Hewlett Packard
Enterprise

Secure State Persistence in Coconut-SVSM

Jean Snyman and Geoffrey Ndu

January 2025

Goals

Provide mechanisms within Coconut-SVSM for secure persistence of CVM state (vTPM NV memory, UEFI variables, etc.) in untrusted hypervisor-provided storage

These mechanisms should make it possible to:

1. Ensure confidentiality and authentication of the persisted data.
2. Use the vTPM to reliably attest the state of a specific CVM instance.

   Including those aspects unique to the operating environment of a CVM, such as whether the CVM has been cloned or had its state rolled back to an earlier version.
3. Continue use of existing applications which depend on a TPM.
4. Run CVMs in edge environments with intermittent connectivity.

Current Status

PR 528 adds the ability for Coconut-SVSM to attest the launch state of a CVM to an external key broker in exchange for a key

With a virtio (or similar) driver added in a future patch, this can be used to achieve state persistence.

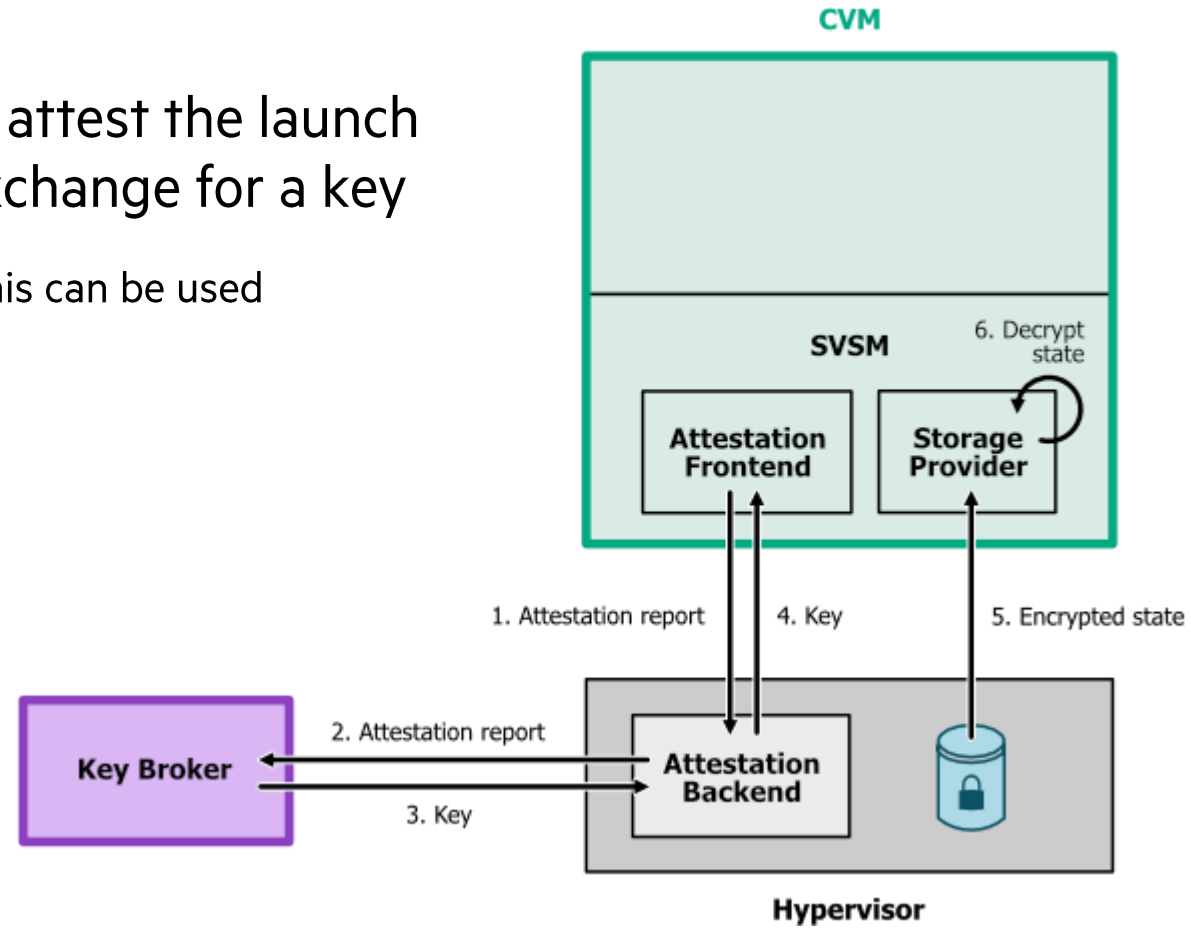
> ⚠ Protocol Security
>
> Because communication is proxied through the hypervisor, the protocol is vulnerable to manipulation.

