3.1 Attested Launch

In this thesis, "Attested Launch" is a concept that relies upon Remote Attestation to ensure that a given system may only be started if a given Remote Attestation mechanism is carried out successfully and makes it inoperable any other way, through the late provisioning of critical cryptographic material.

In this Proof of Concept, this will be achieved by storing the system (Ubuntu 20.04)'s root partition behind a LUKS encrypted disk for which the key will only be transmitted upon a successful RA mechanism run. In the event of a RA failure, no key shall be delivered, and the system will not be able to start.

3.1.1 Attested Launch Objectives

The sole objective of this protocol is to ensure that key cryptographic material required for the operation of the system is withheld. The system is thus required to request such material from a third party every time it attempts to start. In doing so, it also requires that the cryptographic material may not be stolen by unrelated third parties. Indeed, if the cryptographic material can be stolen, the third party may be bypassed.

It also requires implementing replay protection, such that a captured communication between the VM and third party may be used to bypass future connections. As such, it must also ensure that only trusted code may run within the system. If any code from an untrusted third party is run within the system, the cryptographic material may be stolen.

3.1.2 Attested Launch Remote Attestation Requirements

From those above-mentioned objectives, we can already establish some requirements for the underlying remote attestation mechanism. The aforementioned mechanism must be able to attest to the initial state of a system and not only an application. It must also be able to carry arbitrary data such that replay protection may be implemented in the form of a nonce within the attestation report. Furthermore, it must be possible to ensure that only code belonging to the VM owner may run within the VM, or at the very least, that a strong chain of trust may be established from the initial VM image provided by the host (and properly audited), up to the OS level.

3.2 Remote Attestation Mechanism Selection

Of the three remote attestation mechanisms investigated in depth, only two of them are suitable for this thesis. Intel TDX was not available for the majority of the thesis' timeframe; as such, it was also non-applicable to use as a basis to design the mechanism. However, it should be noted that Intel TDX is the only mechanism that is able to protect from a somewhat complex threat studied further in section 3.4.5.

There is the choice of AMD SEV(-ES) and AMD SEV-SNP, both with their positives and negatives.

However, from the point of view of adapting this Proof of Concept to a public cloud, it makes no sense to use AMD SEV(-ES) with all the restrictions and issues it brings to public clouds, as seen in the analysis of Google Cloud Platform in subsection 2.4.1. Indeed, since no cloud provider allows third parties to be the VM owners of AMD SEV(-ES) VMs, there is no possibility of establishing a shared secret between the VM and ourselves or even making use of an attestation report. If we were to ignore this glaring issue, there would still be the vast improvements brought by AMD SEV-SNP that would still cement the decision to ignore SEV(-ES).

As such, this proof of concept shall be done on the SEV-SNP remote attestation platform, with some considerations taken to avoid preventing its implementation with Intel TDX if possible.

3.2.1 Bootstrapped Chain of Trust

In order for this method of achieving an attested launch to remain secure, we must first ensure that a chain of trust can be established all the way from the initial VM image (which will be measured with the remote attestation mechanism), all the way until the attested launch process is run. This would allow ensuring that only trusted code has been run up to the point the attested launch process is underway.

This design is meant to provide an attested launch mechanism for a KVM/QEMU-based VM running on a host with SEV-SNP enabled; as such, we can expect the following chain of trust:

