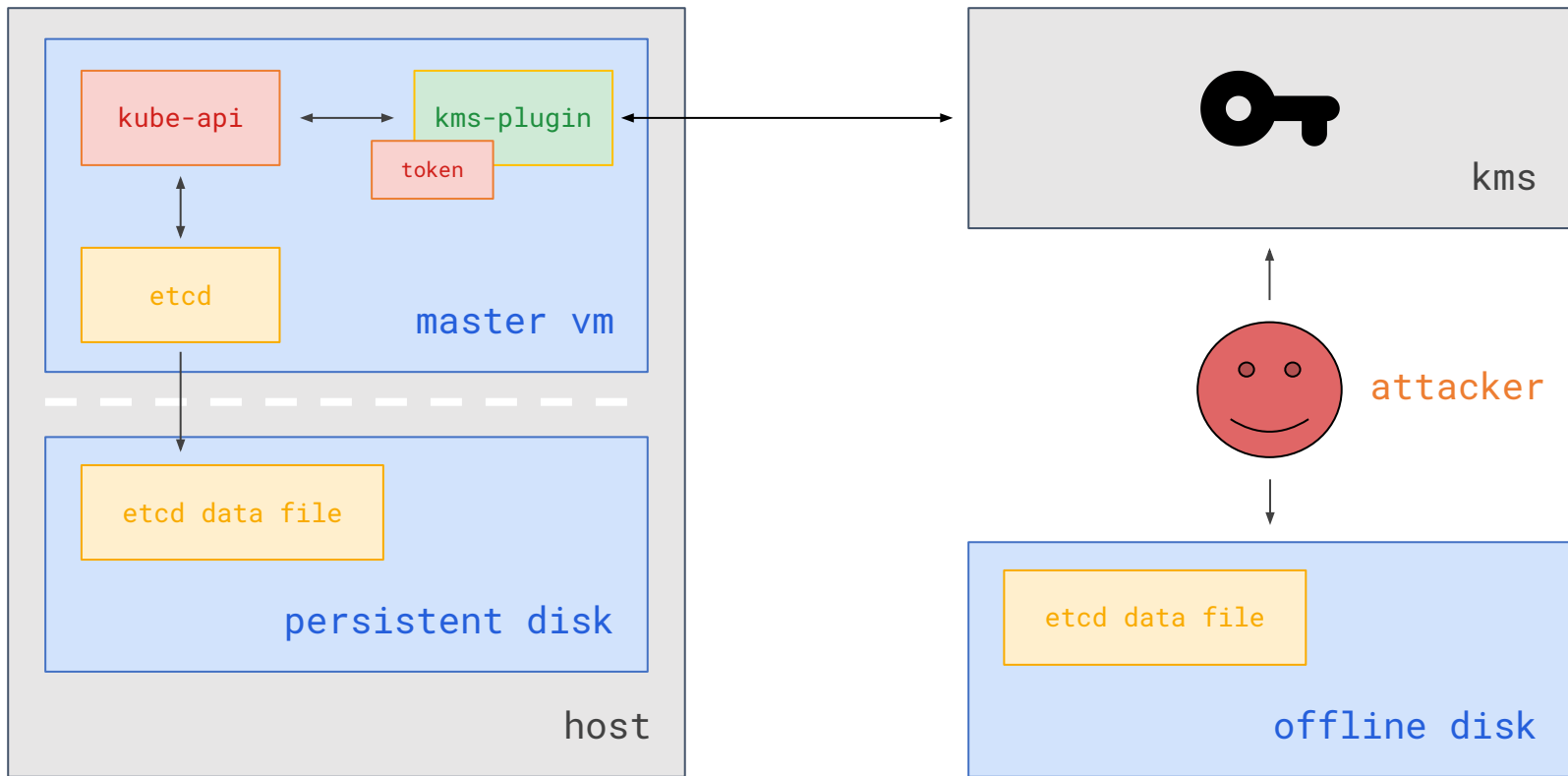
But things are in a much better state!