The protocol is likely not secure for:

- UEFI variable persistence
- TPM attestation
- Storage of arbitrary secrets within the TPM
- Environments with multiple key brokers

The protocol may be secure for:

- FDE with keys appropriately sealed against PCRs

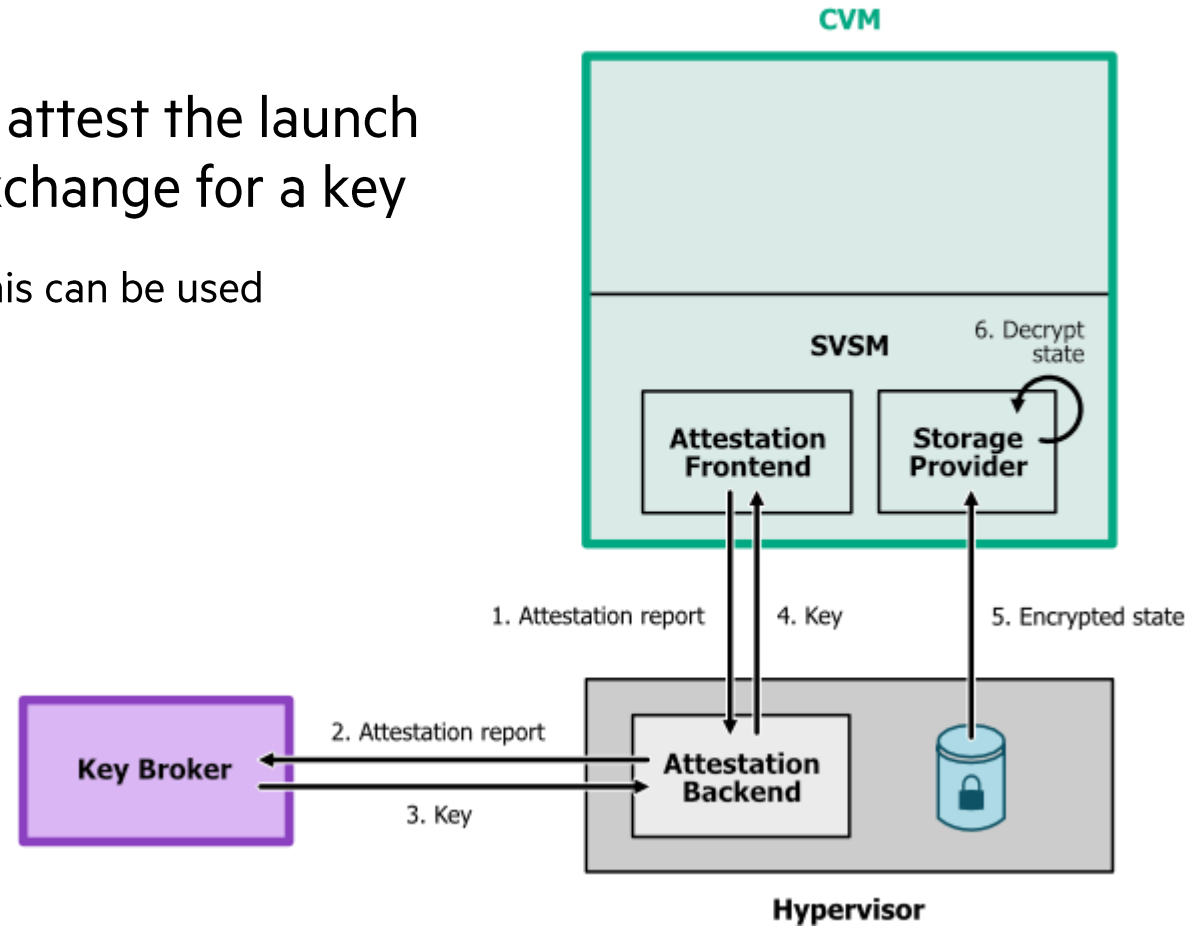


» See Security Analysis at https://stringlytyped.github.io/publications/csvsm-proxy-security-analysis/

Current Status

PR 528 adds the ability for Coconut-SVSM to attest the launch state of a CVM to an external key broker in exchange for a key

With a virtio (or similar) driver added in a future patch, this can be used to achieve state persistence.