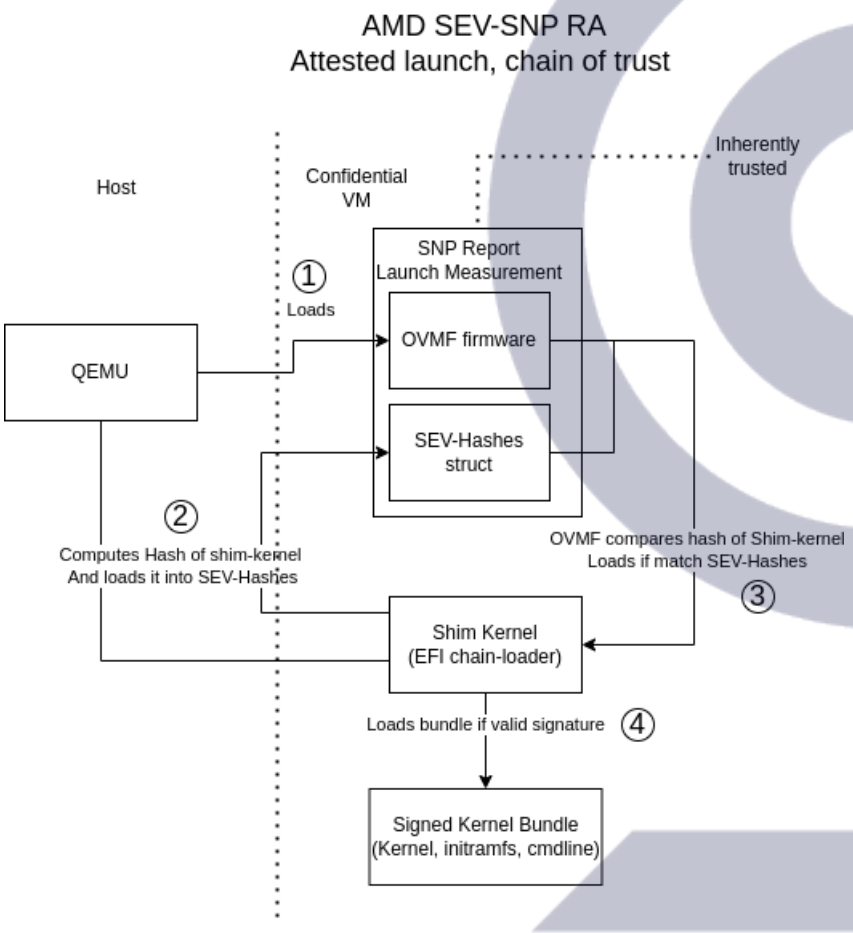

Figure 3.1: Proof of Concept Attested Launch Trust Chain

The initial chain anchor will be based on an AMD SEV-SNP report of the initial VM contents.

QEMU will load the OVMF firmware along with the SEV-Hashes structure (The firmware itself and SEV-Hashes, will be measured during the attestation)

OVMF will then rely upon the SEV-Hashes structure to load and execute an EFI stub that matches its expected hashes.

From there, the EFI stub will contain the relevant public key to verify the signature of the EFI bundle containing the relevant kernel, initramfs, and cmdline.

As such, the inherent trust in the AMD Attestation Report Launch Digest, which covers the contents of the OVMF firmware and EFI stub hash, can be bootstrapped to the EFI stub, assuming that OVMF correctly loads and verifies the EIF stub. From there, the trust can be bootstrapped into the Signed Kernel Bundle, assuming the efi stub correctly validates the bundle's signature before loading it.

As such, when transmitting the attestation report to a third party as part of the initramfs boot process, the third party can be assured that only trusted code was executed in the VM prior to that transmission.

3.3 Attested Launch Protocol Overview

This protocol relies on the following entities :

- **Guest VM**

  AMD SEV-SNP VM instance that is trying to prove it has not been tampered with to request cryptographic secrets.

- **Verifier**

  The entity holding the required cryptographic secrets, provisioning them to the Guest VM if it can assert it has not been tampered with.

For the protocol to be useable with multiple VM instances simultaneously, it must be possible to segregate the LUKS secret between instances.

In order to do so, a 32-byte identifier will be used during the protocol.

Due to the nature of using a given firmware image with multiple identifiable instances, the identifier will necessarily be visible by the VM host (thus CSP) and assignable to any instance.

The security implications of such requirements will be studied in the threat model.

3.4 Attested Launch Threat Model

In the context of this attested launch mechanism, the most important asset to secure is the cryptographic payload associated with a given identifier (the LUKS key that unlocks the instance's disks) which will be the focus of this threat modeling.

This threat model will exclusively focus on the Attested Launch protocol and not the underlying Remote Attestation protocol security and threat model itself. It is assumed that AMD will take the necessary steps to ensure the security of SEV-SNP through security advisories and updates to relevant firmware blobs.

3.4.1 Threat Model Overview

A high-level overview of the threat model can be found within Figure 3.2.