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

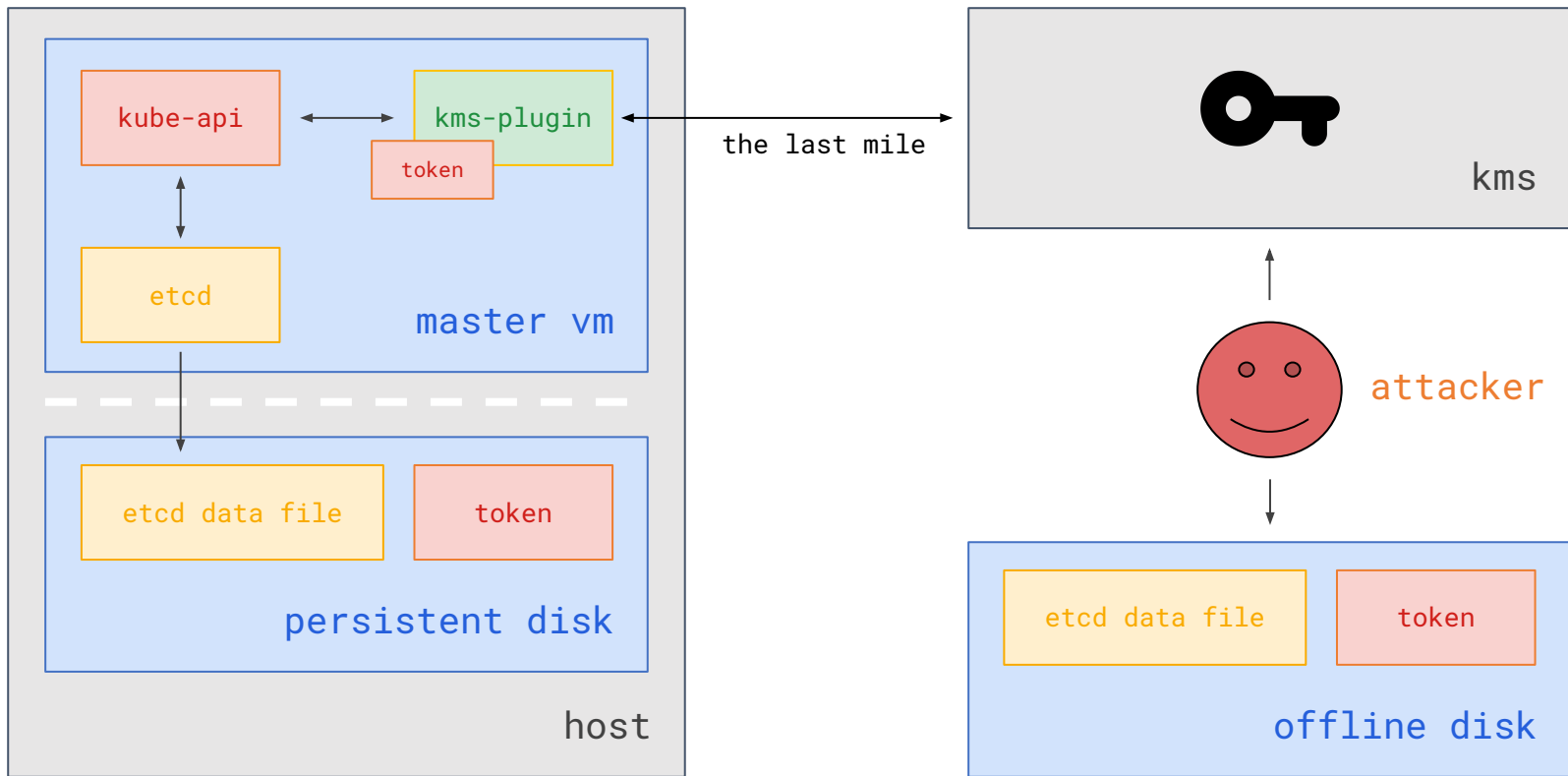
3. Solving the first secret problem



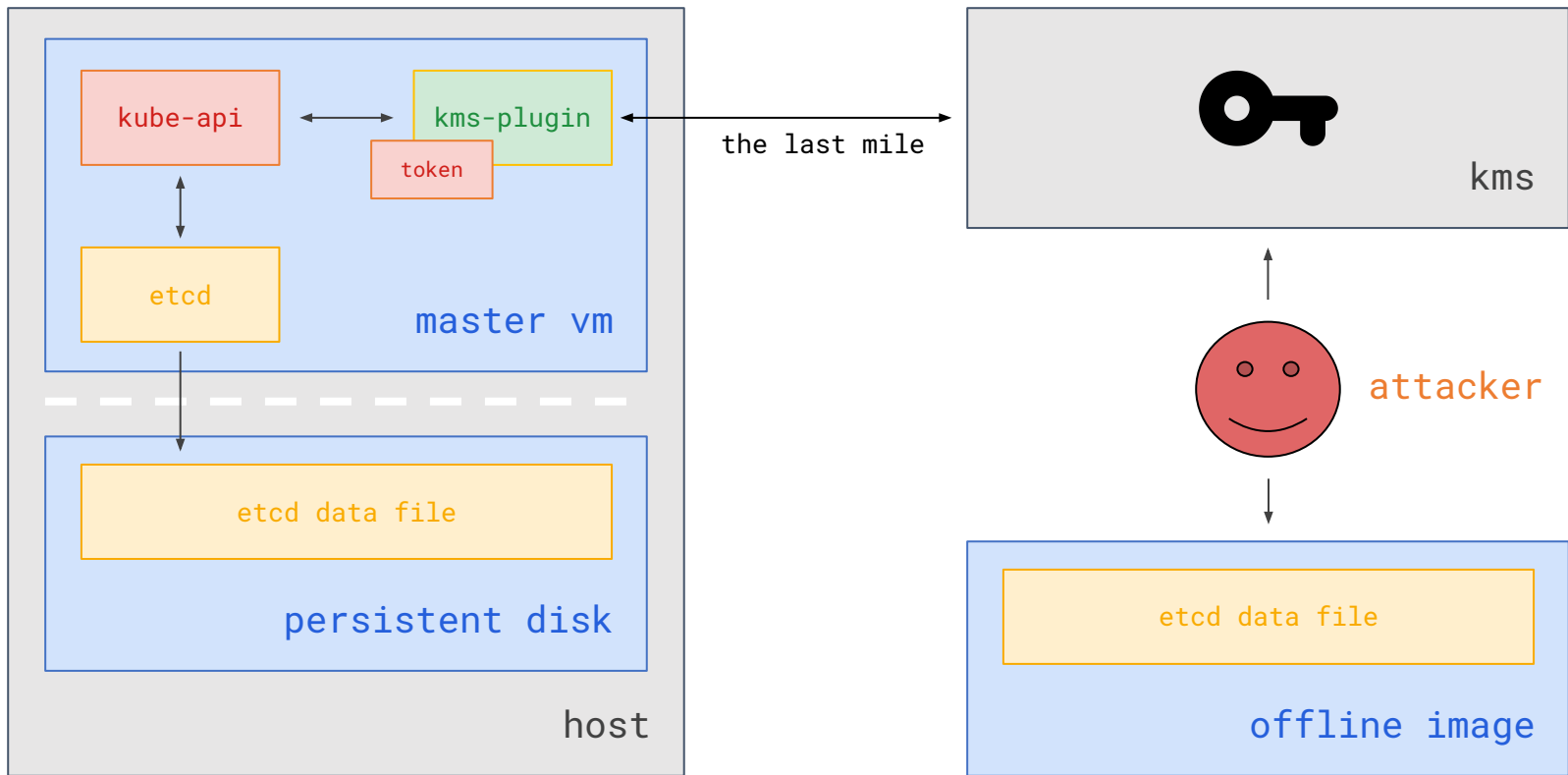
KMS Plugin



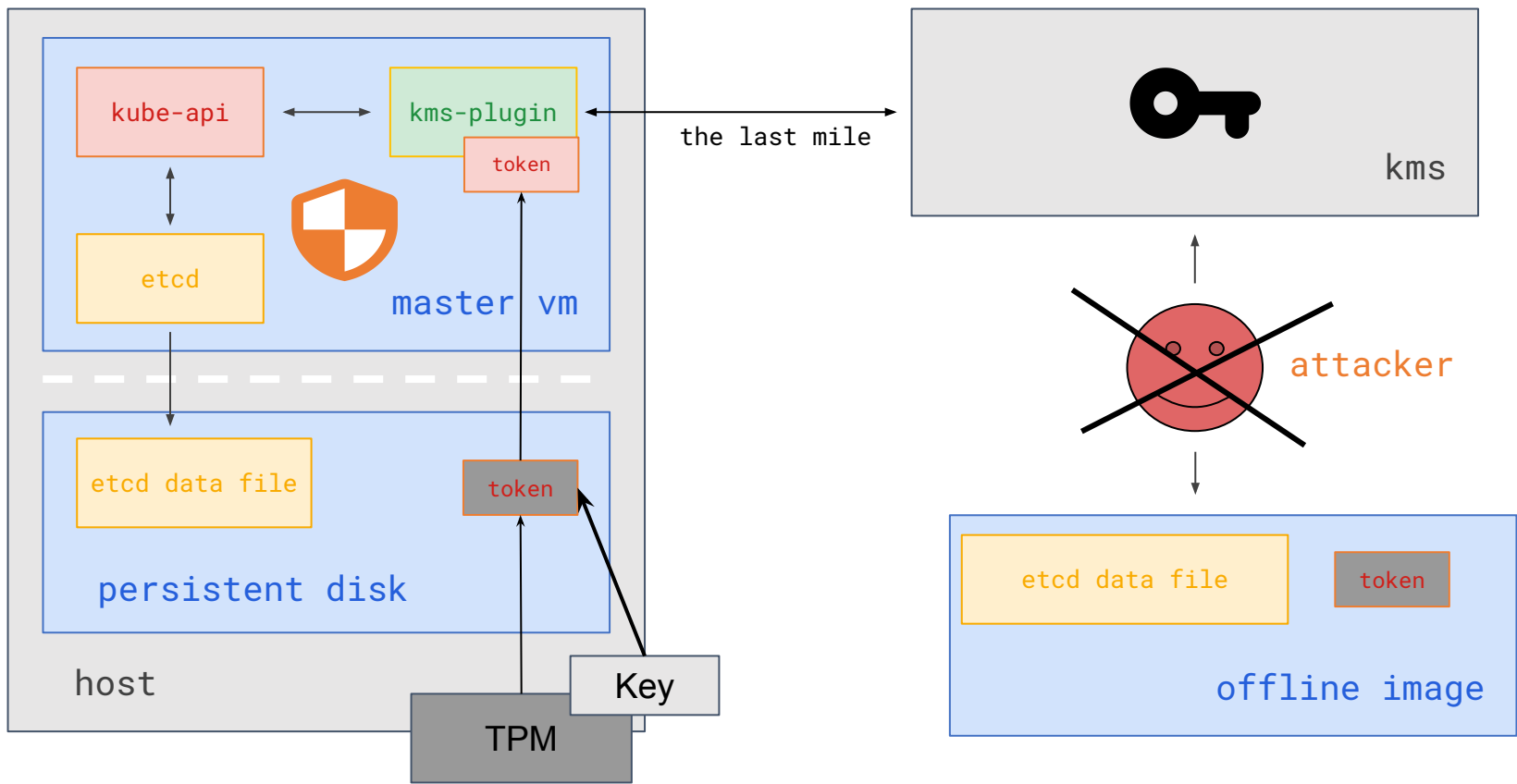
Threat Model



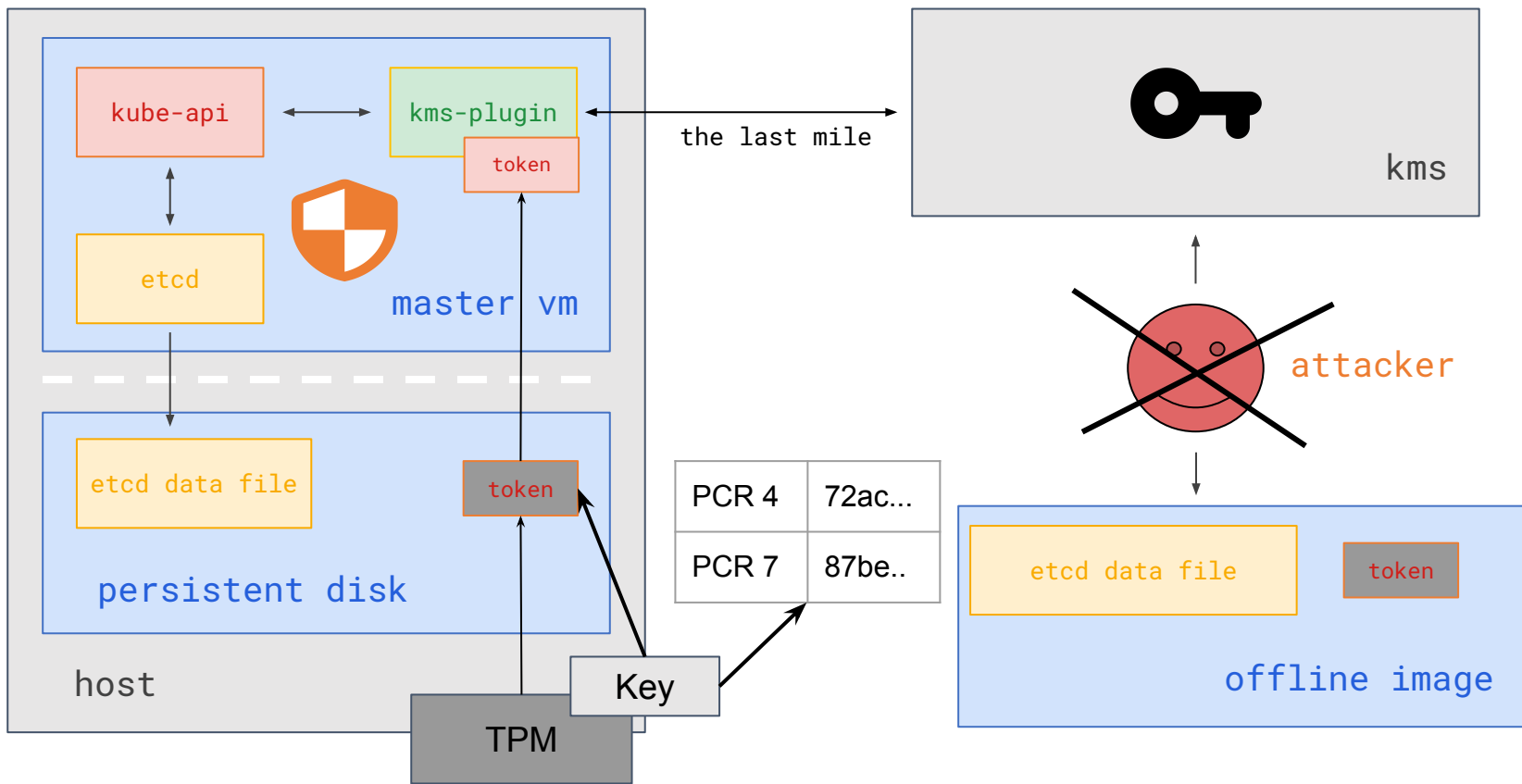