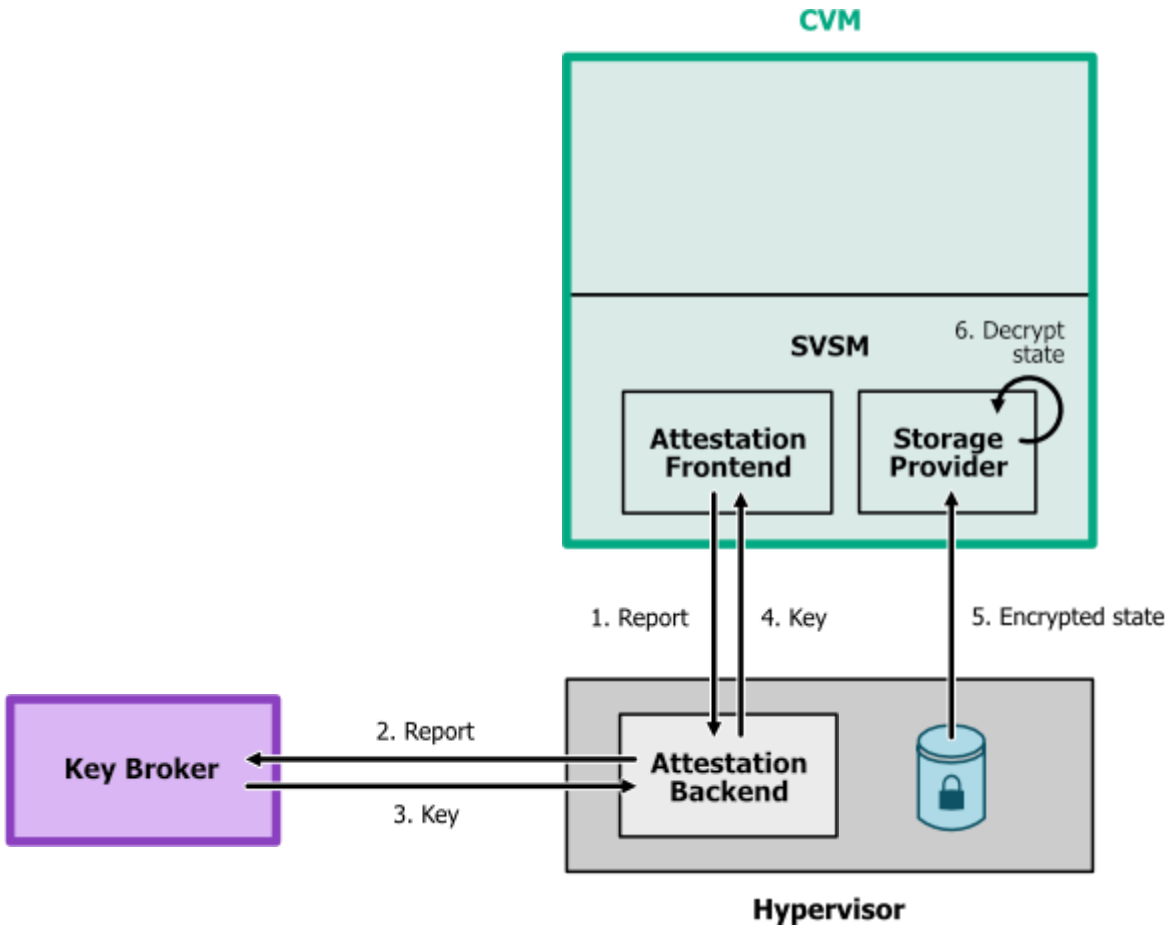
ⓘ Requirements Checklist

1. Ensure ☑ confidentiality and ☐ authentication of persisted data
2. Reliable vTPM attestation of ☐ CVM state, bindable to the ☐ specific CVM instance
3. ☑ TPM backwards compatibility
4. ☐ Bootable without a network connection