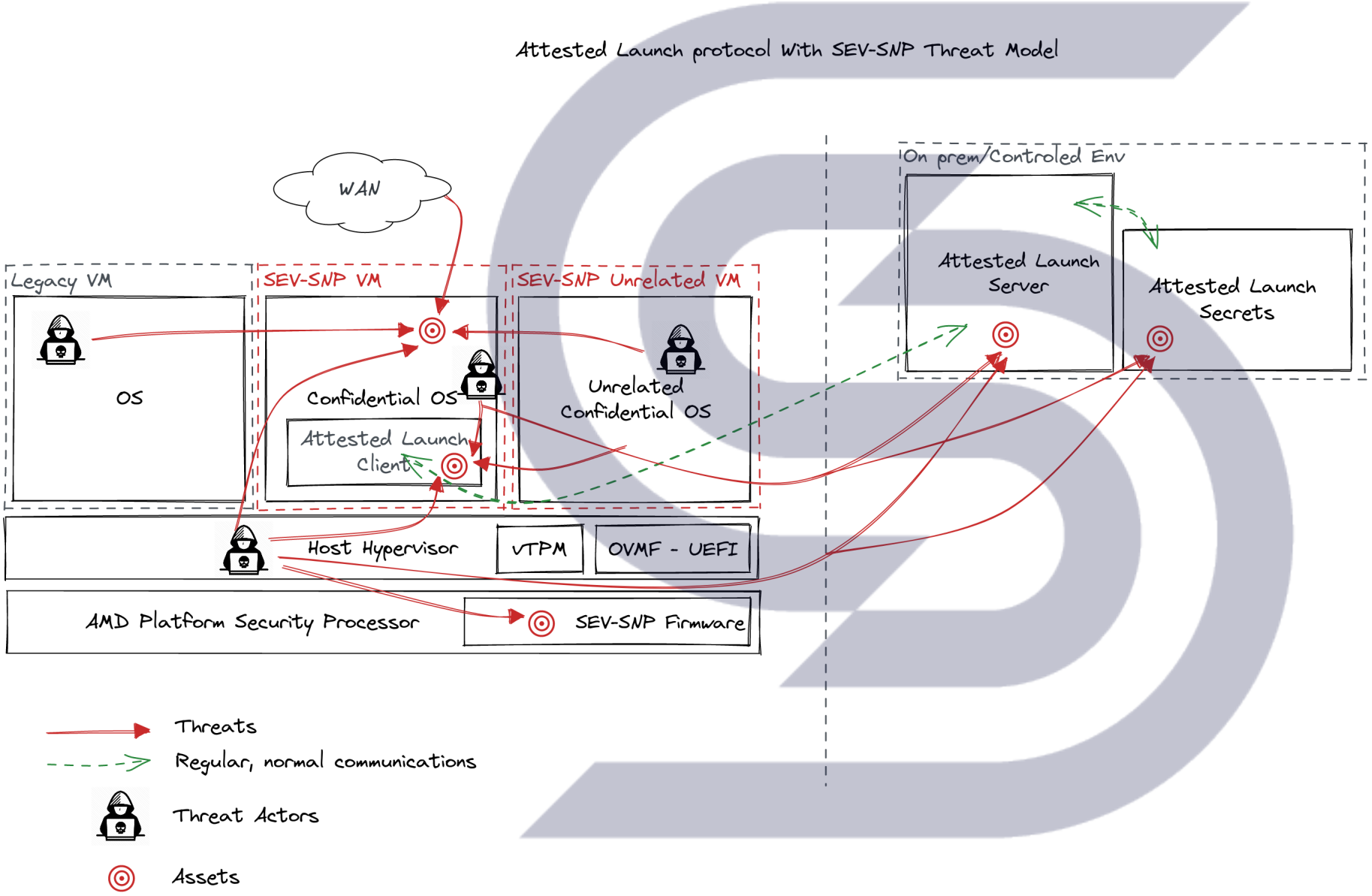

Figure 3.2: Threat Model of Attested Launch on AMD SEV-SNP

3.4.2 Trust Boundaries

This threat model considers two distinct boundaries:

- **On Premise, Controlled Environment**

  An area that contains the Attested Launch server as well as its secrets. This area must be quite significantly hardened and protected, as it contains the most important assets.

- **VM Host, Uncontrolled Environment**

  This is an area that contains most of the platform upon which the untrusted part of our Attested Launch protocol will run. This area contains many threat actors that may be colluding with each other; as such, the protocol needs to be designed to be resilient against this environment.

3.4.3 Assets

This threat model considers the following assets (with shorthand forms) to protect:

- **Confidential OS (COS)**

  The OS was launched with the help of the Attested Launch protocol. It must be protected from any other unrelated VMs, SEV-SNP protected or not, from the host itself, as well as from the outside world through its internet connectivity.

  This asset is, however, considered outside the scope of the Attested Launch protocol proper.

- **Attested Lauch Client (ALC)**

  The client that runs the Attested Launch protocol within the confidential VM at early boot. It must be protected from interference from the Confidential VM itself, any other SEV-SNP confidential VM, or the host itself.

- **SEV-SNP Firmware (SEV-FW)**

  The firmware that ensures the security of the SEV-SNP-based Confidential VM. It must be protected from interference by the host.

- **Attested Launch Server (ALSer)**

  The server that runs the Attested Launch protocol within the controlled environment. It must be protected from interference by the host or the confidential VM itself.

