Attestations: The roots of trust

Authors: Christophe de Dinechin, with contributions from James Bottomley, Dr David Gilbert, Uri Lublin, Leonardo Milleri, Tyler Fanelli.

Part 1 - Confidential Computing Background
Part 2 - Attestation in Confidential Computing
Part 3 - Confidential Computing Use Cases
Part 4- From root of trust to actual trust
Part 5 - Confidential Computing Platform-Specific Details
Part 6 - Support technologies related to Confidential Computing

Part 1 - Confidential Computing Background

This article is the first in a 6-part series, where we present various usage models for Confidential Computing, a set of technologies designed to protect data in use, for example using memory encryption, and the requirements to get the expected security and trust benefits from the technology.

In the whole series, we will focus on four primary use cases: confidential *virtual machines*, confidential *workloads*, confidential *containers* and finally confidential *clusters*. In all use cases, we will see that establishing a solid chain of trust uses similar, if subtly different, *attestation* methods, which make it possible for a confidential platform to attest to some of its properties. We will discuss various implementations of this idea, as well as alternatives that were considered.

In this first article, we will provide some background about Confidential Computing and its history, and establish some terminology that we will need to cover the topic.

A Brief History of Trusted Computing and Attestation

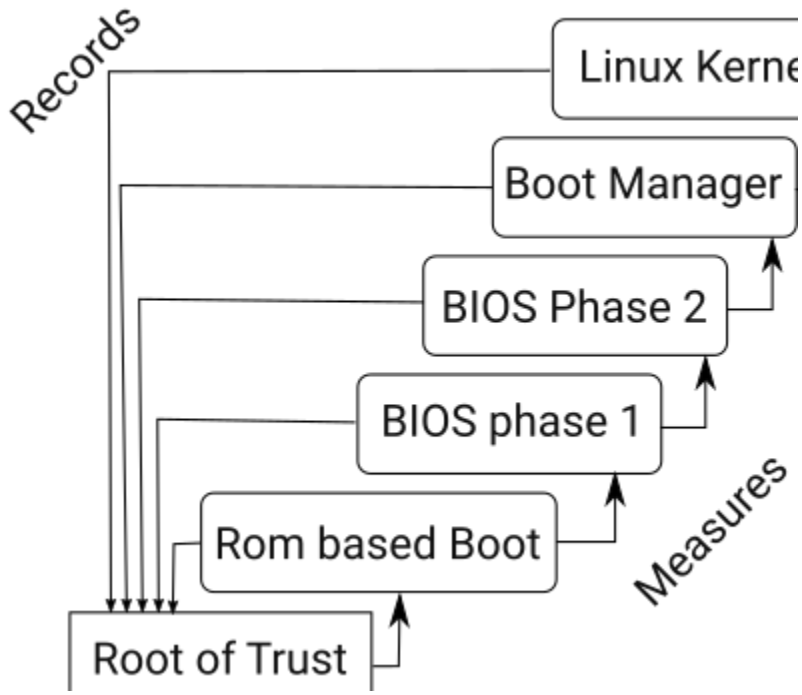
More than two decades ago, the computing industry saw a need to have trust in the components running on a remote computer (PC, laptop or even server) without necessarily having control of it. The solution was to build a chain of trust beginning with a small physical “root of trust” using a one way hash function. The way this works is as follows :

1. The root of trust first *measures* itself, meaning that it computes a cryptographic hash, either over its own code for software, or of a well known version identifier for hardware.

2. Then it measures the next stage of code to boot, for example the boot ROM for x86 platforms.
3. Once this is done, control is transferred to the boot ROM, which finds and measures the next stage of code to run and so on.

The principle is that before execution, each next stage is measured by the prior stage, meaning that if all hashes are of trusted components then the entire boot sequence must be trustworthy because the chain of hashes is grounded in the root of trust. An attacker could inject bogus code, or substitute their own code to the expected original code at any stage (except in the root of trust). However, should this happen, the measurement by the prior stage would indicate a wrong hash.

[SVG diagram showing measurement and recording. Unfortunately google docs are too primitive to insert svg's so this is a png rendering]



The mechanism we just described is a form of *attestation*: it proves to a third party some important properties about the system being used, in that case that the boot sequence only contains known components. There are many other properties that can be attested, many attestation mechanisms, and the root of trust itself can take many forms.

In the Beginning was the Physical Root of Trust

A long time ago, security researchers realized that the component that formed the root of trust would have to be a separate (and tamper proof) piece from the system being measured. This led to the development in 2000 of the Trusted Platform Modules (TPM): A fairly inexpensive and tiny chip that could be inserted into any complex system to provide a tamper resistant root of trust which could then reliably form the base of the measurement chain. Over the years, the functionality of the TPM has grown, so that it is now more like a cryptographic coprocessor, but its fundamental job of being the root of trust remains the same, and so does the construction of the chain of trust.

All Confidential Computing platforms presented here inherit this core idea of an isolated, independent physical root of trust using cryptographic methods to confirm the validity of an entire chain of trust.

Confidential Computing

Confidential Computing provides a set of technologies designed to protect *data in use* (i.e. in memory), such as the data currently being processed by the machine, or stored in memory. This complements existing technologies that protect *data at rest* (e.g. disk encryption) and *data in transit* (e.g. network encryption).

Confidential Computing is a core technology, now built in many mainstream processors or systems:

- AMD delivered the Secure Encrypted Virtualization (SEV) in 2017. The technology originally featured memory encryption only. A later iteration, SEV-ES (Encrypted State) added protection for the CPU state. Finally, the current version of the technology, SEV-SNP (Secure Nested Pages) ensures memory integrity.
- Intel proposes a similar technology called Trust Domain Extensions (TDX), which just started shipping with the Sapphire Rapids processor family. An older related technology called Software Guard Extensions (SGX), introduced in 2015, allowed memory encryption for operating system processes. However, it is now deprecated except on Xeon.
- IBM Z mainframes feature Secure Execution (SE), which takes a slightly different, more firmware-centric approach, owing to the architecture being virtualization-centric for so long.

- Power has a Protected Execution Facility (PEF), introducing what they call an *ultravisor*, that offers higher privilege than the hypervisor (for virtual machines) and the supervisor (for the operating system), and grants access to secure memory.
- ARM is developing the Confidential Compute Architecture (CCA), which introduces the Realm Management Extensions (RME) in the hardware, and allows firmware to separate the resources between realms that cannot access one another.

While all these technologies share the same goal, they differ widely in architecture, design and implementation details. Even the two major x86 vendors take very different approaches to the same problem. Among the primary differences are the roles and weights of firmware, hardware or adjunct security processors in the security picture. This horridly complex landscape makes it difficult for the software vendors to present a uniform user experience across the board.

Host, guest, tenant and owner

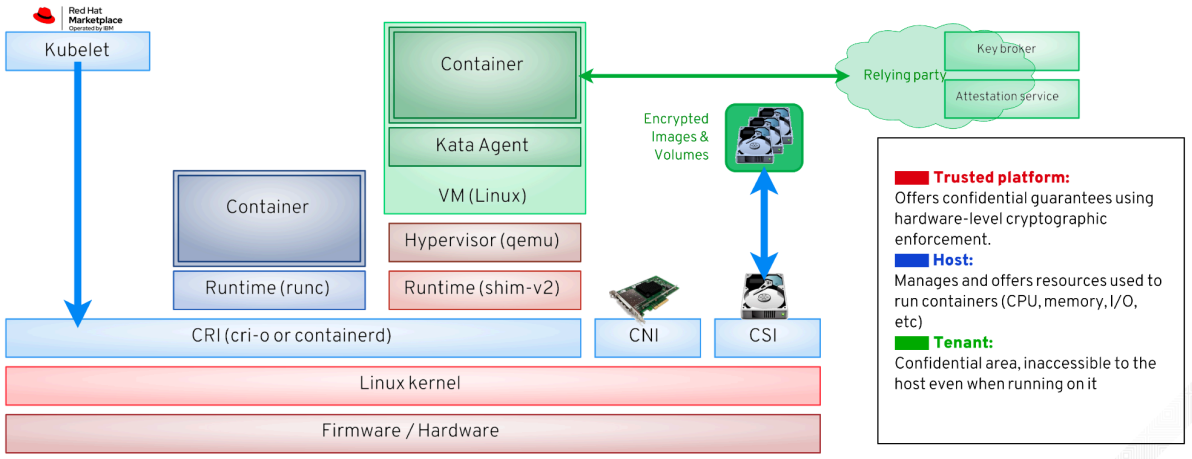
In Confidential Computing, the host platform is *no longer trusted*. It belongs to a different trust domain than the guest operating system. This forces us to introduce a new terminology. When talking about virtualization, we usually make a distinction between *host* and *guest*, and this implies – correctly – that the guest has no special confidentiality rights or guarantees for anything inside the host.

By contrast, in a way reminiscent of lodging, we will talk about a *tenant* for a confidential virtual machine. The tenant does have additional rights to confidentiality, similar to the restrictions preventing a building manager from accessing a tenant's private apartment.

We can also talk about the *owner* of the virtual machine, notably when we refer to components outside of the virtual machine, but that belong to the same trust domain, such as a server providing secrets to the virtual machine. In other words, the tenant's trust domain may extend beyond the virtual machine itself, and in that case, we prefer to talk about ownership. Notably, we will see that the integrity of Confidential Computing, and the security benefits it can provide, often relies on controlled access to external resources, collectively known as the *relying party*. This includes in particular services providing support for various forms of *attestation* to ensure that the execution environment is indeed trusted, as well as other services delivering *secrets* or *keys*.

The diagram below, corresponding to a confidential containers scenario, illustrate these three trust domains of interest using different color:

- In red, the *trusted platform* offers services that must be relied on not just to execute the software components, but also to provide cryptographic-level guarantees.
- In blue, the *host* manages the physical resources, but it is not trusted with any data that resides on it. Data sent to disk or network devices must be encrypted, for example.
- In green, the *owner*, which on the diagram includes a role as a *tenant* on the host, within the virtual machine, as well as another role with external resources collectively forming the *relying party*.



Example red, green that [tbuskey] can tell differences. Red green too

What does Confidential Computing guarantee

In the marketing literature, you will often see vague attributes such as “security” touted as expected benefits of Confidential Computing. However, in reality, the additional security is limited to one particular aspect, namely confidentiality of data in use. And even that limited benefit requires some care.

The only thing that a Confidential Computing guarantees is… *confidentiality*, most notably the confidentiality of data in use, including data stored in random-access memory, in the processors’ internal registers, and in the hypervisor’s data structures used to manage the virtual machine. In other words, what the technology ensures is that data being processed by the virtual machine will not be accessible outside of the trust domain. Most notably, the host, the hypervisor, other processes on the same host, other virtual machines and physical devices with DMA capabilities should all remain unable to access cleartext data. Note that some platforms like current-generation IBM-Z may offer some level of Confidential Computing without necessarily implementing it through physical memory encryption.

Confidentiality can be understood as protecting the data from being read from outside of the trusted domain. However, it was quickly understood that this also requires *integrity protection*, to make sure that a malicious actor cannot tamper with the trusted domain. Such tampering would make it all too easy not just to corrupt trusted data, but possibly even to take over the execution flow sufficiently to cause a data leak.

In practical terms, this means that a malicious system administrator on a public cloud can no longer dump the memory of a virtual machine to try and steal passwords as they are being processed. What they would get from such a dump would, at best, be encrypted versions of the password. This is illustrated in this demo of confidential workloads, where a password in memory is shown to be accessible to a host administrator if not using Confidential Computing (at time marker 01:23), and no longer when Confidential Computing encrypts the memory (at time marker 03:50).

Note that on many platforms, the memory encryption key resides in hardware (which may be a dedicated security processor running its own firmware like on AMD-SEV, or a separate hardware-protected area of memory like for ARM-CCA), and barring hardware-level exploits finding a way to extract it, there is no practical way to decrypt encrypted memory. Most importantly, the keys cannot be accessed through human errors, for example through social engineering, and that matters since human error is one of the most common methods to compromise a system.

The *only* failure mode of concern for Confidential Computing is a *data leak*. Denial of service (DoS) is specifically out of scope, and for a good reason: the host manages physical resources, and can legitimately deny their access at any time, and for any purpose, ranging from the mundane (throttling for cost reasons) to the catastrophic (device failure, power outage or datacenter flooding). Similarly, from a confidentiality point of view, a crash is an acceptable result, as long as that crash cannot be exploited to leak confidential data.

Note: This uproots the usual security model for the guest operating system. In a traditional model, the execution environment is globally trusted: it makes no sense to think about the security damage that could result if program instructions start misbehaving, if register data is altered or if memory content changes at random. However, in a Confidential Computing scenario, some of these questions become relevant.

Intel started documenting various new potential threats, and it's fair to say that there was some sharp criticism from the kernel community, even after significant rework. One of the most obvious attack vectors is the hypervisor. A malicious hypervisor can relatively easily inject random data or otherwise disturb the execution of the guest, facilitating timing attacks or lying about the state and capabilities of the supporting platform to suppress necessary mitigations. Emulation of I/O devices, notably access to device registers in the PCI space, may require additional scrutiny if bad data can lead to controlled guest crashes that would expose confidential data. This is still an active area of research.

Various kinds of proof

Like all security technologies, Confidential Computing relies on a *chain of trust* which maintains the security of the whole system. That chain of trust starts with a *root of trust*, i.e. an authoritative source that can vouch for the encryption keys being used. It is built on various *cryptographic proofs* that provide some strong guarantees.