The last-mile problem



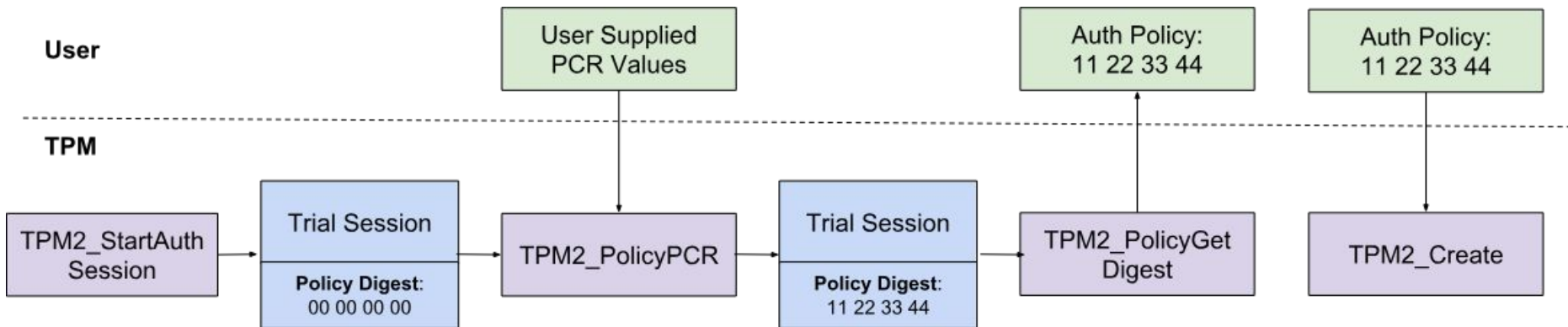
Goal: Do **NOT** get access to keys



Solution: Seal KMS Credential to TPM

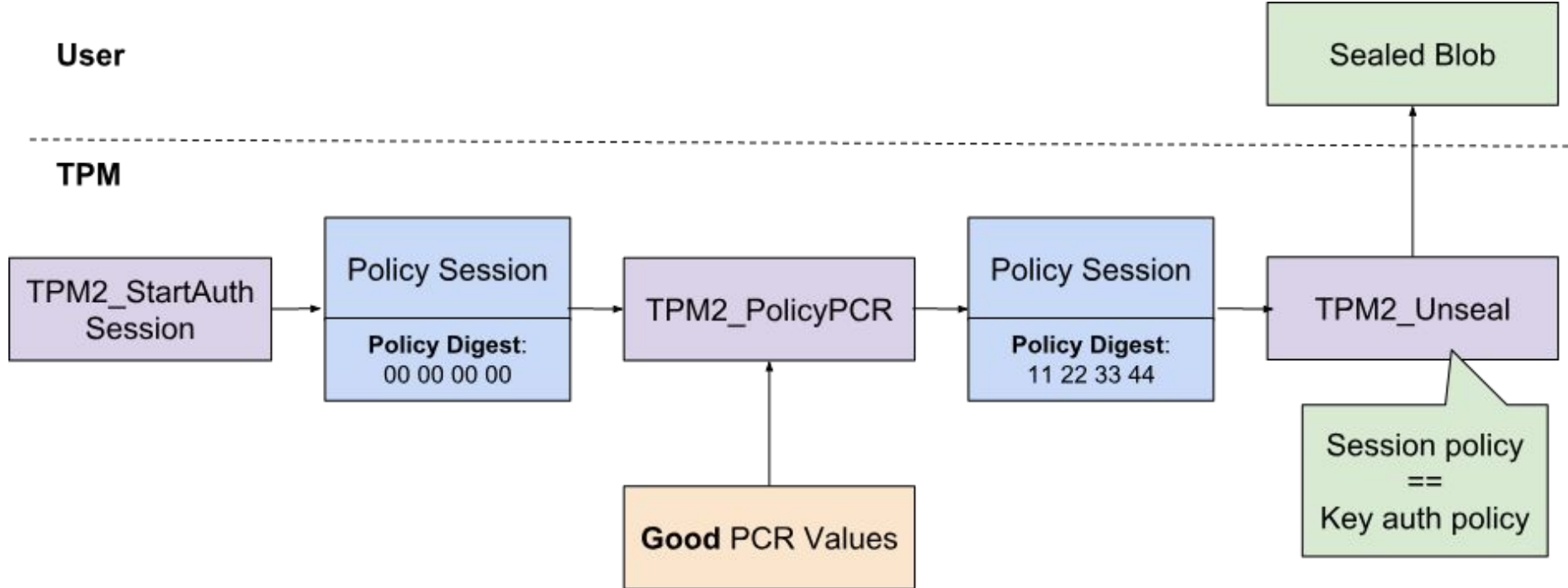


Sealing to PCR Values

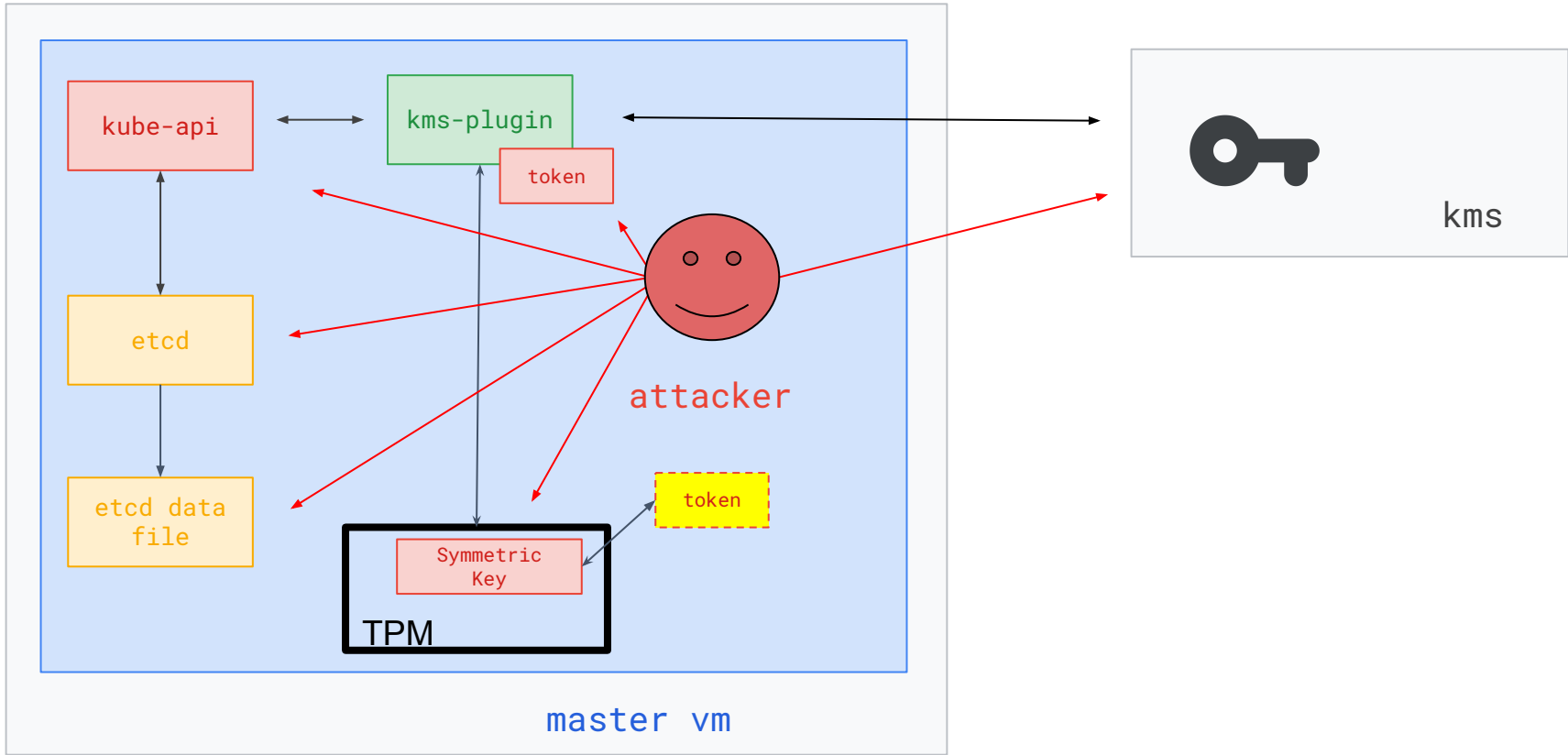