- **Attested Launch Secrets (ALSec)**

The secrets are distributed by the Attested Launch Server on successful protocol execution. This is the most critical asset of the system. It must be thoroughly protected from interference by the host or the confidential VM itself.

3.4.4 Threat Actors

It also considers the following threat actors:

- **Legacy VM attacker**

  An attacker running within the context of a non-SEV-SNP VM wishes to attack the SEV-SNP VM.

- **Unrelated SEV-SNP VM attacker**

  An attacker running within the context of a SEV-SNP VM wishes to attack the SEV-SNP VM or the Attested Launch client.

- **VM Host**

  The host that starts the AMD SEV-SNP VM instance is in full control of the underlying hardware, communications of the VM Guest, contents of the VM Guest, and relevant SEV-SNP firmware. It wishes to attack the Confidential VM, the Attested Launch Client, the Server, Secrets, and the SEV-SNP firmware.

- **VM Guest**

  Guest that runs within the SEV-SNP VM Confidential OS. It wishes to attack both the Attested Launch Client, Server, and Secrets.

3.4.5 Threat Scenarios

A high-level overview of the threat scenarios that were considered with the threat model can be found within Figure 3.2.

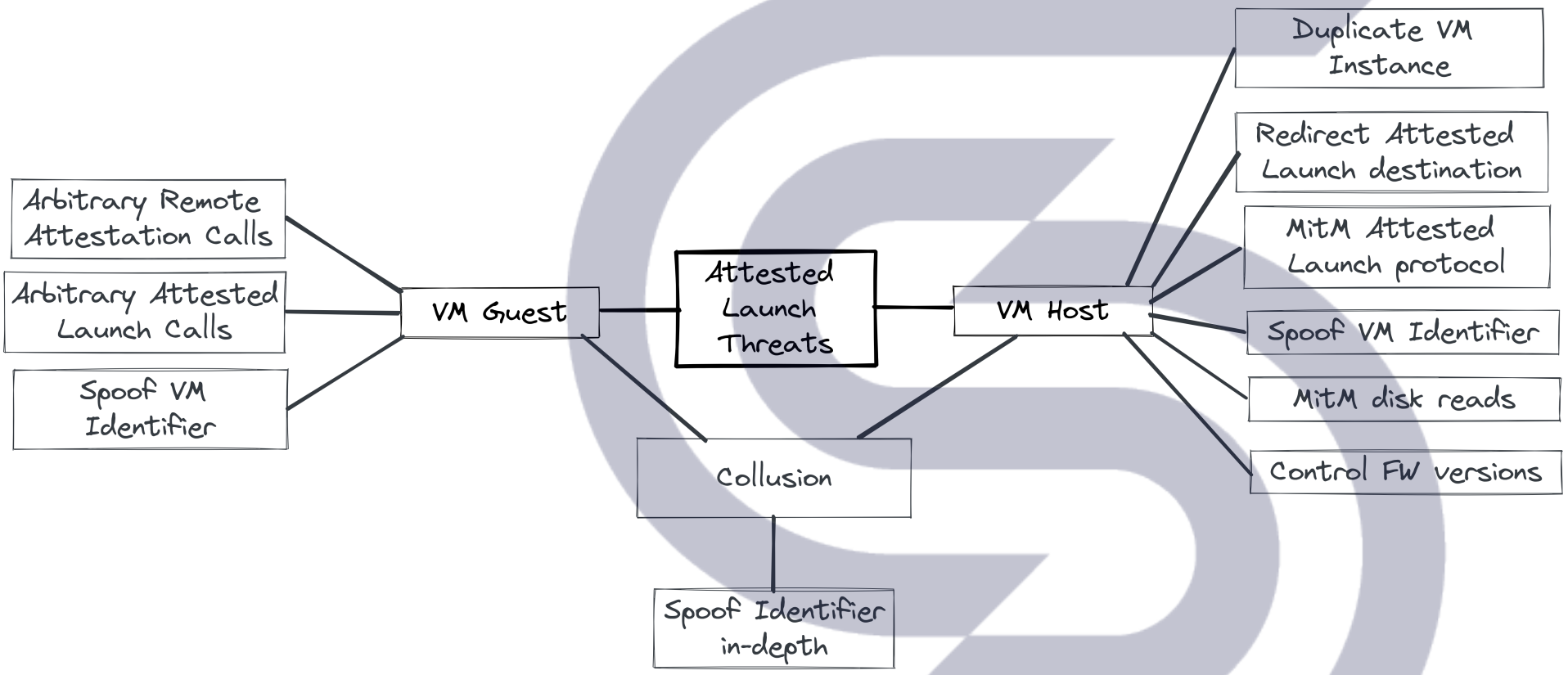

Figure 3.3: Threat Scenarios against Attested Launch on AMD SEV-SNP

VM Host Threat Scenarios

During the design phase of this Attested Launch mechanism, the following threat scenarios were considered, along with relevant mitigations when relevant for the VM Host :