The most common forms of proof include:

- *Certificates*, which prove someone's identity. If you use Secure Boot to start your computer, Microsoft-issued certificates confirm Microsoft's *identity* as the publisher of the software being booted.
- *Encryption,* guaranteeing that the data is only readable by its intended recipient. When you connect to any e-commerce site today, HTTPS *encrypts* the data between your computer and the web server.
- *Integrity* to ensure that data has not been tampered with. When you use the Git source code management, the "hash" that identifies each commit also guarantees its *integrity*, something that developers with a disk going bad sometimes discover the hard way. Integrity is often proven by means of cryptographic *measurements*, such as a cryptographic hash computed based on the contents of an area of memory.

Conclusion

As any emerging technology, Confidential Computing introduced a large number of concepts, and in doing so, is creating its own acronym-filled jargon. In the next section, we will focus on one particular concept, namely attestation.

Part 2 - Attestation in Confidential Computing

This article is the second in a 6-part series, where we present various usage models for Confidential Computing, a set of technologies designed to protect data in use, for example using memory encryption, and the requirements to get the expected security and trust benefits from the technology.

In the whole series, we will focus on four primary use cases: confidential *virtual machines*, confidential *workloads*, confidential *containers* and finally confidential *clusters*. In all use cases, we will see that establishing a solid chain of trust uses similar, if subtly different, *attestation*

methods, which make it possible for a confidential platform to attest to some of its properties. We will discuss various implementations of this idea, as well as alternatives that were considered.

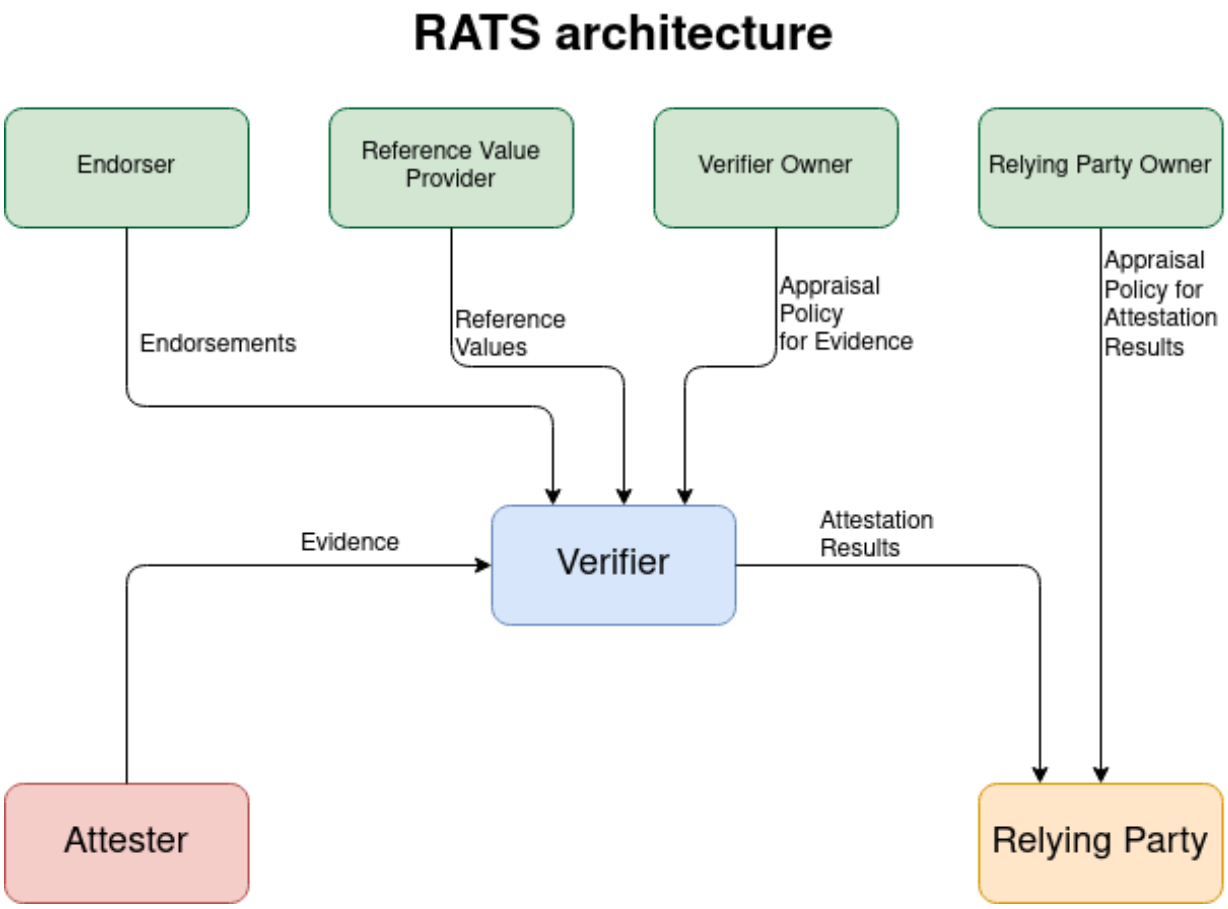
In this second article, we will focus on attestation, as a method to prove specific properties of the system and components being used.

The need for attestation

In a Confidential Computing environment, another form of proof called *attestation* becomes increasingly important. Generally speaking, attestation is designed to prove a property of a system to a third party.

In the case of Confidential Computing, it generally means a proof that the execution environment can be trusted before starting to execute code or before delivering any secret information.

At the highest level, one very general definition of attestation is described by the Internet Engineering Task Force (IETF) Remote Attestation Procedures (RATS) architecture using the diagram below:



We will use the terminology from this diagram. The benefit of this model for the attestation process is that it clearly delineates the responsibilities of each component.

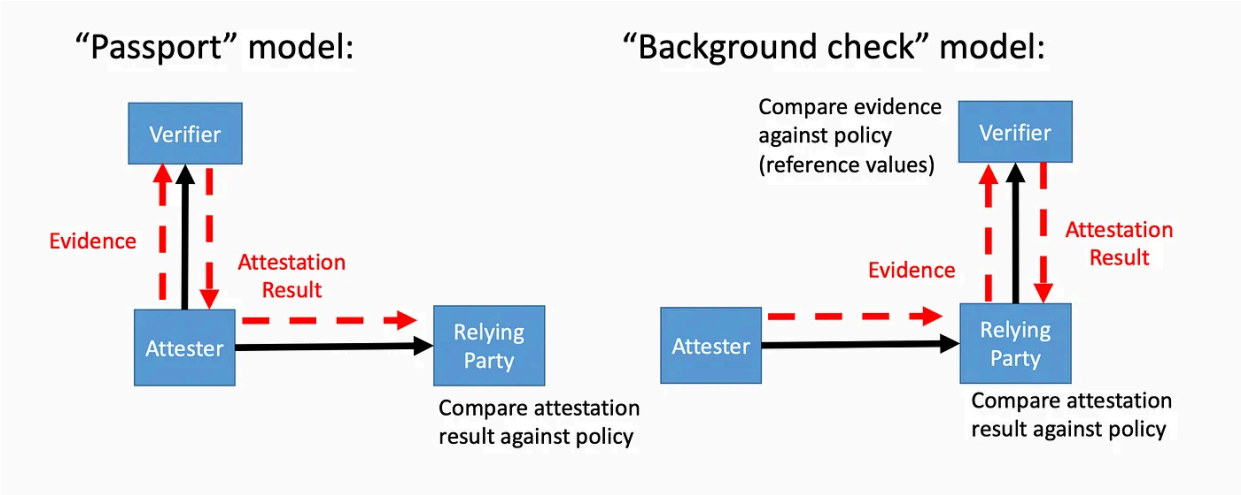
Remote attestation

Remote attestation decouples the generation of evidence from its verification, allowing for example an *attestation server* (AS) to dynamically respond to situations such as the discovery of new vulnerabilities and start rejecting a previously accepted configuration. A physical chip like a TPM can only do very limited policy enforcement. Using a remote server allows for a much richer policy verification, as well as near real-time updates for new vulnerabilities.

You may have heard about remote attestation outside of Confidential Computing through projects such as Keylime, which provides remote boot attestation for TPM-based systems. An important difference is that in its current typical usage model, Keylime focuses on proving compliance after the fact (non-blocking attestation), whereas in the case of Confidential Computing, attestation will typically have to pass before anything confidential is entrusted to the platform.

Two distinct attestation models can be used, known as the *passport model* and the *background check model*:

- The passport model is where the attester pre-validates an identification token with the verifier, that it can then present to the relying party. The real-life equivalent is presenting your passport.
- The background check model is where the relying party will ask for a verification when the attester presents its evidence. A real-life equivalent would be the verification of biometric measurements.



The cryptographic verification provided by Confidential Computing technologies generally lends itself more to a background check model than to a passport model. This is generally more useful, notably because it makes it possible to revoke access at any time. Unless stated otherwise, we will generally mean a background check when we talk about remote attestation in this post.

Components for remote attestation

Typically, remote attestation in Confidential Computing will involve a variety of recurring components:

- An *attestation server* (AS) that will submit the virtual machine to a cryptographic challenge to validate the measurement presented as evidence. It will act as a verifier on behalf of the relying party.
- An *attestation client* (AC) that lets the attester send evidence to an attestation server.

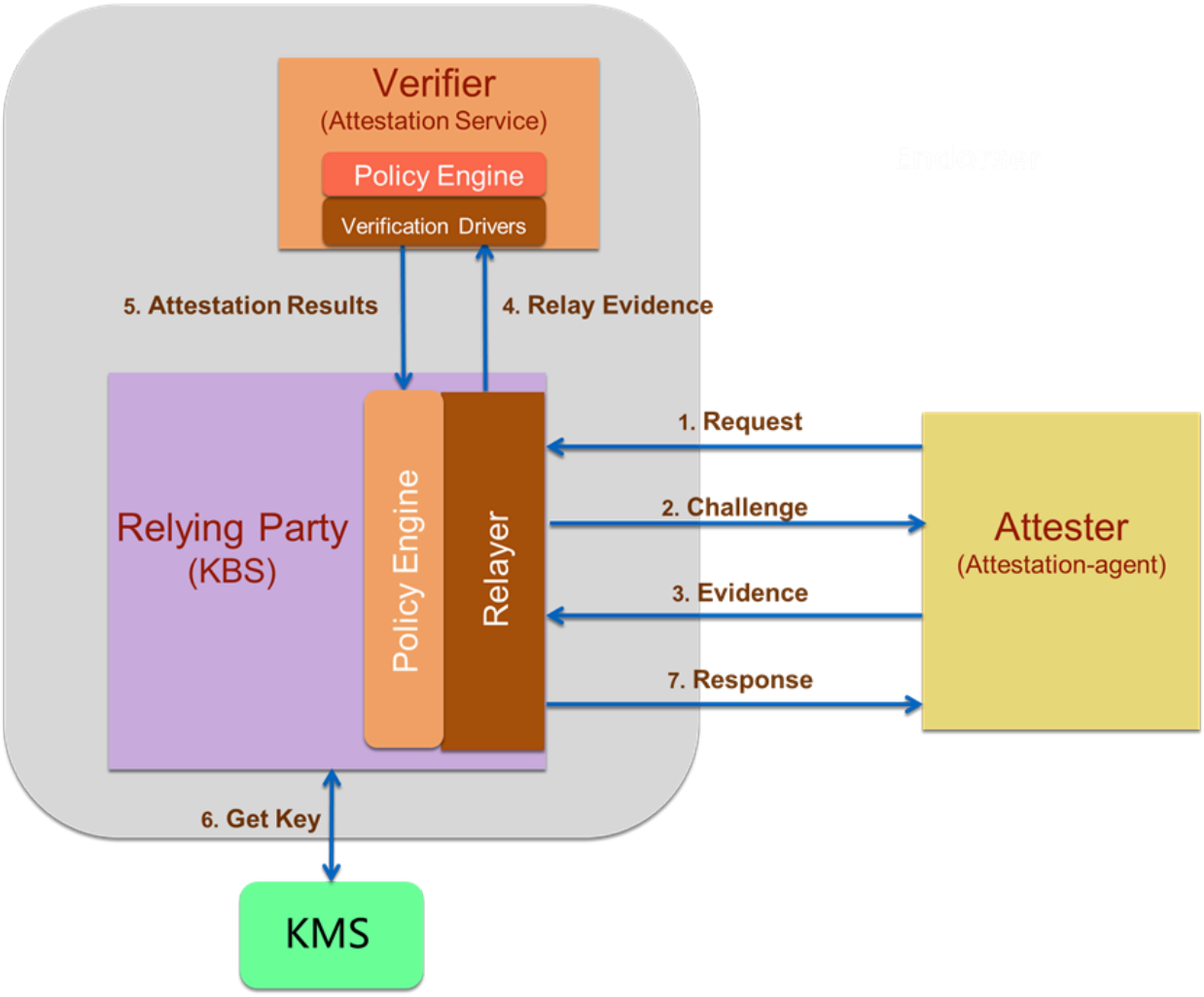
- A *key broker service* (KBS) will store secrets such as disk encryption keys, and release them only when verification is successful. This KBS can be part of a larger *key management system* (KMS).
- A *key broker client* (KBC) will receive the keys from the KBS on behalf of the attester.

Note that the key brokering part is, conceptually, distinct from attestation itself, as evidenced by products that focus primarily on attestation like Keylime, without necessarily providing key brokering services. However, in the case of Confidential Computing, the attestation service is typically not intended for humans to check compliance of their inventory, but becomes an integral part of the confidentiality guarantee, typically through the delivery of secrets.