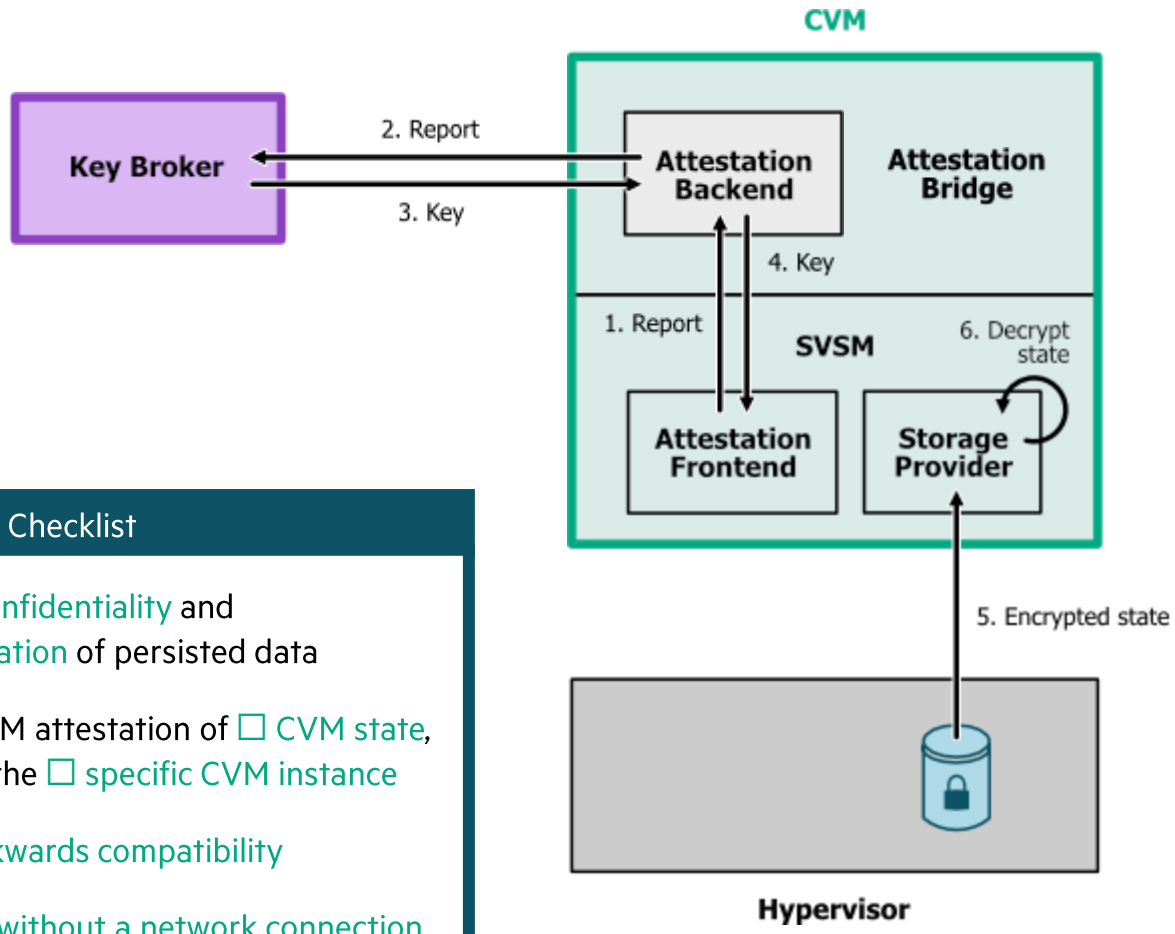
See Security Analysis at

» https://stringlytyped.github.io/publications/csvsm-proxy-security-analysis/

Simple Solution



Simple Solution



ⓘ Requirements Checklist

1. Ensure ☑ confidentiality and ☑ authentication of persisted data
2. Reliable vTPM attestation of ☐ CVM state, bindable to the ☐ specific CVM instance
3. ☑ TPM backwards compatibility
4. ☐ Bootable without a network connection

During boot of the CVM, Coconut-SVSM invokes the **attestation bridge** to attest the launch state of the TEE and retrieve the state decryption key.