- **Duplicate VM Instance**

  At any time, the VM host may freely duplicate the contents of a VM instance, including any encrypted drives, firmware, and relevant metadata (such as the instance identifier). Thus, in the event of critical bug fixes within the VM instance TCB, the VM Host may start a copy with vulnerable code.

  This threat may be mitigated by issuing updated attested firmware data along with system updates and ensuring that previous versions can not be used for attested launch processes.

- **Redirect Attested Launch destination**

  At any time, the VM host may freely redirect the communications relevant to the attested launch process between various VM instances through basic network manipulation. The VM host is also able to spoof the Verifier through similar network manipulation.

  This threat may be mitigated by ensuring all communications related to the attested launch are done through a session, and that all communications originating from the Verfier be authenticated (I.E.: through a TLS1.3 channel).

- **MitM Attested Launch protocol**

  The VM host may peek at the contents of the communications between the VM guest and Verifier at any time though basic network communication manipulation and alter them as it wishes.

  This threat may be mitigated by ensuring the protocol runs on an encrypted channel with integrity protection (such as TLS 1.3).

- **Spoof VM Identifer**

  At any time, the VM host may freely replace the identification metadata associated with the VM instance prior to launching it, as a prerequisite to being able to run multiple distinct instances from the same firmware prevents binding the identifier directly into the firmware itself. Thus forcing the VM to run through the Attested Launch protocol with the wrong identifier and potentially receiving a valid cryptographic payload associated with another identifier.

  This threat may be mitigated by ensuring the Attested Launch agent fails properly and cleans up its memory if it receives an invalid key.

- **MitM disk reads**

The VM host may at any time peek at the raw contents of the disk being read from and written to, as well as modify its contents when read.

This threat may be mitigated by ensuring that the disk contents are properly encrypted (using LUKS) and that their integrity is protected using LUKS's integrity protection mode.

VM Guest Threat Scenarios

The following threat scenarios were considered, along with relevant mitigations when relevant, for the VM Guest :

- **Arbitrary Remote Attestation Calls**

  At any time, the VM guest may run its own Remote Attestation call with arbitrary user data.

  This is deemed to be a low-priority threat and will not be mitigated at the current time. Mitigation possibilities would involve running the OS at SEV-SNP VMPL1 or above and running a custom secure service at VMPL0 that is exclusively allowed to run Remote Attestation calls. Or ensuring that the OS running within the VM properly prevents unauthorized remote attestation calls.

- **Arbitrary Attested Launch runs**

  At any time, the VM guest may run its own Attested Launch manually through the help of the Arbitrary Remote Attestation calls, allowing it to get its own disk encryption keys, along with a valid Remote Attestation report for future use with the Attested Launch protocol.

  This threat may be partially mitigated by ensuring the freshness of Remote Attestation reports used for the Attested Launch protocol through the use of a specific nonce dedicated to the Remote Attestation report generation, preventing the later use of stale reports to obtain attested launch secrets.

  Finally, to prevent a VM guest from obtaining its cryptographic secret, either the use of Intel TDX technologies with its runtime measurement, which allows for runtime state measurements on top of initial state measurements, is implemented, or the mitigation for arbitrary remote attestation calls is implemented.

- **Spoof VM Identifier**

  At any time, the VM guest is able to run the remote attestation report with an arbitrary identifier, with valid initial measurements, thus receiving the resulting cryptographic payload associated with another VM instance.

  This is the most critical capability of a VM guest regarding the security of the assets used with the Attested Launch protocol. As such, it must be carefully mitigated.

It must not be possible for the VM Guest to manipulate in any way the identifier used for the Attested launch. As such, the VM Identifier should be attested with the Attestation Report but must not be controllable by the VM Guest itself.

A single way to achieve this without having to generate unique firmware per instance was identified, which is to bind the identifier to a field that is only controllable by the VM Host, and immutable within a running VM Guest.

Within AMD SEV-SNP, there exists such a field, the HOST_DATA field, which is set by the VM Host at launch, may not be changed by the guest in any way, and is resent in the remote attestation report.