source <https://google.github.io/tpm-js>

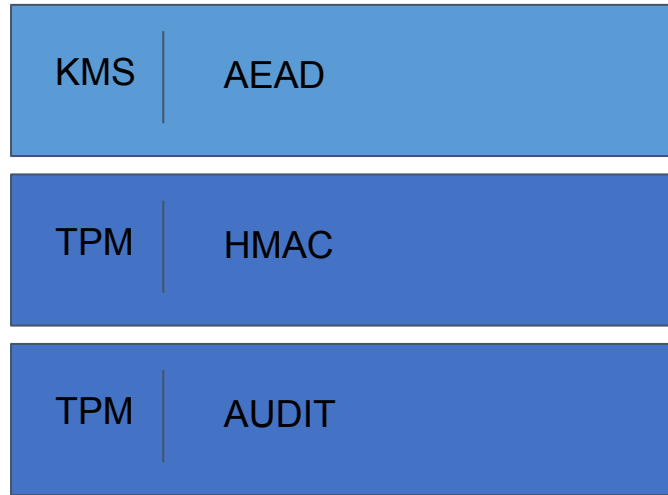
Unsealing



4. Tamper-evident audit logs



Threat Model



Building Blocks

$$\text{audit}_{\text{new}} = \mathbf{H}_{\text{auditAlg}}(\text{audit}_{\text{old}} \parallel \text{inputHash} \parallel \text{outputHash})$$

Auditing TPM Commands

Issued certificate #1:

Certificate:

Data:

Serial Number:

10:e6:fc:62:b7:41:8a:d5:00:5e:45:b6

pub:

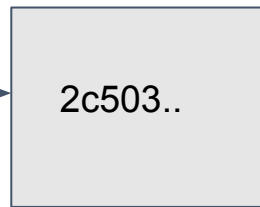
00:c9:22:69:31:8a:d6:6c:ea:da:c3:7f:2c:ac:a5:



TPM2-Sign ...