Candidates for **attestation bridge**:

- Customised UEFI firmware
- UEFI application
- Minimal service OS

⚠ Must be part of the launch measurement

Should ideally run at a **lower** VMPL than the OS and a **higher** VMPL than the SVSM

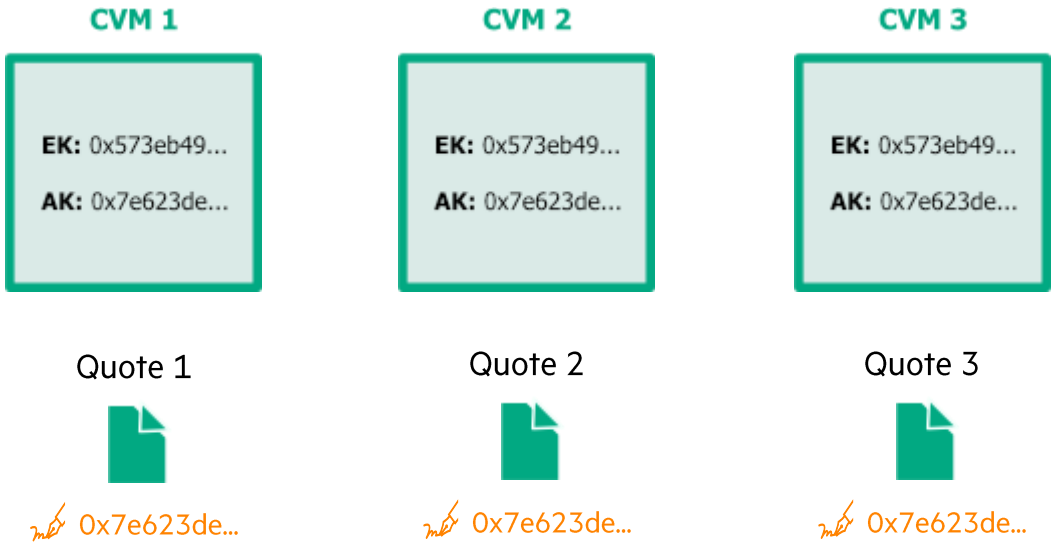
PCRs are retained on state decryption.

vTPM Duplication Issue

Ephemeral vTPM:

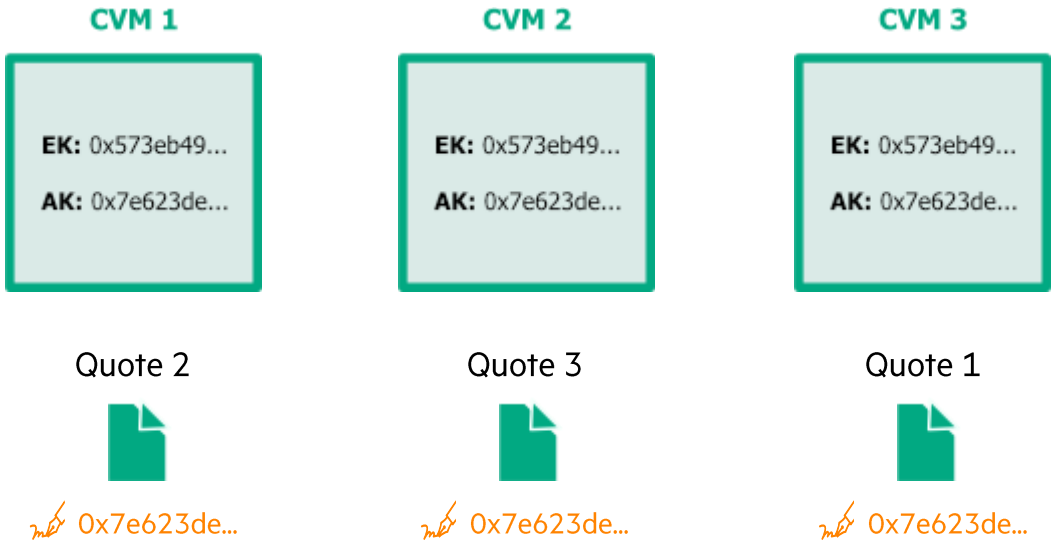
vTPM Duplication Issue

Persisted vTPM:



vTPM Duplication Issue

Persisted vTPM:

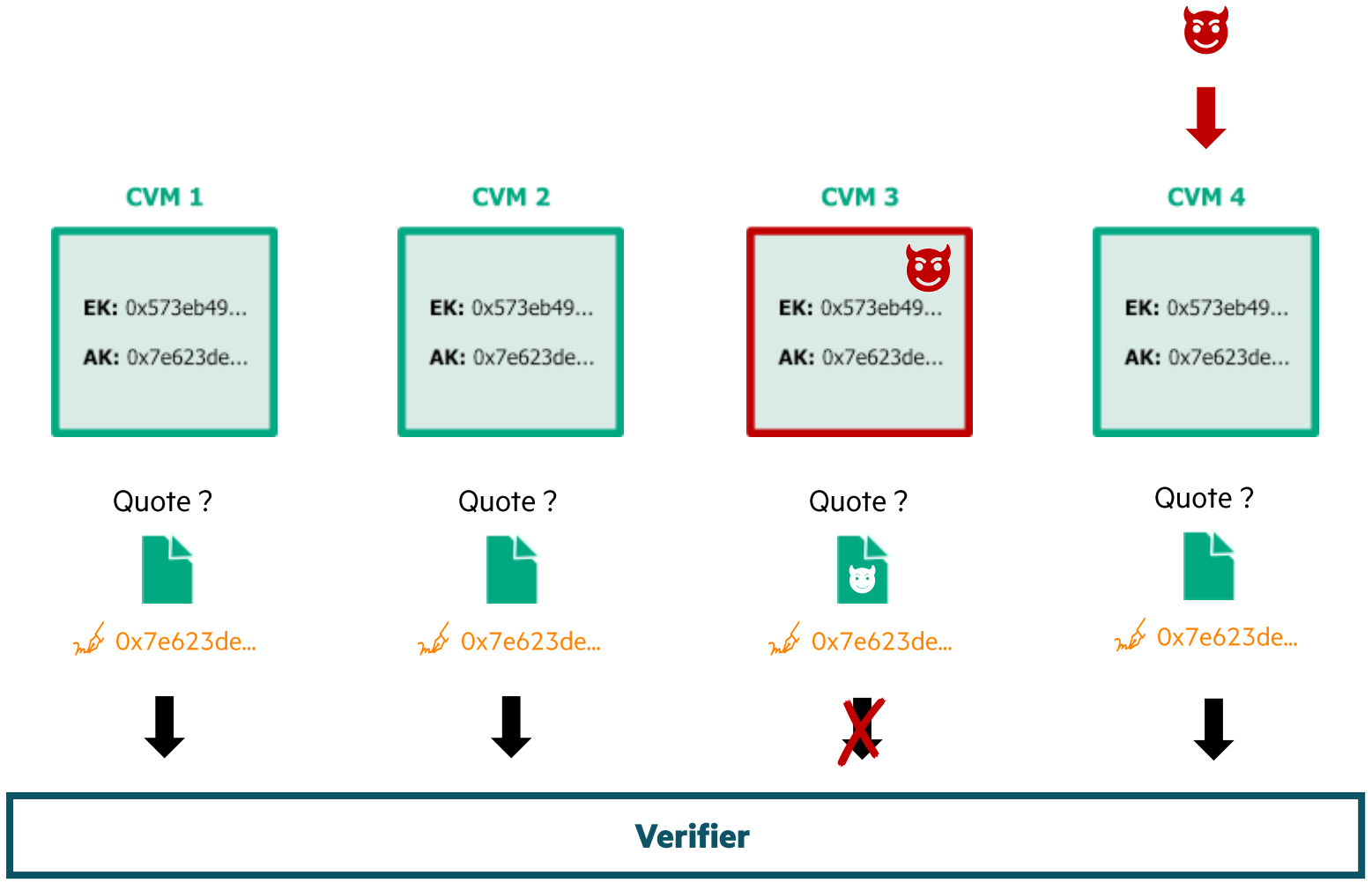


vTPM Duplication Issue

Persisted vTPM:

vTPM Duplication Issue

Persisted vTPM:



vTPM Duplication Issue

Takeaway:

A hybrid approach is required with persisted keys used alongside ephemeral keys.

⚠ Primary Seeds

It may be tempting to exclude the vTPM's primary seeds from the data which is persisted.

However, many applications today assume the presence of a long-lived EK.

TPMs already have a mechanism for loading additional EKs.