As such, to properly mitigate the possibility of a guest spoofing its own VM Identifier with the goal of obtaining cryptographic secrets associated with another Identifier, the VM Identifier shall be set by the host within the HOST_DATA field of the SEV-SNP VM launch process.

VM Guest & Host Collusion Threat Scenarios

Finally, there is the following threat scenario if both the VM Guest and VM Host collude together to attack the attested launch protocol:

- **Spoof Identifier in-depth**

  At any time, the VM host may coordinate with the VM Guest to attempt to extract the cryptographic secret associated with any Identifier. This would allow the VM host to extract the cryptographic secrets associated with any VM currently running within their control.

  This is mitigated by applying both VM identifier spoof threat mitigations; Indeed, if the VM always crashes at boot time if an incorrect key (that does not decrypt the disk) is provided through the Host-given identifier, as well as the use of an attested field settable only by the host to carry the ID, it would prevent this threat.

The sole remaining threat that was not mitigated due to the design complexity, and the choice of using AMD SEV-SNP as a remote attestation mechanism, is that the VM Host may collude with the VM Guest such that the VM Host obtains the encryption key for the VM Guest. However, this threat is somewhat unconsequential. Indeed, assuming basic security practices from the VM Guest, its collusion with the VM Host would be "voluntary" from the point of view of the Attested Launch protocol.

As such, Table 3.1 lists the threats, along with their impact, if they are within the scope of protocol design, the priority of their mitigation, and if they are mitigated within the provided Attested Launch protocol.

Entity	Threat	Impacted asset[1]	Within scope ?	Mitigated?
VM Host	Duplicate VM Instance	COS	No	No
	Redirect Attested Launch destination	COS, ALC, ALSer	Yes	Yes
	MitM Attested Launch protocol	COS, ALSec	Yes	Yes
	Spoof VM Identifier	COS, ALSer, ALSec	No	No
	MitM disk reads	COS	No	No
VM Guest	Arbitrary Remote Attestation calls	SEV-FW	No	No
	Arbitrary Attested Launch Calls	ALSec	Yes	Partial
	Spoof VM Identifier	ALSec	Yes	Yes
Collusion Host & Guest	Spoof Identifier in-depth	ALSec	Yes	Yes

[1] Shorthand forms of assets are found in subsection 3.4.3

Table 3.1: Summary of Attested Launch Threats.

3.5 Protocol Design

The following are fundamental communication security requirements, along with a detailed overview of the protocol itself.

3.5.1 Channel Security Requirements

This protocol would rely upon a 1-way authenticated, confidential channel that is integrity-preserving. Such a channel may be established through TCP over TLS 1.3 between the guest VM and Verifier; the guest will be able to authenticate the Verifier through a provisioned certificate chain in the initramfs with the help of pinned trust anchors.

The use of a 1-way authenticated channel instead of a 2-way authenticated channel is that this protocol itself does not care about who is communicating with the server. Any guest communicating with the server holding a valid identifier and its associated launch measurements from the remote attestation mechanism will be given the secret associated with the identifier. As such, the task of ensuring that a received secret is verified in depth (i.e., that it properly unlocks the underlying disk) and safe aborts in case of failure is left to the measured code to implement.

The use of 2-way authenticated channels would prove too complex to set up in a way that may not be manipulated by either the underlying host or guests themselves.

3.5.2 Communications Between the VM and Verifier

The communication between the guest VM and the Verifier is somewhat outside the scope of the protocol itself. It is assumed that the underlying host and measured par of the guest will cooperate in such a way as to facilitate the opening of a TLS 1.3 channel to the Verifier through the time synchronization required for TLS 1.3 and networking necessary for cloud service provider contexts.