Attestation flow

The diagram below shows an example of attestation flow for Confidential Computing:

- A *request* is sent to the attestation server by the attester.
- The server responds with a *challenge*, which typically includes a nonce to avoid replay attacks.
- The attester submits its *evidence*, which combines elements of the challenge with attester-provided data, in such a way that the result cannot be reproduced by a third party, nor be of use by anyone but the originator of the challenge, to block man-in-the-middle attacks.
- The attestation server includes a verifier that applies various policies. This could include constraints about what kind of evidence is accepted, expiration dates or revocation lists.
- If attestation is successful, the key broker service is instructed to release the keys for that specific attester.
- The keys are packaged in a response that the KBC in the attester can consume.



Recipients of the proof

There is another useful way to categorize various forms of attestation, based on who is the intended recipient of the proof:

- *User-facing*: an individual using the system wants proof that the system is trustworthy.
- *Workload-facing*: a workload running on the system wants to ensure that it runs on a trusted execution environment (TEE).
- *Peer-facing*: a workload wants to ensure that a peer workload is itself trustworthy and running on a trustworthy platform.
- *System-facing*: system software (including firmware, bootloader or operating system) wants to guarantee its own integrity and the integrity of its execution environment.
- *Cluster-facing*: nodes in a cluster want to ensure that the integrity of the whole cluster is not compromised, notably to preclude non-confidential nodes from joining.

Not all of these categories of attestation are useful in all use cases, and this list is by no means exhaustive. One could attest software, hardware, configurations, and more.

Securely Recording the Measurements: The need for hashing

The physical Root of Trust doesn't usually contain enough storage for all the measurements. This is certainly true for today's TPMs. So we have to resort to a hashing trick: the device usually only stores cryptographic hashes that can be used to verify the actual record in ordinary memory.

Each measurement is an ordered set of log entries consisting of a hash (the machine measurement) and a human readable description. In the case of TPMs, the log hashes are recorded in a Platform Configuration Register (PCR) whose value begins at zero and is "extended" by each measurement. Extended means that the new value is the hash of the old value and the new measurement.

Since hash functions are not reversible, it is impossible[1] to construct a different sequence of measurement extensions that will result in the same PCR value at the top. This property means that the single PCR hash value can be used to verify the entire sequence of measurements is correct and has not been tampered with. Given a correct log, anyone can verify the PCR value by replaying all the recorded measurements through the hash extension function and verifying they come up with the same value. Note that this means the log must be replayed in exactly the same order.

In order to attest to the entire log, the root of trust usually signs the single PCR hash value. Since the only thing that can be done to a PCR is extend it, there's no real need for security around who can do the extension: the object for most attackers is to penetrate the system undetected and a bogus extension would lead to a log verification failure and immediate detection. This can actually be used as a feature, where a later stage can extend an earlier measurement, deliberately altering it, for example to grant access to different secrets as execution progresses.

When verifying the state of the system, it is tempting to see the single PCR value as the correct indicator of state (which it is). However, all system components (and configurations) change quite often over time which can cause a combinatorial explosion in the number of possible PCR

---

[1] While finding a hash collision is theoretically possible, it would take longer than the lifetime of the universe to achieve. Once a hash function becomes vulnerable to a collision in less time than this (e.g. md5, sha1) it is discarded as insecure and a new hash function is invented (e.g. sha2, sha3).

values representing acceptable system state, so sensible verifier solutions usually insist on consuming the log so they can see the state of each individual component rather than relying on the single PCR value to represent it. In other words, the PCR only *attests* the validity of a configuration, but may not be the best way to access it.

Identity and privacy concerns

In order to sign the measurements, each TPM has to be provisioned with a unique private key, which must be trusted by the party relying on the signature. Unfortunately, this unique key also serves to uniquely identify the system being measured and has led to accusations of the TPM and Trusted Computing generally being more about social control than security.

To allay these fears, the Trusted Computing Group (guardians of the TPM specification) went to considerable lengths to build privacy safeguards into the TPM attestation mechanisms. Nowadays, of course, most attestations are done by people who know where the system being measured is and what it's supposed to be running. Under these circumstances, all of the privacy protections now serve only to complicate the attestation mechanism. If you've ever wondered why it's so complicated to get the TPM to give you a quote, this is the reason. However, while it's a practical concern if you actually write libraries talking to a TPM, we will not concern ourselves with identity and privacy in this blog article.

Conclusion

In this second article, we covered the basic ideas about attestation, and how it can be useful for Confidential Computing in general. In the next article, we will enumerate the most important use cases for Confidential Computing, and see how they differ in their use of the same underlying technology, as well as how this impacts the implementation of attestation.

Part 3 - Confidential Computing Use Cases

This article is the third in a 6-part series, where we present various usage models for Confidential Computing, a set of technologies designed to protect data in use, for example using memory encryption, and the requirements to get the expected security and trust benefits from the technology.

In the whole series, we will focus on four primary use cases: confidential *virtual machines*, confidential *workloads*, confidential *containers* and finally confidential *clusters*. In all use cases,

we will see that establishing a solid chain of trust uses similar, if subtly different, *attestation* methods, which make it possible for a confidential platform to attest to some of its properties. We will discuss various implementations of this idea, as well as alternatives that were considered.

In this third article, we consider the four most important use cases for Confidential Computing: confidential *virtual machines*, confidential *workloads*, confidential *containers* and confidential *clusters*. This will allow us to better understand the trade-offs between the various approaches, and how this impacts the implementation of attestation.

Usage models of Confidential Computing

In the existing implementations (with the notable exception of Intel SGX), Confidential Computing is fundamentally tied to virtualization. A *trust domain* corresponds to a virtual machine, each domain having its own encryption keys and being isolated from all other domains, including the host the virtual machine is running on.

There are several usage models to consume these basic building bricks:

- A *confidential virtual machine* (CVM) is a virtual machine running with the additional protections provided by Confidential Computing technologies, and obeying security requirements to ensure that these protections are useful. Running SEV-SNP instances on Azure is an example of this use case.
- A *confidential workload* (CW) is a very lightweight virtual machine using virtualization only to provide some level of isolation, but otherwise using host resources mostly in the same way as a process or container would. This use case is exemplified by libkrun, and can now be used with podman using the “krun” runtime.
- *Confidential containers* (CCn) will use lightweight virtual machines as Kubernetes pods to run containers. The primary representative in that category is the Confidential Containers project, derived from Kata Containers, which recently joined the Cloud Native Computing Foundation (CNCF).
- A *confidential cluster* (CCl) is a cluster of confidential virtual machines, which are considered as being part of a single trust domain. The Constellation project is one of the early offerings in that space, and provides a consistent analysis of the security implications in that problem space.

There may be more than the ones we list here, but at the time of writing, these four use cases are the current development focus of the free software community.

Confidential Virtual Machines

The most direct application of the Confidential Computing technology is confidential virtual machines. This use case takes advantage of the technology without wrapping it into additional logic or semantics.

However, as we pointed out, in order to get the full benefits of the additional confidentiality, we must secure the rest of the system, so that the data that we protect through memory encryption cannot be recovered from a non-encrypted disk image, for example. Consequently, a CVM must use encrypted disks and networking. It also needs to use a secure boot path, in order to guarantee that the system software running in the virtual machine is the correct one, and that it was not tampered with.

This model is useful to run standard applications (as opposed to containerized ones), independent operating systems, or when the owner of the virtual machine can define ahead of time a complete execution environment. In such scenarios, the owner needs to build and encrypt individual *disk images* for the virtual machines that will generally contain anything that is necessary to run the application. Notably, various application secrets may reside on the disk image itself.

As a result, in that configuration, the primary security concern is to prevent the confidential software from running in a possibly compromised environment. We want to preclude the host from tampering with boot options, or from starting the virtual machine with a random, and possibly malicious, firmware or kernel, which could possibly be used to leak data.

In the cloud, one way to achieve this objective is to tie the encryption keys for the disk to a specific system software configuration. This can be done by sealing the required encryption keys in a virtual Trusted Platform Module (vTPM), so that they can only be used with a virtual machine associated with that specific TPM, and only when the TPM-measured boot configuration matches the desired policy. Note that for this to be robust, the vTPM itself needs to be protected by the underlying Confidential Computing technology, the attestation of the vTPM being linked to the attestation of the Confidential Computing system.

In this post, we will illustrate this approach by explaining how Red Hat Enterprise Linux 9.2 can take advantage of Azure SEV-SNP instances, using an Azure-provided virtual TPM. In that scenario, Microsoft provides system-facing attestation, validating the initial system measurements through the Microsoft Azure Attestation service. This unlocks keys used to decrypt the vTPM state.

On premise, or if you control the hardware directly, you may want to deploy your own attestation services. As we will see below, the way to do that largely depends on the target platform.

Confidential Workloads

Confidential workloads is an innovative way to run containers using a very lightweight virtualization technique, where the guest kernel is packaged as a shared library in the host. The open source project that introduced this lightweight virtualization model is called libkrun, and the tool to run containers from standard container images is called krunvm.

This model is useful to quickly run and deploy small container-based applications, typically with a single container. The driving factor for confidential workloads is quick startup time and reduced resource usage for higher density. The current implementation also features a good integration with podman (which the Kata Containers dropped, and therefore Confidential Containers lack). The ecosystem includes tooling to create workload images from OCI container images, and a simple attestation server. This is well described in a nice blog post containing an illustrative demonstration.

The primary concern in this scenario is to ensure that the workload is running in a TEE with a valid system software stack, and that only the workload is running there. This concern is expressed as follows by the above blog:

> *When intending to run a confidential workload on another system (e.g. on a machine from a cloud provider), it is reasonable for a client to inquire "How do I know this workload is actually running on a TEE, and how do I know that my workload (and ONLY my workload) are what is running inside this TEE?". For sensitive workloads, a client would like to ensure that there is no nefarious code/data being run, as this code/data can be violating the integrity of the TEE.*

In this scenario, the entire workload, including both the kernel and user space, is therefore registered for attestation, as well as a valid configuration to run it. A successful attestation will deliver the disk encryption key, unlocking the disk that the workload needs. In that respect, confidential workloads, while working a bit like containers, are actually closer to confidential virtual machines. A consequence of this is that you need to build each individual workload and register it with the attestation server.

Confidential Containers

Confidential Containers is a sandbox project in the Cloud Native Computing Foundation (CNCF). It derives from Kata Containers, a project using virtualization to run containers, using a virtual machine for each pod (a pod is a Kubernetes unit that can contain one or more related containers). The two projects share most of the developer community, and the "confidential" aspect is merged back into Kata Containers on a regular basis. So Confidential Containers is a kind of "advanced development branch" of Kata Containers, more than a fork.

As a result, the Confidential Containers project inherits a solid foundation for a project that is still so early in its development, including a vibrant community, a number of industrial potential users, know-how and resources on best practices, continuous integration (CI), and the collaboration of heavyweights such as Alibaba, Ant Group, IBM, Intel, Microsoft, Red Hat and many others. However, the project is still in its infancy, with version 0.5 to be released in April 2023 and a release schedule of about 6 weeks.

One of the primary concerns for this project is to make Confidential Computing easy to consume at scale. This implies being able to ignore, to the largest possible extent, details of the hosts being used to provide the resources, including their CPU architecture, and integrating well with existing orchestration tools such as OpenShift or Kubernetes. The project is currently developing and testing with AMD SEV, SEV-ES and SEV-SNP, Intel SGX and TDX, and IBM s390x SE. The installation of all the required artifacts for this project is also made relatively simple thanks to a kubernetes operator that deploys the necessary software on a cluster, and makes them easy to consume using the widely used Kubernetes concept of runtime class.

Note that Kata Containers lost the compatibility with podman in version 2.0, which makes it less convenient to use from the command-line than confidential workloads.

The attestation procedure is similarly flexible, with a generic architecture that can deal with local and remote attestation, including pre-attestation as required for early versions of AMD SEV, or firmware-based attestation as is required for the IBM s390x SE platform. There is even support for disk-based "attestation" to make it possible to develop and test on platforms that do not support Confidential Computing.

At least in the current implementation, the attestation process only covers the execution environment, but not the workload being downloaded. This could change over time, as the project is discussing the structure of the reference provider service, appraisal policies or container metadata validation. But the current approach means that there is a bit more flexibility in the deployment of workloads, since you do not necessarily have to register each individual workload, but only each individual class of execution environment.

A distantly related project is Inclavare Containers, which is based on the Intel SGX enclave model.

Confidential Clusters

Confidential clusters are the last use case we are going to discuss here. Edgeless Constellation is an open-source implementation of this approach.

In that approach, entire Kubernetes clusters are built out of confidential virtual machines. This makes it somewhat easier to deploy an infrastructure where everything, from individual containers to the cluster's control plane, runs inside a set of confidential virtual machines: when all the nodes in the cluster are confidential virtual machines, all the containers running within the cluster (on a confidential VM) are confidential as well. This makes it relatively easy to deploy even the most complicated combinations of containers, including operators or deployments.

In addition to the per-CVM (single node) attestations that make sense for the earlier scenarios, new concerns emerge, like making sure that a non-confidential node does not join a confidential cluster, which would facilitate leaks of confidential data. For that reason, Constellation provides additional attestation services, a JoinService to verify that a new node can safely join a cluster, and a user-facing VerificationService to check if a cluster is legit.