TPM Audit Register



2c503..

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:

TPM2-Sign ...

TPM Audit Register

9b863..

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:

TPM2-Sign ...

TPM Audit Register

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:

TPM2-Sign ...

TPM Audit Register

8b300..

Logs are examined 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:

TPM Audit Register

TPM2-Sign ... →

9b863... ≠ **8b300**

Logs are examined 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:

TPM Audit Register

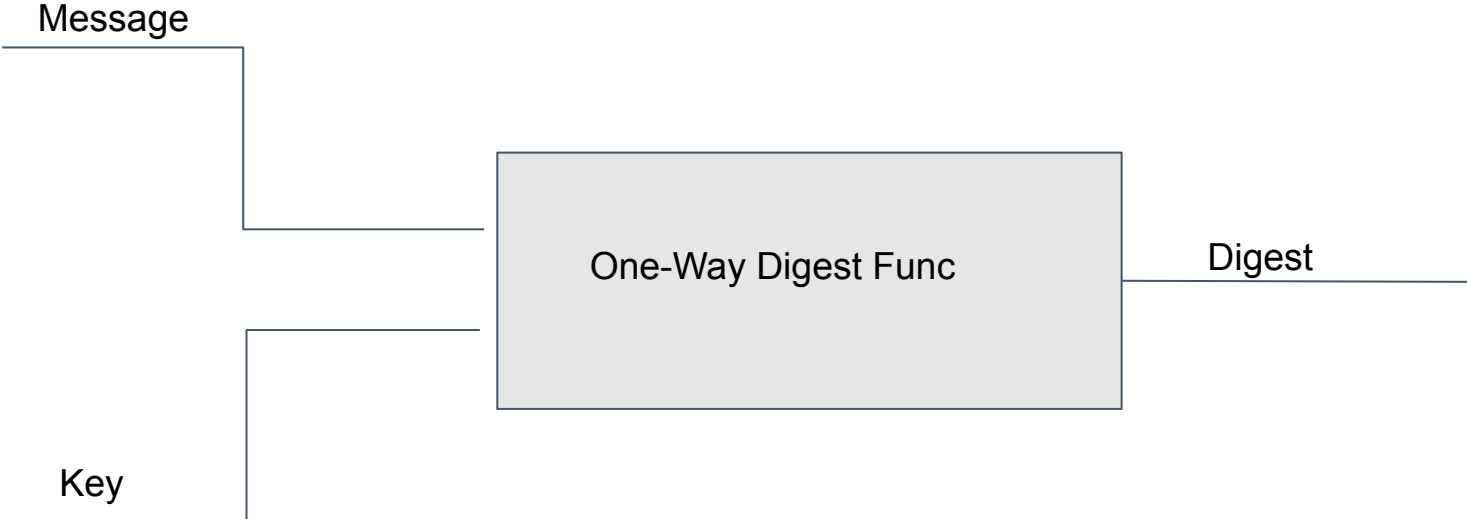
TPM2-Sign ...