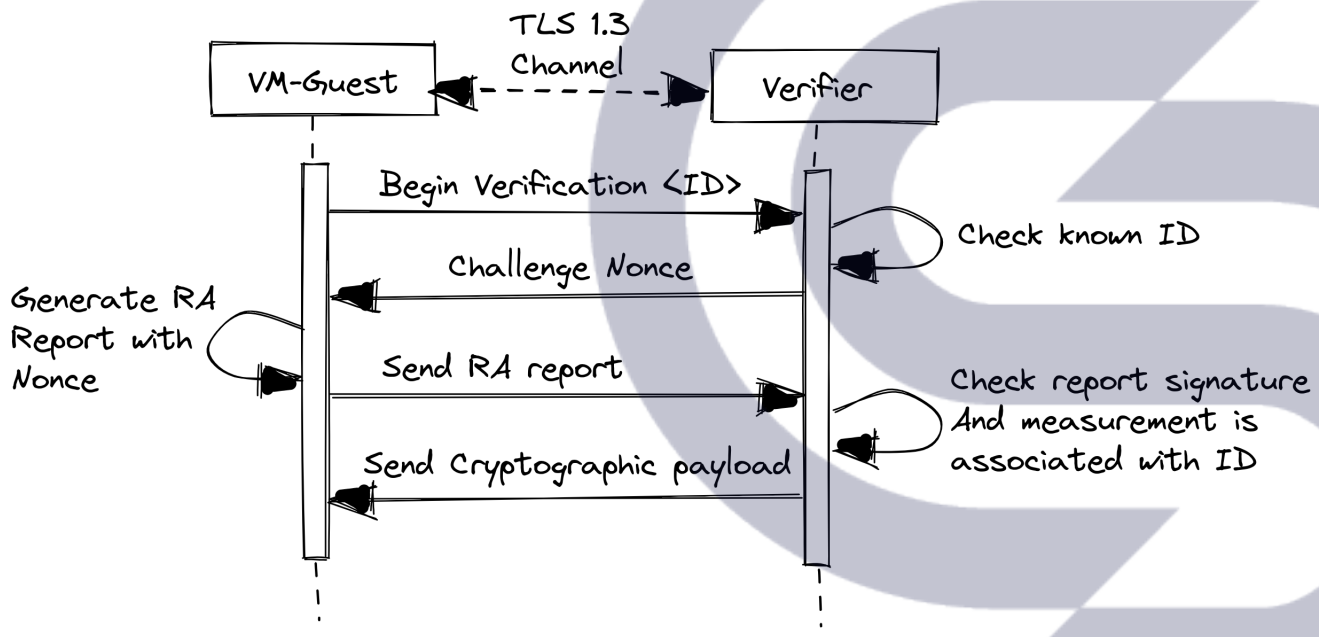

Figure 3.4: Proof of Concept Attested Launch Protocol

For this proof of concept, a simple TCP protocol was designed to be used atop a TLS 1.3 channel Figure 3.4:

1. The guest VM connects to the Verifier through its virtio-vsock with the help of the host.

2. The guest VM sends the message "Begin Verification <ID> ", ID being a 32-byte VM-specific identifier provided by KVM at VM launch in the HostData SNP field (or baked into the OVMF image if KVM does not allow for HostData provisioning).

3. The Verifier checks that the ID value is known and sends a 32-byte nonce.

4. The guest VM attaches to the attestation report the nonce and the ID value. It then transmits the attestation report to the Verifier.

5. The Verifier then checks if the attestation report is correctly signed with AMD's keys, that the attached nonce matches the one provided during the communication, and that the ID value matches the one used to initiate the session.

6. From there, the Verifier sends a payload containing any relevant cryptographic material.

3.5.3 Pre-Existing Code

This proof of concept will primarily be inspired by the AMD SEV-guest GitHub example (`https://github.com/AMDESE/sev-guest`) and based upon the AMD SEV-SNP minimal GitHub example (`https://github.com/AMDESE/AMDSEV/tree/sev-snp-devel`).

Those projects use AMD's fork of the OVMF/EDK2 project, Linux kernel, and QEMU/KVM project.

While the OVMF and Linux kernel changes have been mainlined, QEMU still needs to merge the SNP support patches, and as such, we will still need to rely upon their forks.

3.6 Linux OS Attested Launch Boot Process

Assuming a setup trying to launch a conventional Linux-based OS within an encrypted filesystem, once the Linux Kernel has been fully started, prior to being able to load the system within the encrypted disk, a minimal filesystem named "Initial ramdisk" consisting of (among other components) the tooling required to properly decrypt the filesystem shall be loaded to memory, and the Linux kernel will then give control to code within this temporary filesystem. From there, the initial ramdisk will facilitate the decryption of the contents of the disk before finally loading the actual system contained within.

For Attested Launch, it will behave much the same way, the GuestVM will begin to boot as specified in the Chain of Trust section subsection 3.2.1.

Once the guest reaches the initial ramdisk stage, It will attempt to contact the Verifier to initiate the attestation process. The boot process will fail if the Verifier is unavailable and abort the startup.

If the guest can go through the Attested Launch protocol successfully and receive the luks encryption key, the boot process will continue normally, using the received key to decrypt its root disk. If the received key is invalid, the boot process will fail and abort the startup.

The mechanism to go through the Attested Launch protocol and return the received key to cryptsetup will rely on the "keyscript" option of crypttab.