Conclusion

The four major usage models for Confidential Computing use the same underlying technology in different ways. This leads to important differences in how trust is established, what kind of proof is expected, and who expects these proofs. In the next article, we will discuss the general principles of moving from a root of trust to actual trust in a system.

Part 4- From root of trust to actual trust

This article is the fourth in a 6-part series, where we present various usage models for Confidential Computing, a set of technologies designed to protect data in use, for example using memory encryption, and the requirements to get the expected security and trust benefits from the technology.

In the whole series, we will focus on four primary use cases: confidential *virtual machines*, confidential *workloads*, confidential *containers* and finally confidential *clusters*. In all use cases,

we will see that establishing a solid chain of trust uses similar, if subtly different, *attestation* methods, which make it possible for a confidential platform to attest to some of its properties. We will discuss various implementations of this idea, as well as alternatives that were considered.

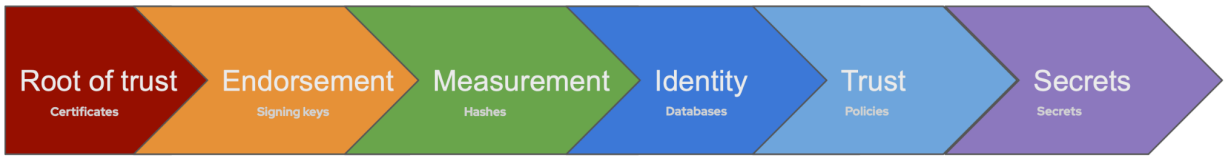
In this article, we will focus on establishing a chain of trust, and introduce a very simple REMITS pipeline that we can use to compare and contrast the various forms of attestation using a single referential.

Simplified REMITS model

Some principles and techniques used to implement a chain of trust in Confidential Computing are general enough that they apply to all Confidential Computing platforms. Most of them can be traced back to the initial efforts in Trusted Computing, which we mentioned in our *Brief History of Trusted Computing* in the first article in the series.

Attestation is useful to build a chain of trust incrementally, in a continuous way from a root of trust all the way to actual trust about a system, generally expressed in the form of secrets being delivered. We will now see the fundamental principles of this process, and propose a super-simplified model of this chain of trust construction that will allow us to compare and contrast different implementations more easily.

To be able to compare wildly differing implementation, we propose a very simple chain of trust REMITS pipeline, which identifies six components:



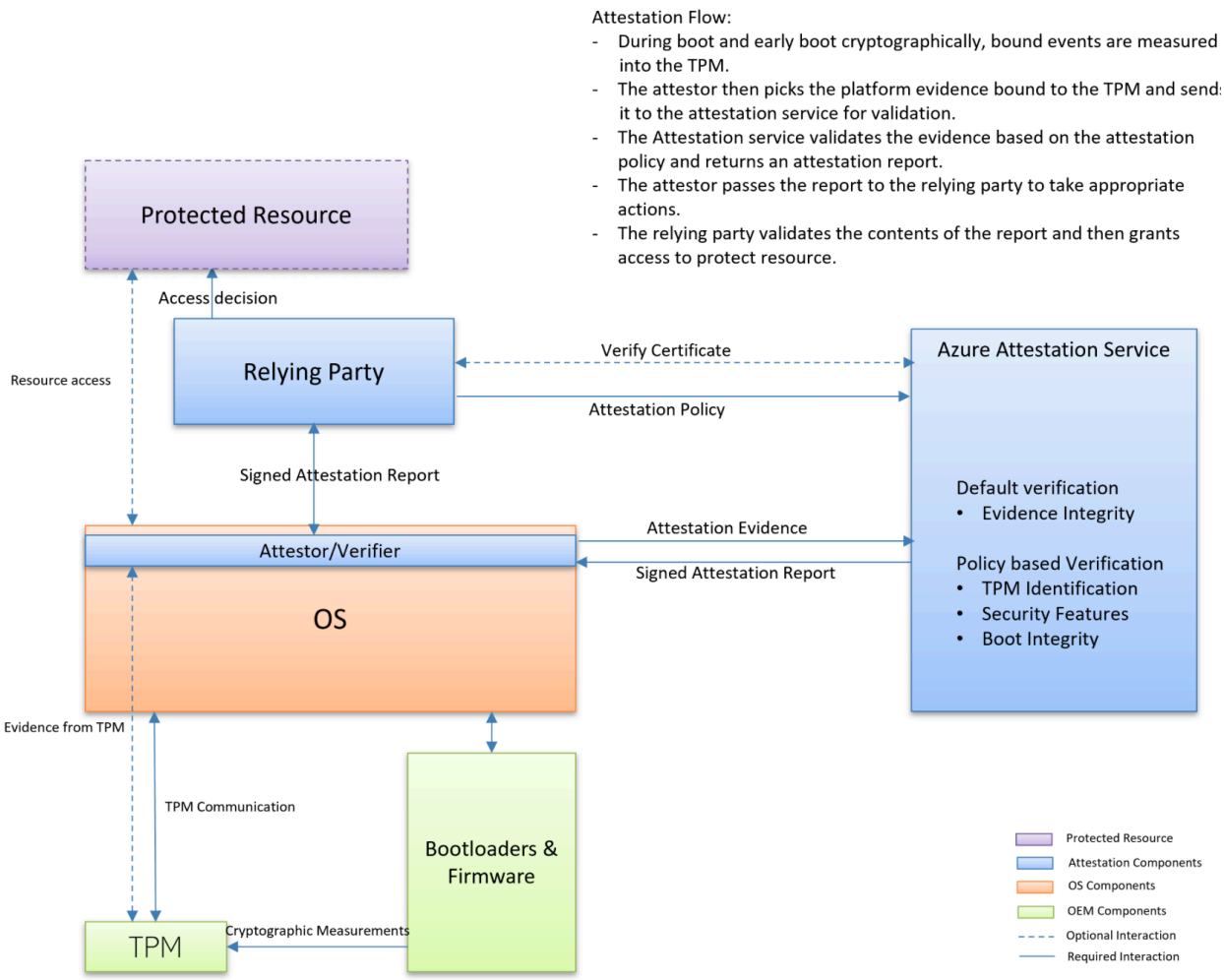
- **R**oot of Trust: This is a shared, immutable piece of information such as a private key from a chip manufacturer where the public key is known, which is used to validate the whole chain of trust. It should be noted that this is also a single point of failure for the whole scheme: if the private key is leaked, the trust in the entire chain is lost.
- **E**ndorsement: An endorsement makes it possible to create per-device information that can reliably be traced back to the root of trust. For example, a per-chip endorsement key that is signed by the manufacturer's key can confirm that the manufacturer gives its seal of approval to that chip.

- **M**easurement: A measurement generates data that reliably identifies a particular configuration of the attester. In the case of Confidential Computing, this will typically include a cryptographic hash of the software stack including configuration parameters such as command line or security options. The measurement must also be reliably traced back to an endorsed device, and from there to the root of trust. This forms the evidence provided by the attester in the RATS model.
- **I**dentity: The verification process in the RATS model is intended to prove the identity of the attester based on the evidence being submitted, and comparing it with known reference values. In the case of remote attestation, this identity is typically represented as some internal token in the attestation service.
- **T**rust: The actual trust is not built from the identity alone, but as the RATS model shows, by also applying policies to appraise the evidence. For example, in the case of Confidential Containers, we may want to have more lax policies during development that will grant additional rights to developers. In other words, developers will receive a higher level of trust than a regular deployment because of different policies.
- **S**ecrets: The manifestation of trust is sharing various secrets that will be handed to the attester and allow it to access confidential data. For example, the secrets typically include disk encryption keys without which the workloads will not be able to execute. Proceeding that way ensures that without trust, there will be no harm done.

This pipeline model is extremely simplified, but it has the benefit of being widely applicable, which allows us to compare various implementations.

Attestation on physical hosts with a Trusted Platform Module (TPM)

The attestation of a physical host typically uses a trusted platform module. Microsoft has extensive documentation on this process, shown below:



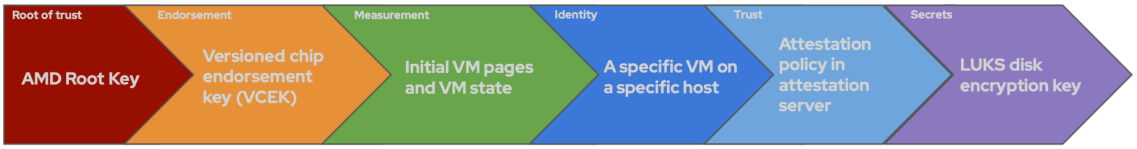
In this process, the REMITS pipeline looks like this:

Virtual TPMs to run Confidential VMs on cloud services

Generally speaking, cloud providers seem to be taking a direction where they expose the trust in the system using a virtual Trusted Platform Module (vTPM). There is a good reason for that, since this is a known interface that was, among other things, made mandatory for secure boot on Windows 11. We can illustrate this with the Microsoft Azure support for SEV-SNP instances.

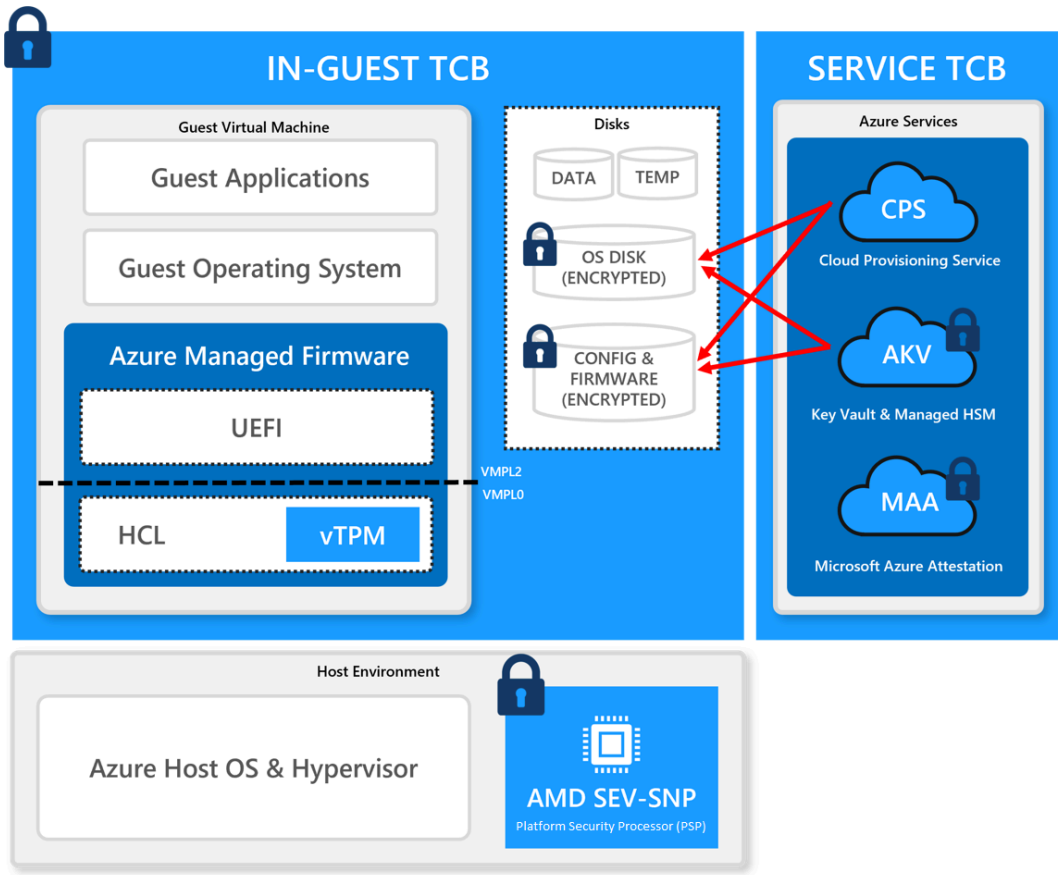
In this scenario, the REMITS pipeline actually runs twice, the first time in the new CC environment, in a platform specific way, the second time exposing more standard interfaces to the workload:

- In the first run, we use hardware-provided evidence against the cloud provider's own attestation architecture such as Microsoft Azure Attestation. If successful, this first run unlocks the secrets necessary to build a virtual TPM e.g. from persistent storage. How this is done precisely is part of Microsoft's secret sauce, and the corresponding software is not open source as far as we know. The root of trust in that first run is in hardware, for example an AMD root key (ARK).



- A second run will then start with the vTPM as a root of trust, and the secrets become accessible through the standard mechanisms specified for all TPMs, which were described in the previous section. So except for the root of trust being a virtual TPM, it is otherwise equivalent.

The Microsoft Azure infrastructure provides guidance on how to integrate with their attestation service, illustrated below, and they provide multiple choices of implementation based on who you decide to trust initially (i.e. what root of trust you are willing to use as the starting point for your chains of trust).