Num Of Logs = 2

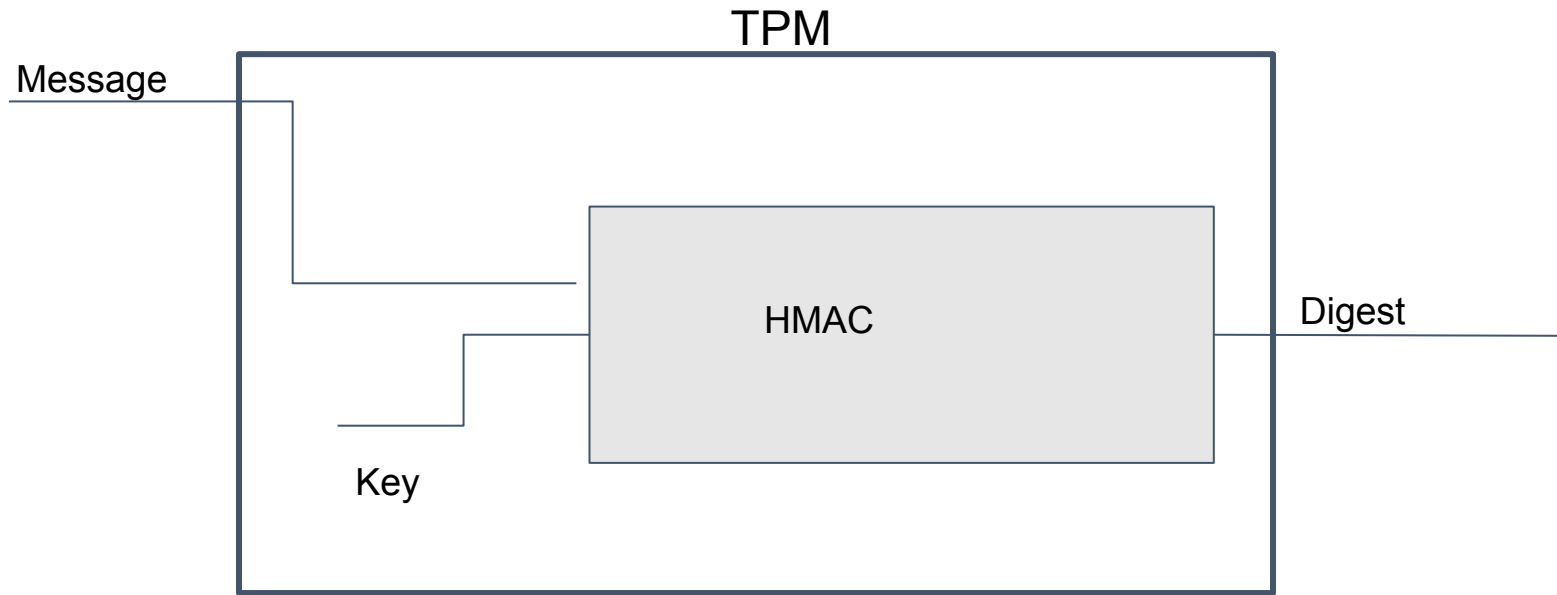
What if attackers reset and replay

KMS	AEAD
TPM	HMAC
TPM	AUDIT 

Building Blocks



Hash-Based Message Authentication Code



TPM



Message

Key

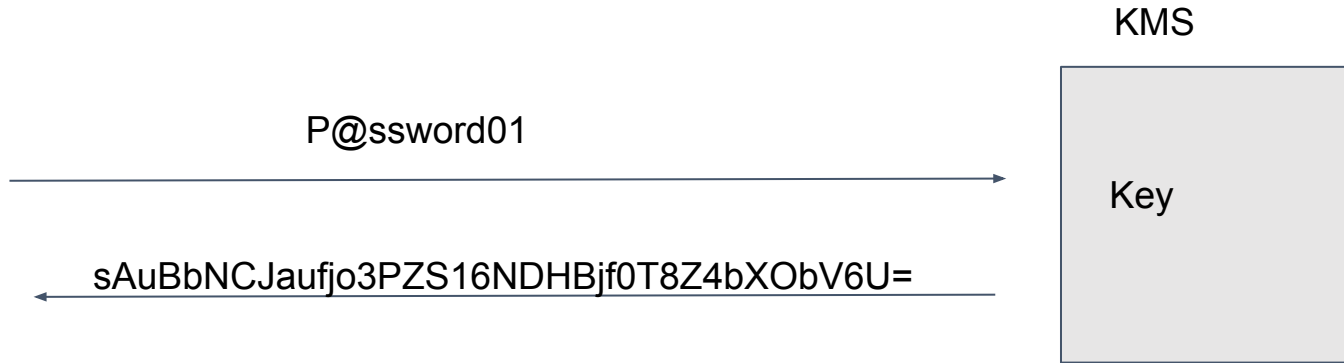
HMAC

Digest

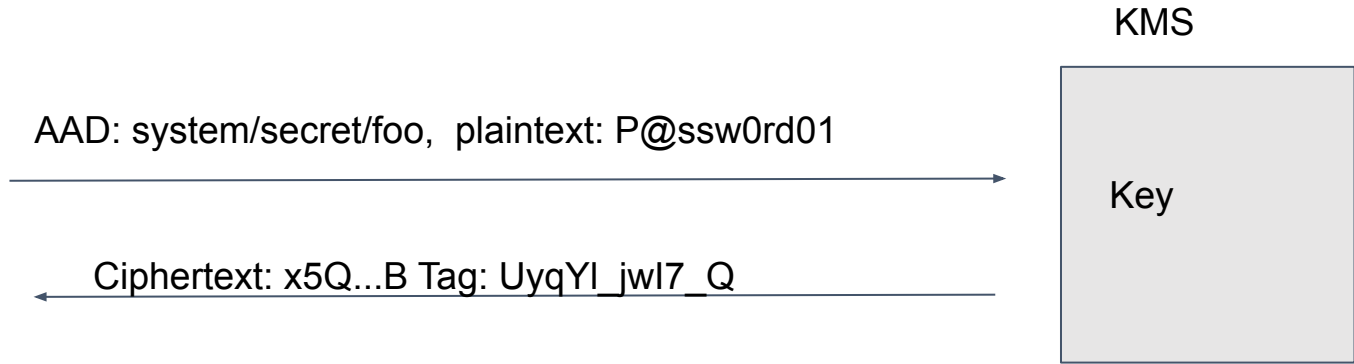
TPM2_HMAC

KMS	AEAD
TPM	HMAC 
TPM	AUDIT 

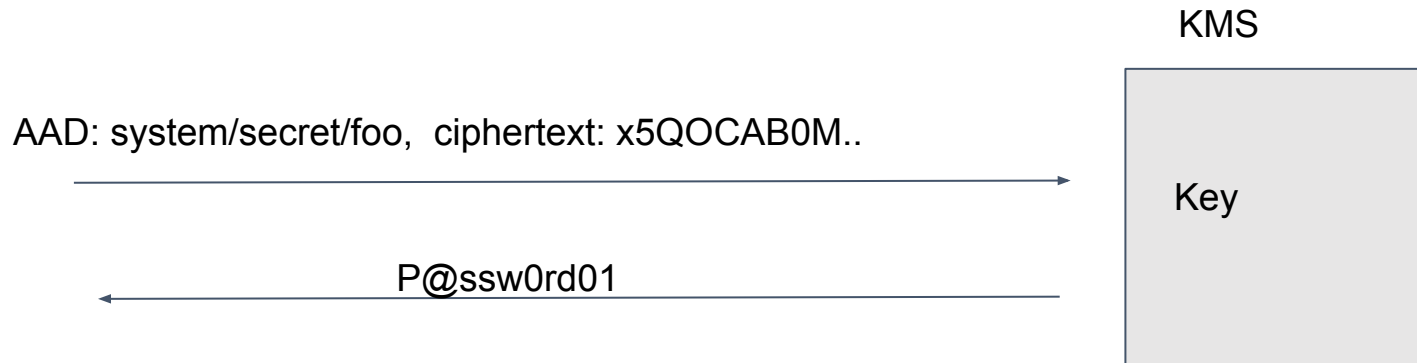
Building Blocks



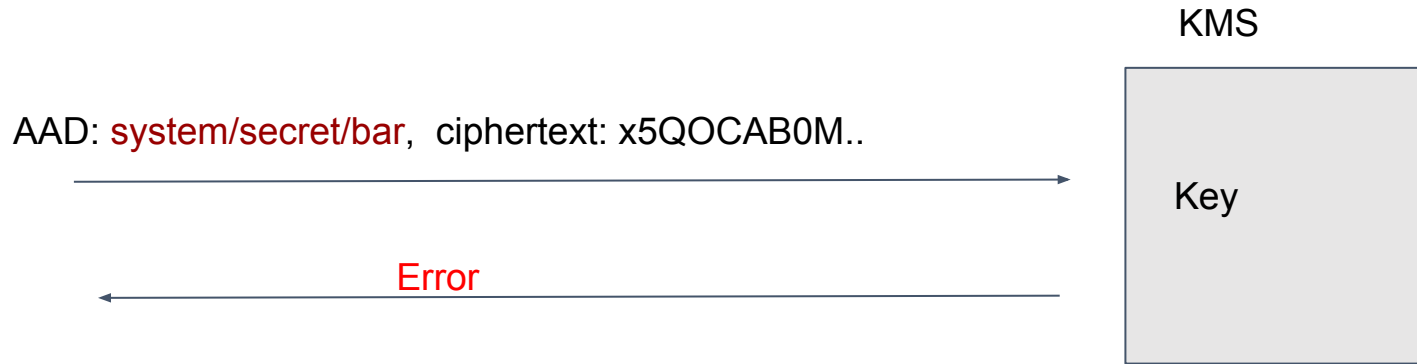
Symmetric Encryption



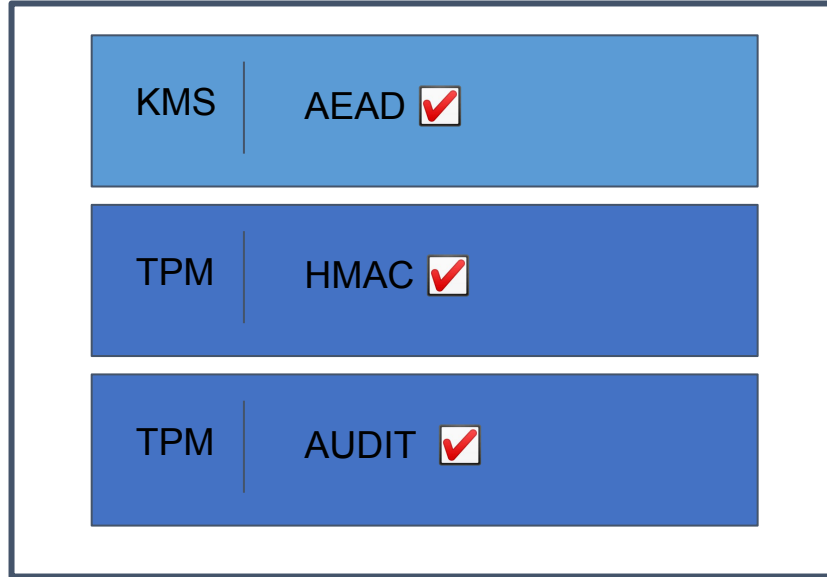
AEAD Encryption



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

On-prem