3.7 Attested Launch Protocol Security Claims

The following are security claims that may be obtained from the protocol, assuming that the technology of the underlying Remote Attestation mechanism was not compromised and that the entire VM code was controlled by the VM developer :

- A VM may only start if its initial code was not tampered with.

  Indeed, through Remote Attestation measurement, we can check that the initial code was as expected. We can also claim that

- A VM may only start if it was launched inside authorized hardware (AMD SEV-SNP compatible CPU).

  Indeed, through Remote Attestation measurement, we can check that the system is running on a genuine CPU.

- Disk encryption keys may only be distributed to the VM if it is trusted.

  Indeed, through the first two security claims, we can be sure that we are running with untampered code within genuine, trusted hardware. As such, only a trusted instance may receive the encryption keys.

- Disk encryption keys may not be distributed by the protocol to any entity other than the VM or code running within the VM.

  Indeed, since the protocol runs within a TLS 1.3 channel that authenticates the Verifier, preventing a man-in-the-middle attack, the encryption keys may not be intercepted in transit. Due to the use of an inter-protocol nonce, captured reports may not be replayed at a later date to obtain the encryption keys. Due to the first three security claims, only a trusted VM may trigger the distribution of disk encryption keys. As such, only the VM or any code running within the VM may receive the encryption keys.

  NB: If the underlying protocol was to be Intel TDX, this claim could be pushed further such that "Only the VM may receive the encryption key, and only at boot time".

- A VM may only start if a distant system authorized its start (allows for permanent decommission of remote VMs).

  Indeed, Due to the disk encryption keys never being distributed to an unauthorized third party, if at any point the given VM instance were to be discontinued, this could be enforced through the distant system no longer provisioning the disk encryption keys.

- A VM may start securely even if the underlying host is untrustworthy or even malicious. (However, there remain risks as to data availability under such conditions).

  Indeed, due to all the preview security claims and running exclusively trusted code within the VM, so long as the security of SEV-SNP remains uncompromised, the underlying host is unable to negatively impact the security of the virtual machine, aside from messing with data availability.

3.8 Comparison with Other Remote Attestation Based Protocols

While this protocol based upon Remote Attestation brings many interesting properties, there exist various other protocols and frameworks based upon remote attestation. As such, it is sensible to compare them with our Attested Launch protocol.

- **Enarx**[1]

  Enarx is an open-source platform that allows running WebAssembly applications within a secure Trusted Execution Environment based upon Intel SGX or AMD SEV-SNP. Their scope is purely limited to running applications and not entire operating systems. While they allow for easy deployment on the cloud through pre-existing SGX enclave solutions from various cloud providers, their deployment on AMD SEV-SNP within cloud providers requires the use of bare-metal server offerings with AMD SEV-SNP compatibility, with all the downsides accompanied with this requirement, as was highlighted in subsection 2.4.4.

  As such, the Attested Launch protocol, protecting a system and not simply an application, is not comparable with Enarx.

- **Gramine`https://gramineproject.io/`**

  Gramine is an open-source library/runtime that allows the running of unmodified applications within a secure, Trusted Execution Environment based upon Intel SGX. Their scope is purely limited to running applications and not entire operating systems. They allow for easy deployment on the cloud through pre-existing SGX enclave solutions from various cloud providers. They also allow for easy porting of un-modified software inside Intel SGX enclaves.

  As such, the Attested Launch protocol, protecting a system and not simply an application, is not comparable with Gramine.

- **Occlum`https://occlum.io/`**

  Occlum is an open source library/runtime that allows building with unmodified c/c++ source code, applications that can run within a secure Trusted Execution Environment based upon Intel SGX. Their scope is purely limited to running applications and not entire operating systems. They allow for easy deployment on the cloud through pre-existing SGX enclave solutions from various cloud providers. They also allow for easy porting of un-modified software inside Intel SGX enclaves.

  As such, the Attested Launch protocol, protecting a system and not simply an application, is not comparable with Gramine.

- **Open Enclave SDK`https://openenclave.io/sdk/`**

  Occlum is an open-source library/runtime that allows the building of c/c++ applications that can run within a secure Trusted Execution Environment based upon Intel SGX. Their scope is purely limited to running applications and not entire operating systems. They allow for easy deployment on the cloud through pre-existing SGX enclave solutions from various cloud providers.

  As such, the Attested Launch protocol, protecting a system and not simply an application, is not comparable with Gramine.

[1] `https://enarx.dev/`

While there are many protocols and frameworks to aid with the safe execution of confidential computing workloads through remote attestation, I was unable to find any that were able to run an entire monolithic, fully-fledged operating system within a trusted execution environment.