Attestation in Confidential Workloads

Confidential workloads implement remote attestation based on registration of the workload launch measurement as well as a secret passphrase to unlock the LUKS-encrypted disk:

```
{
    "workload_id": "my_workload",
    "launch_measurement": "hex-encoded-measurement",
    "tee_config":
"{\"flags\":{\"bits\":63},\"minfw\":{\"major\":0,\"minor\":0}",
    "passphrase": "my_secret_passphrase",
}
```

Confidential Workloads uses the same Key Broker Service protocol as Confidential Containers (see below). As a matter of fact, the Confidential Workload project delivered the first implementation of that protocol, called reference-kbs. The blog article documenting workload attestation shows what the protocol looks like internally (this is essentially the same for all variants):

1. The KBC initiates the exchange with a request that provides protocol version number, the type of TEE (SEV SNP in the example), and additional parameters, which are unused for SNP.

```
{
    "version": "0.0.0", // Ignored.
    "tee": "snp",       // AMD SEV-SNP.
    "extra-params": "", // No extra parameters.
}
```

2. The KBS responds with a challenge that includes a nonce to prevent replay attacks:

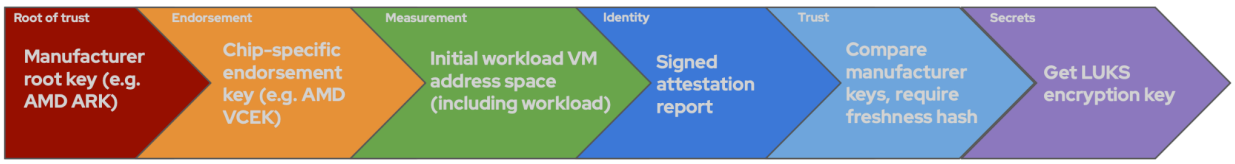
```
{
    "nonce": "vo4jefw854jh5x8ff39f8fh47hf4908fc38u",
    "extra-params": "",     // Also unused in SEV-SNP.
}
```

3. The KBC responds to the challenge with a structure that includes the TEE public key, which the attestation server can use to encrypt secrets it sends back, as well as the attestation report and a “freshness hash” built from the nonce.

```
{
    "tee-pubkey": TeePubKey {
        "alg": "RSA",
        "k_mod": "base64-encoded-rsa-public-modulus",
        "k_exp": "base64-encoded-rsa-public-exponent",
    },
    "tee-evidence": {
        "report": "hex-encoded-attestation-report",
        "cert_chain": "[]", // Attestation server's responsibility
        "gen": "milan"
    },
}
```

4. The KBS will validate this response, checking the freshness hash built from the nonce, checking that the AMD-provided keys match what can be found on AMD servers (there are separate certificates for Naples, Rome, Milan and Genoa generations), extract the versioned chip endorsement key (VCEK) and validate the entire certificate chain, then compare the measurements with what was registered with the server.

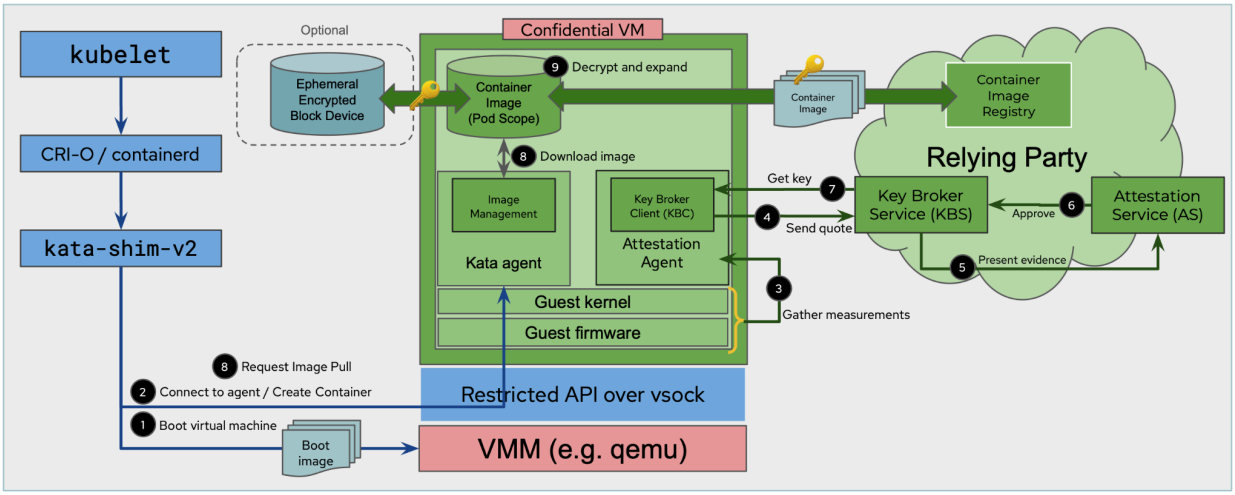
The REMITS pipeline for confidential workloads therefore looks like this:



Confidential Containers Attestation architecture

The remote attestation architecture for the Confidential Containers project is described extensively in another blog article. It evolved significantly over time, with the objective to become more and more generic and to improve performance and reliability. There is good documentation for the attestation service as well as for the key broker service and the underlying hardware-independent protocol.

The overall diagram for Confidential Containers can be found below:

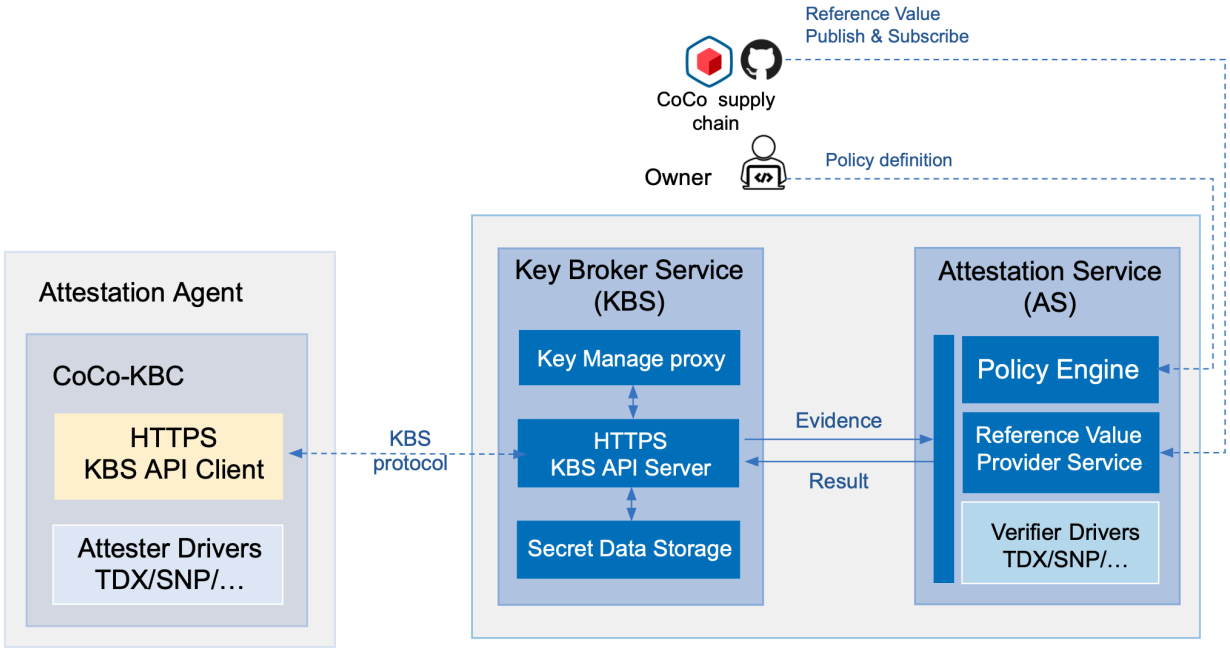


As in an earlier diagram, the components in blue are owned by the host, the components in red are part of the confidential platform, and the components in green form the trusted domain. Data that is on the host in encrypted form uses a blue/green shade.

The part that is relevant for attestation starts with the attestation agent, which will collect a measurement from the hardware through a kernel / firmware interface. Like for a traditional CVM, the root of trust and endorsement are backed by CC hardware, for example an AMD root key and chip endorsement key. From the measurement, the attestation agent builds a quote, presented as evidence and proof of identity to the attestation service. The attestation service will then validate that quote according to its own policies, affirming or denying trust into that quote. If the attestation is successful, then the attestation service will authorize a key broker service to release secrets. The attestation agent in the virtual machine can then use it to decrypt the container image layers (including encrypted layers).

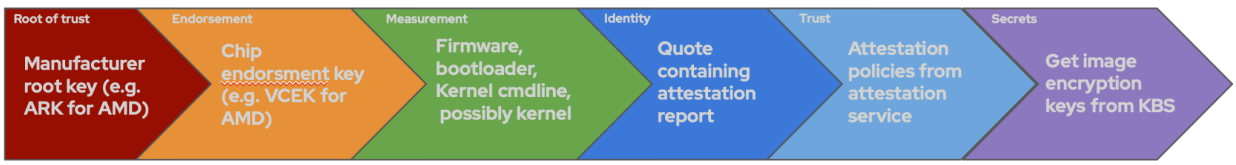
This attestation architecture is currently being revisited so that more code can be shared across platforms. The diagram below shows what is currently being implemented. On the left, the attestation agent is the part that runs inside the confidential virtual machine. In this design, it

talks to the key broker service, which will relay the evidence to the attestation service. This is mostly to simplify the network data path. The diagram also shows important additional aspects from a management point of view, which is the need for the owner to be able to set or update policy definitions, and for the image building process part of the supply chain to publish reference values as new container images are being built.



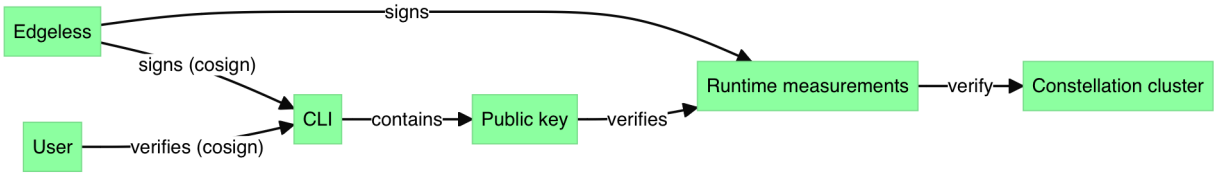
The earlier design allowed multiple key broker services and key broker clients to coexist. However, the only part that is really expected to change from platform to platform are drivers that, on the agent side, collect measurements, and on the verifier side, validate the measurements embedded in the quote.

The REMITS pipeline for confidential containers therefore looks like this:



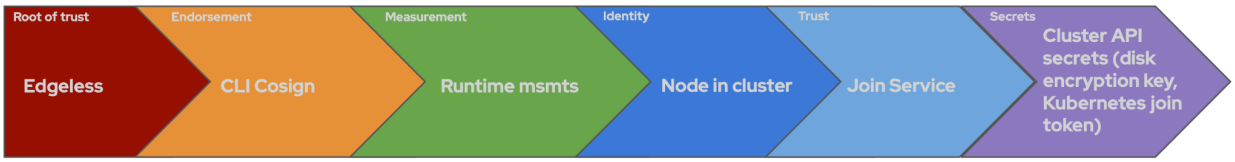
Confidential Clusters

Attestation for confidential clusters is well documented for Edgeless Constellation. They illustrate the chain of trust as follows:



In the Edgeless Constellation case, *runtime measurements* combine infrastructure measurements, like those provided by the confidential virtual machines, and measurements produced by the bootloader and boot chain.

For this particular chain of trust, the REMITS pipeline looks like this:



The Constellation design also includes a service to provide secrets, notably storage encryption keys. They can also use an external key management system (KMS).

However, confidential clusters are one of the scenarios where multiple chains of trust need to be considered, with different personas consuming the resulting trust assessment. Notably, they need to distinguish between cluster-facing attestation (to guarantee that a non-confidential node will not join a cluster, allowing user data to leak through that non-confidential node), and user-facing attestation (to verify the identity and integrity of a cluster before deploying workloads on it).

User-facing verification lets the user verify a cluster, although the result could also be consumed by a tool. The user collects the signed measurements from the configured image (Edgeless offers a public registry for measurements), and can verify the measurements using Edgeless' public key. This can be done either during configuration of the cluster, or at any later time.

```
constellation config fetch-measurements
constellation verify [--cluster-id ...]
```

In that scenario, the result of attestation is in clear, user-visible format, and the user is responsible for analyzing the data and taking action. This is why there is no secret:

Conclusion

We proposed a simple REMITS pipeline, which allows us to have a common way to present the variations in attestation model between the various use cases. Another big source of variation, however, is the differences between hardware platforms. This will be the topic of the next article in this series.

Part 5 - Confidential Computing Platform-Specific Details

This article is the fifth in a 6-part series, where we present various usage models for Confidential Computing, a set of technologies designed to protect data in use, for example using memory encryption, and the requirements to get the expected security and trust benefits from the technology.

In the whole series, we will focus on four primary use cases: confidential *virtual machines*, confidential *workloads*, confidential *containers* and finally confidential *clusters*. In all use cases, we will see that establishing a solid chain of trust uses similar, if subtly different, *attestation* methods, which make it possible for a confidential platform to attest to some of its properties. We will discuss various implementations of this idea, as well as alternatives that were considered.