Cloud Provider

KMS

system/my-api-key
system/my-dba-pwd

proxy

kube-api

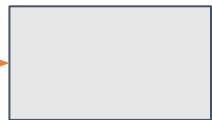
kms-plugin

etcd

TPM2_HMAC

Symmetric Key
TPM

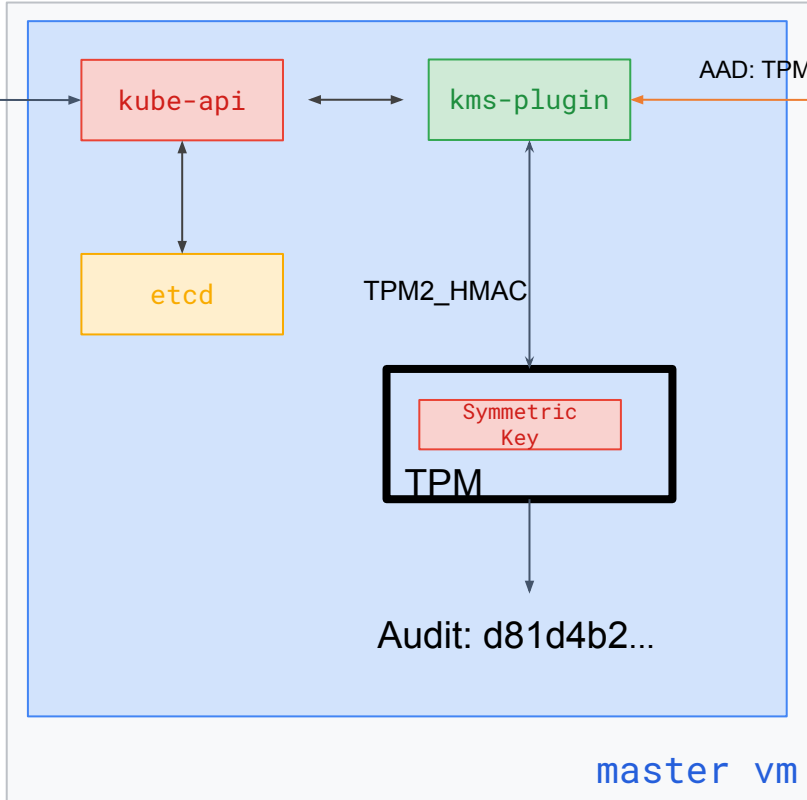
AAD: TPM2_HMAC(KEY)



```
kubectl get secret my-api-key  
kubectl get secret my-dba-key
```



user



master vm

Putting it all together

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



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



KubeCon



CloudNativeCon

Europe 2019

- [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](#)
- [K8S KMS Plugin for Google CloudKMS](#)