Solution:

On state decryption, Coconut-SVSM issues a **TPM2_CreatePrimary** command to create an additional EK under the Endorsement Hierarchy with a seed randomly chosen by Coconut-SVSM.

vTPM Duplication Issue

Solution:

On state decryption, Coconut-SVSM issues a **TPM2_CreatePrimary** command to create an additional EK under the Endorsement Hierarchy with a seed randomly chosen by Coconut-SVSM.

ⓘ Requirements Checklist

1. Ensure ☑ confidentiality and ☑ authentication of persisted data
2. Reliable vTPM attestation of ☐ CVM state, bindable to the ☑ specific CVM instance
3. ☑ TPM backwards compatibility
4. ☐ Bootable without a network connection

Unique Properties of CVMs

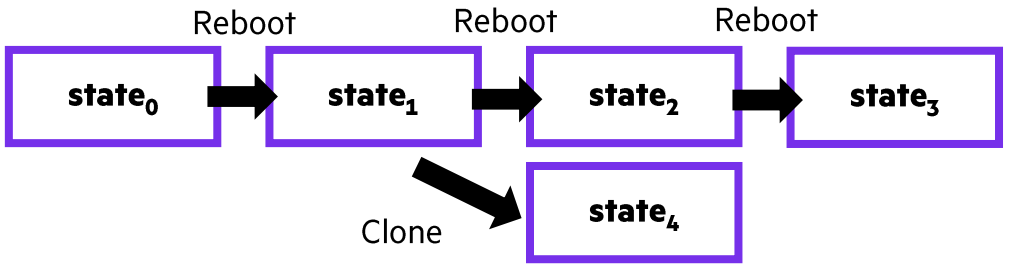
CVMs are fundamentally different from physical systems.

A physical system may be stopped and started but there exists only a single instance for the lifetime of the system.

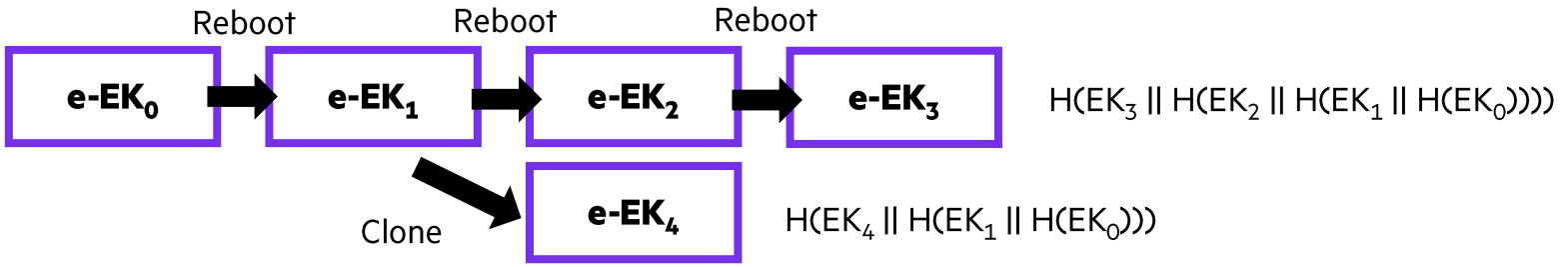
A virtual system may be instantiated any number of times.

- Across reboots, VM instances use the same disk image.
- However, earlier snapshots of the disk image may be retained.
- A VM may be cloned from the latest disk image or an earlier snapshot, which creates a copy of the disk image.
- Therefore, an instance's state is linked to the state of the instances which came before it.

Unique Properties of CVMs



Unique Properties of CVMs



Unique Properties of CVMs

Persisting a rolling hash of the e-EK in NVRAM (using **TPM2_NVExtend**) provides a history of CVM instantiations.

This can be combined with a counter value, also kept in NVRAM, which is increased each time the persisted state is updated.

The instantiation history and counter can be attested using **TPM2_NVCertify** to detect unauthorised cloning and state rollback.

ⓘ Requirements Checklist

1. Ensure ☑ confidentiality and ☑ authentication of persisted data
2. Reliable vTPM attestation of ☑ CVM state, bindable to the ☑ specific CVM instance
3. ☑ TPM backwards compatibility
4. ☐ Bootable without a network connection

Intermittent Network Connectivity

We also have a solution which uses processor-derived keys in place of a key broker for protection of state.

It is more complicated than the above, because we need to handle upgrades to the SVSM, firmware, etc. which change cause a change in the derived key.

This enables use in environments without reliable connectivity.

Thank you