In this article, we will explore the many available Confidential Computing platforms, and discuss how they differ in their implementation details, and specifically in how to perform attestation. The platforms of interest are:

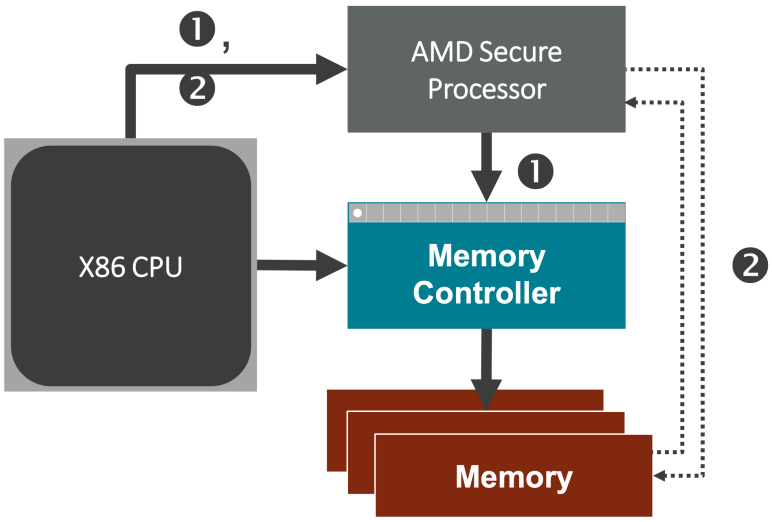
- AMD Secure Encrypted Virtualization (SEV) in its three generations (SEV, SEV-ES and SEV-SNP)
- Intel Trust Domain Extensions (TDX)
- IBM Z Secure Execution (SE)

- OpenPOWER Protected Execution Facility (PEF)
- ARM Confidential Compute Architecture (CCA)

At the moment, qemu officially supports AMD, Power and IBM Z, and Intel maintains branches for TDX. Confidential Containers has some support for SEV and SEV-ES (SEV-SNP being under development), IBM Z on SE, and Intel TDX. Cloud providers are in the process of deploying SEV-SNP support.

AMD Secure Encrypted Virtualization (SEV)

AMD provides Confidential Computing through a technology called *secure encrypted virtualization* (SEV). This technology builds on top of virtualization. It relies on a separate *security processor* running an independent firmware from the primary x86 cores.



Initial iterations of the AMD SEV technology only allowed *pre-attestation* (see chapter 6 in SEV encrypted virtualization API), in which a measurement of the memory content is done before a virtual machine is even started, the attestation process delivering a launch secret. In a first phase, the hypervisor loads the initial boot state in memory, then asks the firmware to encrypt and measure it. The measurement is sent to the attestation service, which responds with a *launch secret*. Only then will the hypervisor start running the guest. The guest can then use that launch secret in a guest-specific way, for example to access an encrypted root disk.

This only allowed a crude form of system-facing attestation, where the attestation process itself was not under control of the guest, but was executed by the host. Additionally, several serious vulnerabilities were found in this scheme (undeSErVed trust, Insecure until Proven Updated).

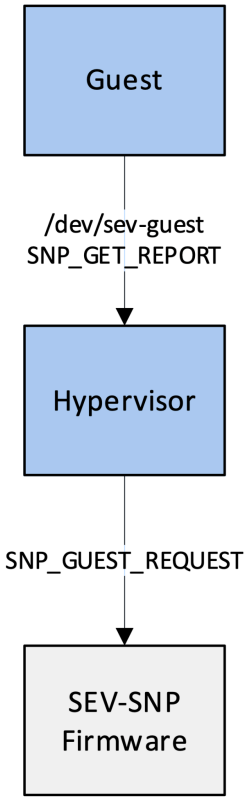
AMD SEV-Encrypted State (SEV-ES)

While the original SEV only encrypts the contents of guest memory, SEV-ES also encrypts the contents of the guest CPU registers, thus stopping leakage of in-flight values and more importantly stopping a malicious hypervisor manipulating the registers to read out encrypted data. It does not however significantly change the attestation model relative to the earlier generation.

AMD SEV-Secure Nested Pages (SEV-SNP)

The third generation of SEV, SEV-SNP, introduced in 2021 EPYC CPUs, added defenses against a malicious hosts manipulation of page mapping, but also made two changes that impact attestation:

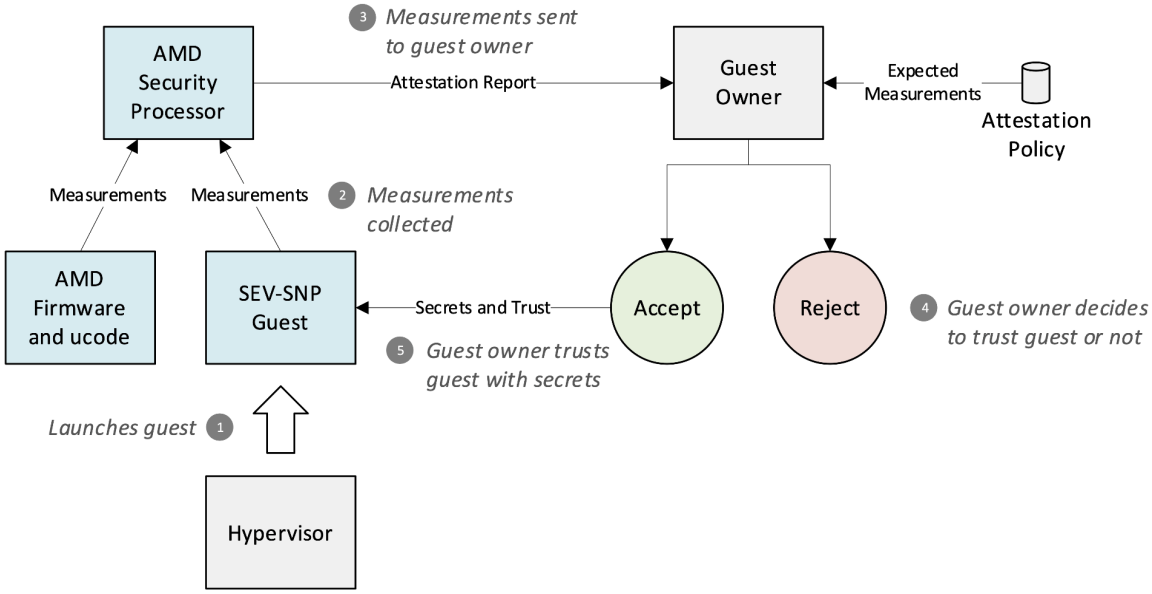
1. SNP provides a mechanism to obtain the attestation quote inside the guest, once the guest is running, whereas earlier forms of SEV had to perform the attestation from the hypervisor. This makes it much easier for the guest to pass a quote to the attestation server. Of course, the request is encrypted before being sent to the firmware, so that the hypervisor has no access to it.



man -k

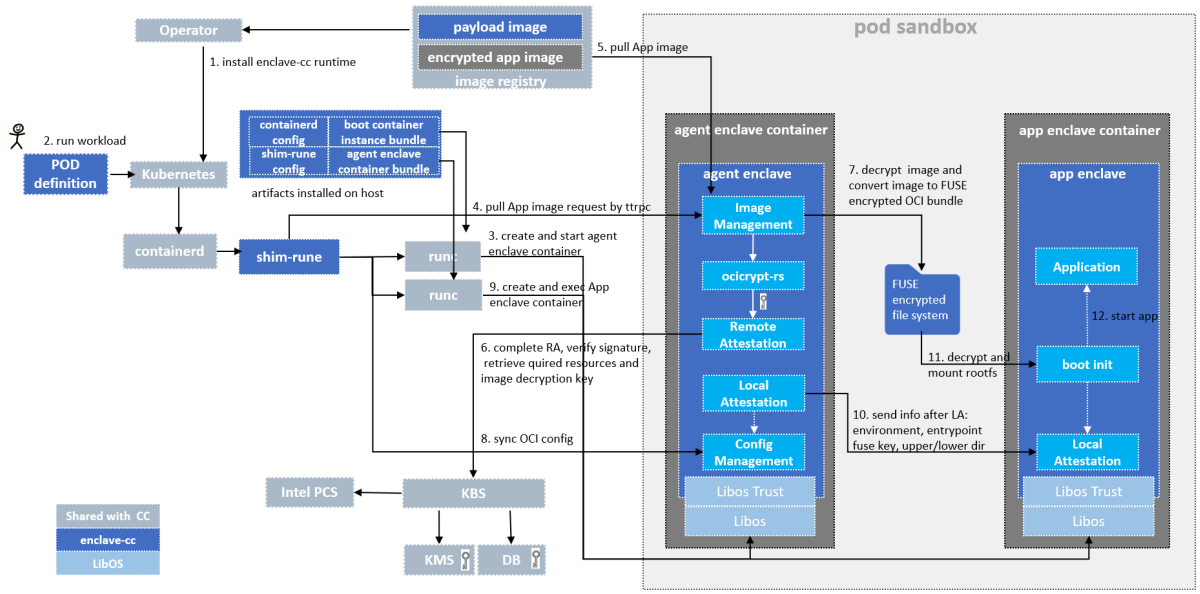
2. SNP includes an extra set of privilege level separation in the guest, VMPL, that allows the secure implementation of a virtual TPM, protected by the same SEV machinery protecting the rest of the guest. A new piece of firmware, the Secure Virtual Machine Service Module (SVSM) runs at the most privileged VMPL level and is planned to incorporate the vTPM implementation. Such support firmware and software will be discussed in more detail in the next article in the series.

The attestation mechanism in the case of SEV-SNP is described by AMD using the following schema:



Intel Software Guard Extensions (SGX)

Intel SGX provided process-level isolation, allowing the creation of secure *enclaves.* This makes it unique among the trusted execution environments being discussed here. There is some support for this technology in the Confidential Containers project, under the name enclave-cc. This approach supports both local and remote attestation, as illustrated below:

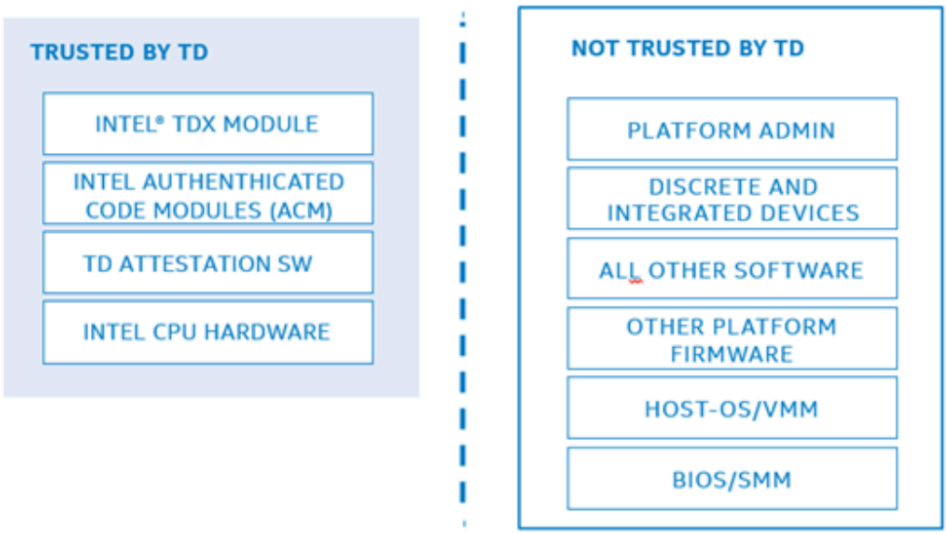


Because it is process-based, that technology requires some additional user-space trickery to be able to run (or emulate) multi-process workloads, i.e. the ability to fork new processes. This is marked as LibOS on the diagram above. On the other hand, it draws the boundary of the trusted computing base much closer to the container we want to run, since there is no independent kernel. This reduces the attack surface and resource requirements significantly.

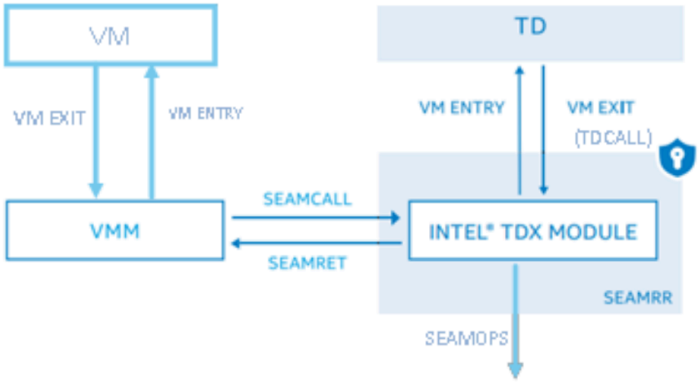
The SGX technology was deprecated in 2021, except on the Xeon line of processors. However, it plays a key role in TDX attestation, to provide a so-called quoting enclave, as discussed in the next section.

Intel Trust Domain Extensions (TDX)

Intel Trust Domain Extensions (TDX) provide features that are roughly similar to AMD SEV. It offers memory and state encryption and integrity to isolate trust domains (TD, confidential virtual machines) from one another, as well as from the host operating system and hypervisor. TDX also addresses state and interrupt integrity much like AMD SEV-SNP, as well as methods ensuring that vulnerabilities in the TDX firmware itself can be securely fixed, and that insecure firmware can be remotely detected.



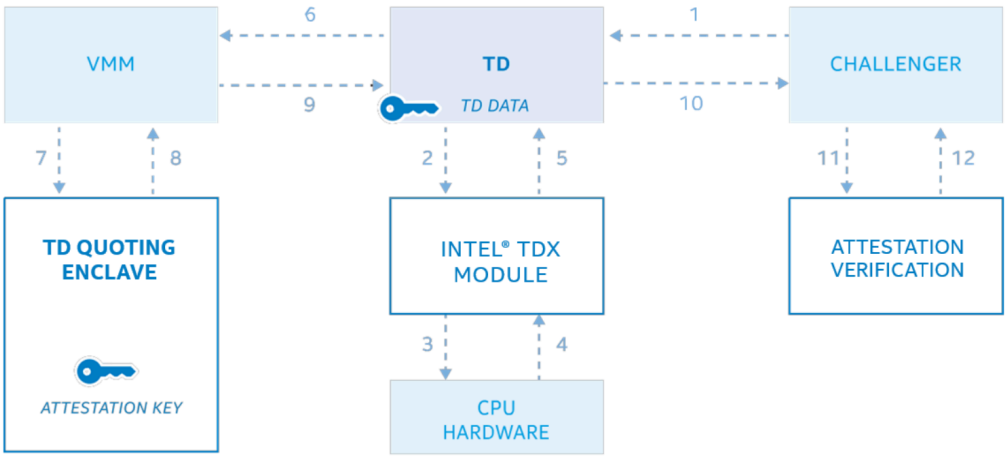
A major design difference compared to AMD's approach is that this is implemented using a new processor mode called *secure arbitration mode* (SEAM), rather than on a separate security processor. The memory used by SEAM mode is in a reserved protected range, defined by dedicated registers in the processor. The main x86 processor enters this special SEAM mode using new dedicated SEAMCALL instructions. In short, the design is based on protected firmware rather than a dedicated security processor.



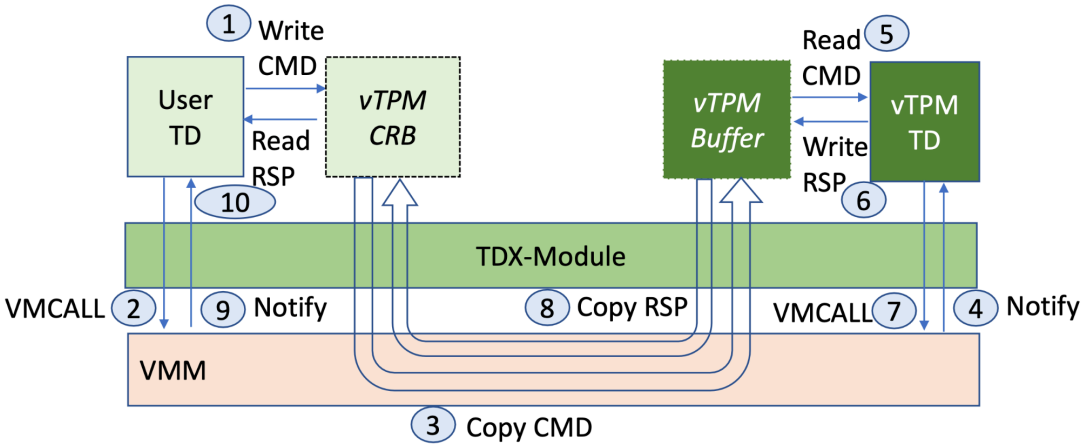
The calls are intended to invoke services provided by a special firmware *authenticated code module* (ACM) provided by Intel. The owners of a TDX system, for example a cloud provider, can define which versions of this module are acceptable. The integrity of this digitally signed module is verified using a TPM by Intel Trusted Execution Technology (TXT).

Since there is no security processor, the TDX architecture relies on a *quoting enclave*, which is an SGX enclave running code provided by Intel that takes advantage of a special instruction,

SEAMREPORT, to generate a quote for a given trusted domain. Intel illustrates this process as follows:



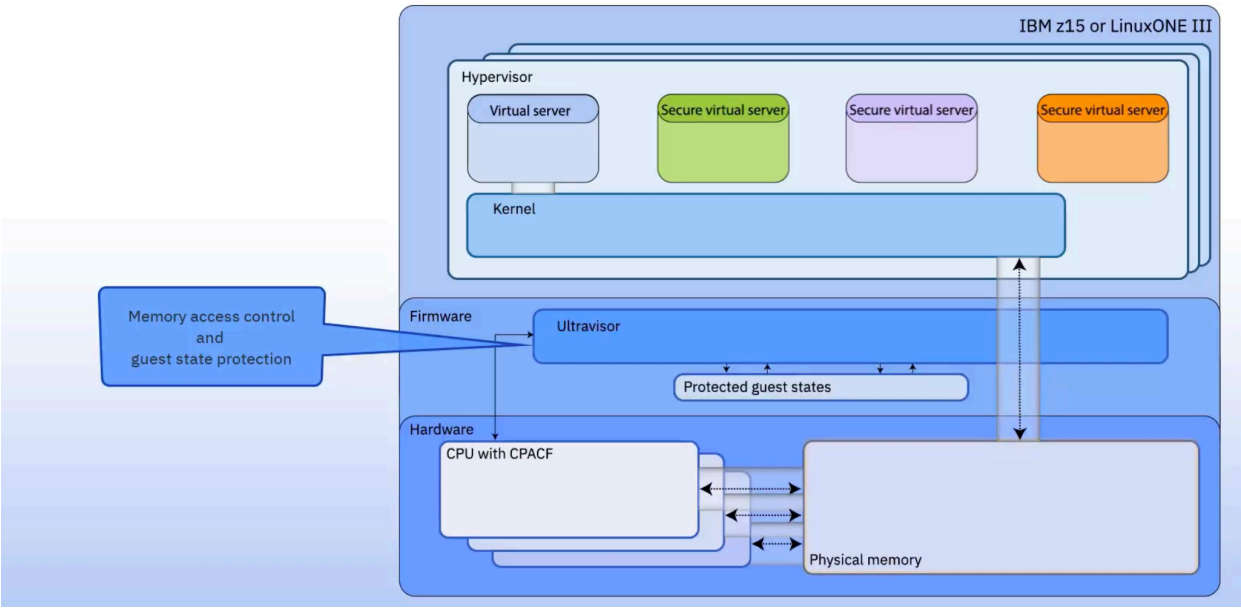
For the same reasons, providing a virtual TPM on a TDX platform will presumably use an enclave or a trusted domain. A design for such a virtual TPM based on a trusted domain storing the data for the various virtual TPMs has been presented by Intel:



Like for AMD-SEV, having a virtual TPM simplifies the deployment of existing workloads, in other words deployments that have not been enlightened specifically for TDX.

IBM Z Secure Execution (SE)

The IBM Secure Execution capabilities is available for the IBM z15 and z16 mainframes. It is a somewhat different design compared to what exists on x86, introducing a new level of system management in the firmware called the *ultravisor* that sits below the hypervisor.

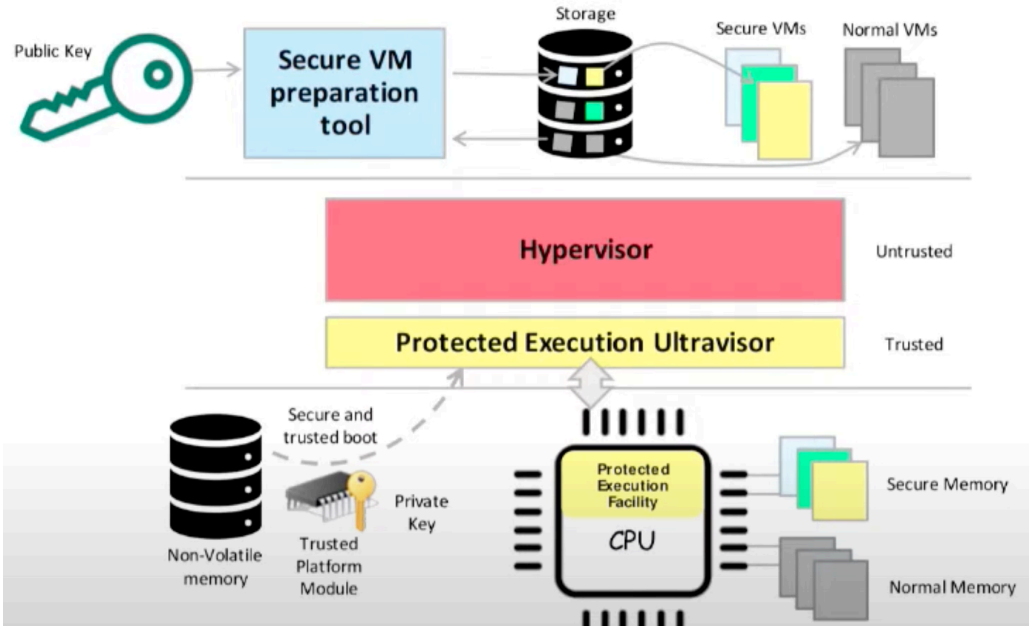


Hardware encryption of memory and remote attestation were introduced with the z16. On the z15, the ultravisor will block access from the host, but physical memory will not be encrypted. This design entrusts the firmware with control and execution of the workloads, which are encrypted and stored with a special SE header. This makes it possible to restrict execution to a particular host or set of trusted hosts.

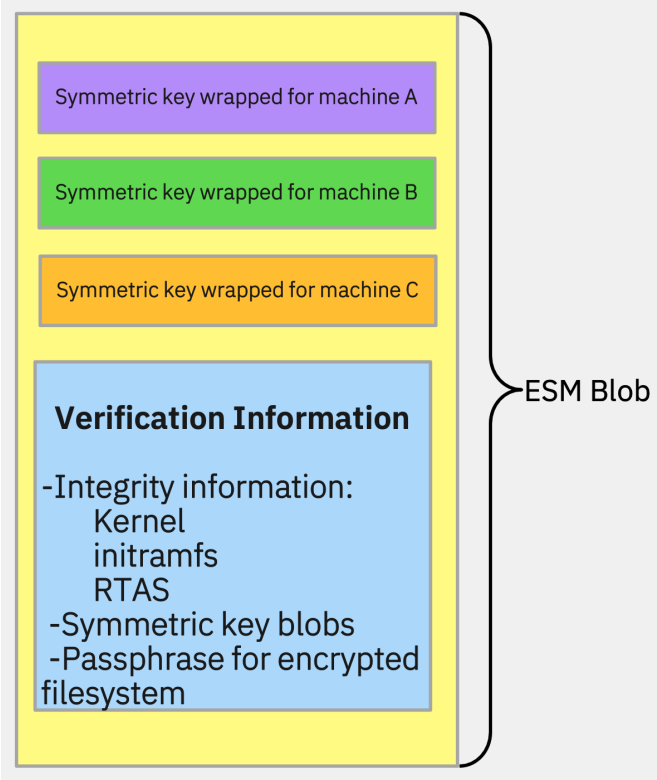
The attestation approach is also different. IBM's point of view is that "IBM Secure Execution does not require external attestation to prove that a guest is secure. If the image contains a unique secret, a successful login implicitly attests an SE guest image". However, they still added remote attestation in order to provide more flexibility, notably to prove a larger set of properties, allowing multiple instances of the same image with instance-specific secrets, presenting a proof to a third party, or proving some property of the execution environment.

Power Protected Execution Facility (PEF)

The Protected Execution Facility (PEF) on OpenPOWER is reminiscent of the Secure Execution on IBM z, in that it is based on an additional *ultravisor* protection level, with the difference that this firmware layer is open source. This platform uses the terminology *secure virtual machines* (SVM) rather than "confidential". In order to run properly on the ultravisor, the hypervisor needs to be paravirtualized, i.e. use ultravisor services for some operations using a new ESM instruction.

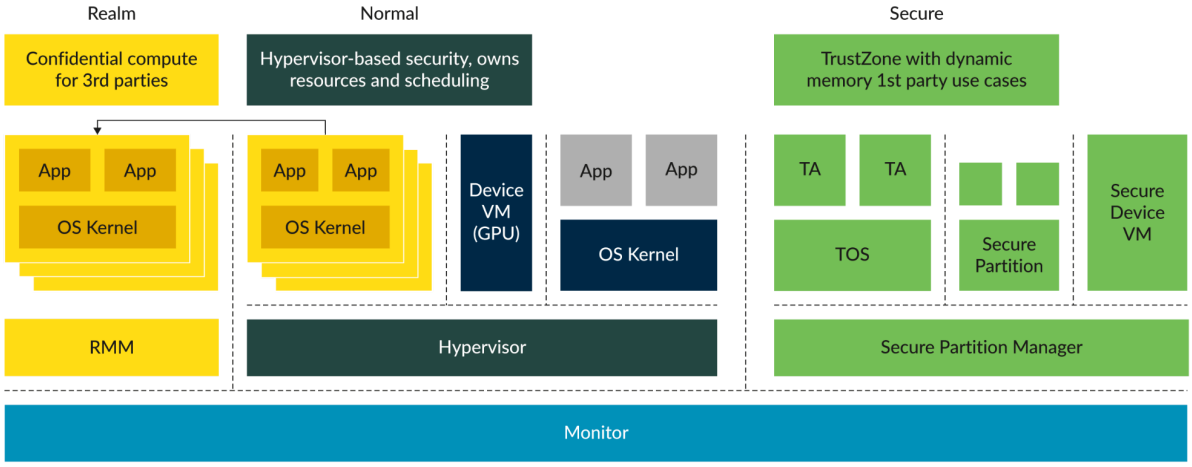


The boot process starts the VM normally, and then performs an ultravisor call to copy the VM address space into secure memory, then searches for a valid authorization key for a specific machine. So a bit like on IBM z, the base attestation model is to prove that you are running on one of the allowed machines. However, the SVM itself can perform a remote attestation, notably in order to implement revocation. IBM presentations highlight that the user, the owner or the host of an SVN all may be given the possibility to revoke an SVM.

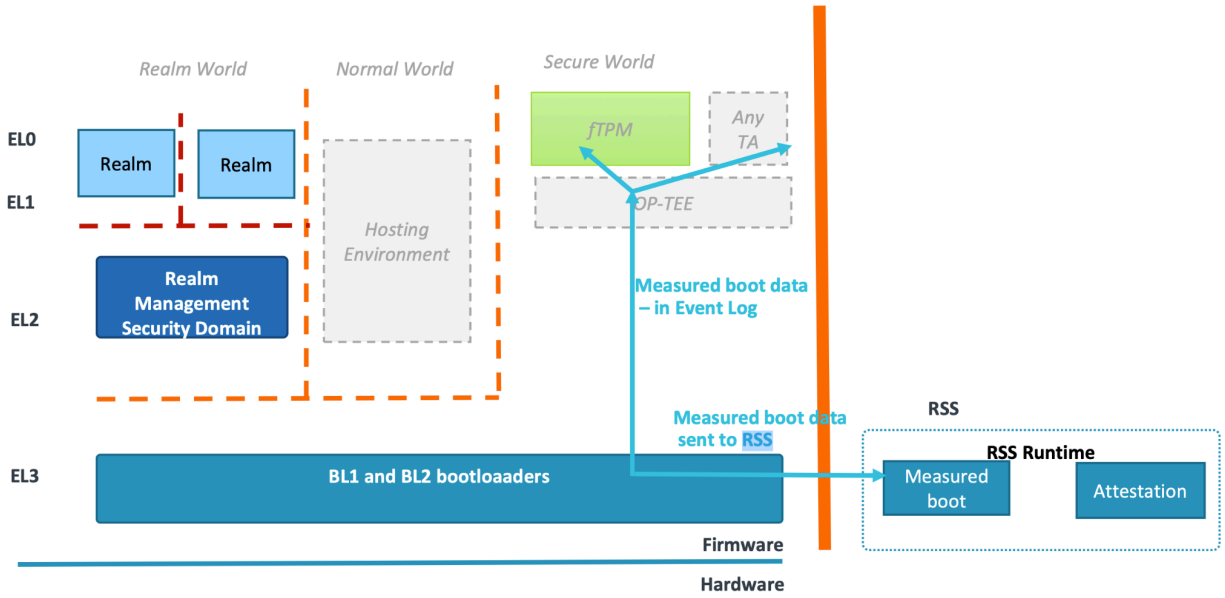


ARM Confidential Compute Architecture (CCA)

For the ARM platform, the confidential computing technology is creatively called confidential compute architecture (CCA). Like on Power, it is based on the introduction of a new exception level, EL3, sitting below the hypervisor, which enables a *monitor mode*. This architecture builds on ARM v8 TrustZone, which was used to split between a “Secure world” (to run trusted applications and operating systems) and a “Normal” world (for standard applications and operating systems). CCA introduces a new concept of *realm*, which is a partitioning of the address space in order to isolate realms from one another, controlled by a new data structure called the granule protection table (GPT), and managed by a new firmware component called the realm management monitor (RMM). The initial root of trust in the system is called the runtime security subsystem (RSS).



Attestation on CCA is [managed primarily by new firmware] that takes advantage of the new facilities, and can build the required attestation reports.



This architecture is still under development.

Conclusion

The available architectures take somewhat different approaches to the same problem, with different emphasis being placed on hardware, software and firmware responsibilities. All the solutions provide roughly the same confidentiality guarantees, with most often an ability to encrypt memory, measure it using cryptographic algorithms for attestation purpose, and offer some provable integrity guarantees about the initial state of a confidential virtual machine,

including the fact that it's running in a trusted execution environment. In the next article, we will go a little more in depth to understand some of the interesting support technologies where active open-source development is happening.

Part 6 - Support technologies related to Confidential Computing

This article is the last in a 6-part series, where we present various usage models for Confidential Computing, a set of technologies designed to protect data in use, for example using memory encryption, and the requirements to get the expected security and trust benefits from the technology.

In the whole series, we will focus on four primary use cases: confidential *virtual machines*, confidential *workloads*, confidential *containers* and finally confidential *clusters*. In all use cases, we will see that establishing a solid chain of trust uses similar, if subtly different, *attestation* methods, which make it possible for a confidential platform to attest to some of its properties. We will discuss various implementations of this idea, as well as alternatives that were considered.

In this article, we will explore interesting support technologies that are under active development in the confidential computing community, and may spark your interest.

Kernel, hypervisor and firmware support

Confidential Computing requires support from the host and guest kernel, as well as from hypervisor and firmware. The state of that support is quite uneven between platforms at the time of this writing. Hardware vendors tend to develop and submit relatively large patch series, which can take a number of iterations to get approved.

Among the active areas of development at the time of this writing are:

- Host kernel support for SEV-SNP
- Hypervisor, guest and host support for TDX (as well as a few ancillary firmware projects).
- Platform support for ARM CCA

The impact this has on the attestation process is primarily the appearance of multiple not-yet-stabilized interfaces to collect measurements about the guest, typically exposed as a

/dev entry with a variety of similar, but not identical ioctls. This is an area where standardization has not even begun.

Platform provisioning tools

Before running confidential virtual machines, it is necessary to provision the host. This provisioning corresponds to the “Endorse” step in the REMITS pipeline, and typically will generate a number of host-specific keys.

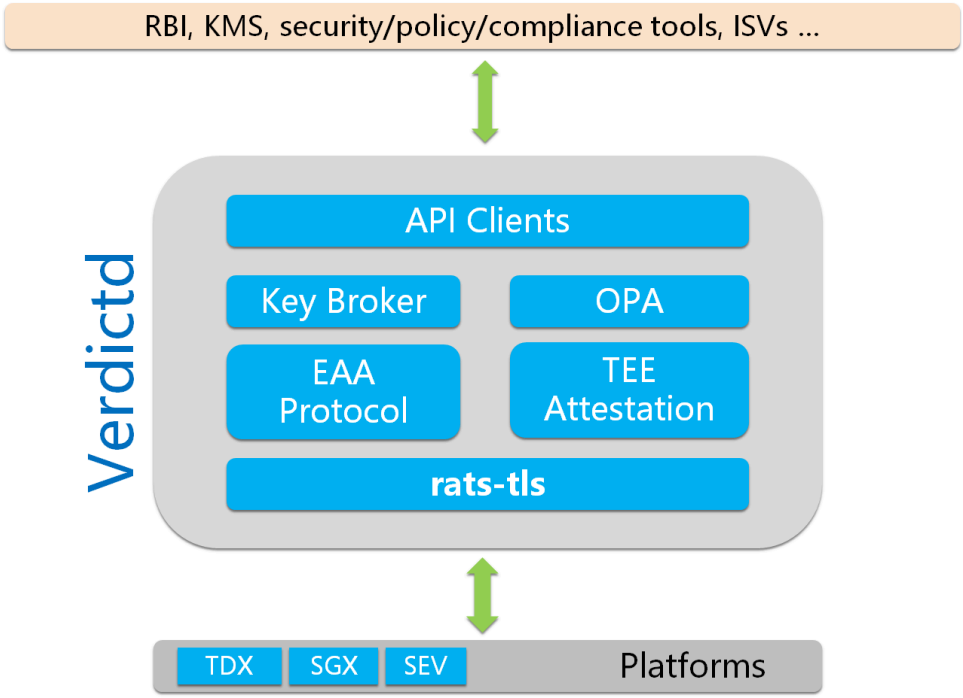
The tools to do that are highly platform-specific, and at least in the case of AMD, there are already two competing toolsets, sev-tool and sevctl. In the present state of things, there isn’t a single set of tools that can present a relatively uniform user interface for all platforms.

Generic Key Broker and Attestation Services

In part 3 of this series, we introduced a REMITS pipeline model that allowed us to compare and contrast various forms of attestation. In this model, the S stands for “Secrets”. The reason is that a good way to ensure that a non-trusted execution environment does not receive sensitive data is to tie its execution to secrets that can only be unlocked through attestation. This is a good reason to tie key or secret brokering to attestation, although the two are conceptually (and in most implementations) separate.

The Confidential Containers project defined a generic key broker service (KBS) which is the primary access point for the agent running in the guest. The KBS relies on the attestation service to verify the evidence from the TEE. This is still a very active area of development, with the objective to make the overall design more modular, and to be able to support more hardware platforms through platform-specific drivers (both on the attestation server side and on the client side).

The Inclavare Containers project developed its own attestation infrastructure, called verdictd, which also integrates with the key broker, incorporates open policy agent support, and is based on their rats-tls project, an implementation of remote attestation procedures (RATS) framework using transport layer security (TLS).

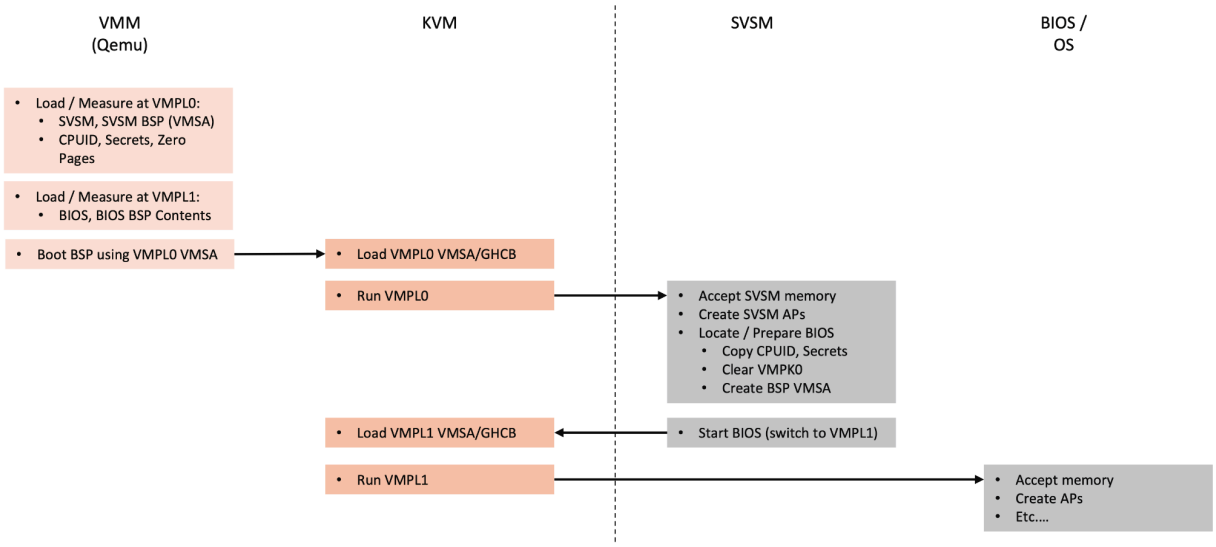


While there are a number of products such as Keylime that are pure attestation players, they were not deemed suitable precisely because they are not designed to act as a synchronous, blocking attestation enforcement mechanism, delivering secrets on success. Instead, Keylime was designed to analyze the compliance of machines in a fleet for manual or semi-automated operator actions.

There was already some sharing of this key broker service interface between Confidential Workloads and Confidential Containers. It is desirable, and appears at the moment possible, to extend this service to suit the needs of confidential virtual machines and confidential clusters as well.

Secure Virtual Machine Service Module (SVSM)

The Secure Virtual Machine Service Module (SVSM) is a new piece of firmware that runs at the most privileged virtual machine privilege level (VMPL0) and can procure secure services that will be used by a secure virtual machine running with reduced privilege.



The primary reason to implement such privilege services is to implement emulation for older interfaces, notably the standard TPM, or to run guests that are not aware that they are running in a confidential virtual machine.

In order to build a virtual TPM that the guest cannot tamper with, it is necessary to protect the vTPM code and data from a malicious guest or hypervisor. Functionally, this is roughly equivalent to what would be monitor code in ARM CCA, hypervisor code in SE or PEF, and a vTPM enclave or TD in TDX.

A draft SVSM specification has been published for review.

Virtual Trusted Platform Modules (vTPM)

While the original TPM was a physical device attached to a physical machine, with the advent of virtual machines, there was a desire to provide similar facilities to the virtual machine guests.

In the Confidential Computing world, the abstraction of the vTPM provides a convenient unified measurement mechanism across multiple Confidential Computing implementations while allowing existing TPM tools to be reused.

Prior to Confidential Computing, a vTPM was often implemented as a separate process or module on the hypervisor; this ensured that the vTPM state was protected from attack by the guest. In the Confidential Computing world however, the vTPM state must be protected from both the host and guest. In the case of SEV-SNP, this can be inside the guest firmware and protected by the use of VMPLs. Other designs use a vTPM running in a separate confidential

VM, TD or realm depending on the technology. Part of the problem is how to connect this vTPM securely to the confidential VM that uses it.

Another challenge with vTPM in a Confidential Computing environment is how to store the vTPM non-volatile state securely without placing trust in the host or surrounding cloud environment. Some implementations sidestep the problem by providing an 'ephemeral vTPM' with new state generated on each boot which may limit the use of some existing TPM tools.

Since both the vTPM and the underlying Confidential Computing technology can produce attestations there needs to be a way to tie them together to prove to an attestation system that the system attested by the vTPM is really running on confidential compute hardware. Ways to do this include:

- Mixing a hash of the vTPMs keys (In particular the endorsement key, EK) into the confidential compute attestation
- Hashing the confidential compute attestation into a vTPM PCR.

Note that the TPM interface tends to provide more capabilities than the simpler measurement registers found in some confidential computing platforms. Not all features are necessarily relevant for confidential computing.

Conclusion

The Confidential Computing ecosystem is an area of intense research and development at the moment. Many pieces need to cooperate to achieve the objective, and no two platforms do it exactly the same way. Even the supporting tools differ from platform to platform.

We can hope that the overview presented in this series of articles will help the readers navigate this very complex landscape.