The Certifier Framework for Confidential Computing Whitepaper

Summary

Confidential computing offers a set of platform-based mechanisms for protecting the confidentiality and integrity of software, and for establishing a robust notion of trust. These protections are effective even in the presence of malware on the same computer and even if the data center administrators, in the datacenter in which the protected software runs, are malicious. Confidential Computing supplies the only principled security mechanism to protect programs no matter where they run (any public cloud or "in-house") as well as providing remote verification of these properties over a communications channel. It is a foundational technology for multi-cloud security.

Confidential Computing provides strong, uniform security for programs and data across the multi-cloud. Beyond significant improvements to general security, confidential computing also can underpin novel "data economy" workloads, since it provides assured confidentiality and integrity for data and processing no matter where a confidential computing program runs, even in data center is operated by a third party. For the first time, it will be possible to provide an infrastructure that gives consumers and enterprises principled, assured, control over their data and how it is shared with others locally or remotely.

Confidential computing platforms relies on hardware including AMD's SEV and Intel's SGX (and forthcoming TDX) as well as similar mechanisms being developed for RISC-V and ARM processors. As explained below, Confidential Computing is defined by the ability of a platform to (1) isolate the program from all other software, (2) measure the program, providing a universal unforgeable identity, (3) protect and recover secrets when requested by the program and (4) attest to statements from the program (i.e.-sign statements from the program with the program measurement). These facilities are being also extended to GPUs in support of important workloads that require external acceleration.

To accelerate the adoption and productive use of confidential computing throughout its enterprise ecosystem, the Certifier Framework for Confidential Computing contains a simple but comprehensive set of libraries and service that allow programmers to develop safe confidential computing programs using a half dozen calls. In order to meet enterprise requirements, while simplifying the work of the developer, these provide key management, secure storage, trust negotiation using declarative policy management. The framework also allows organizations to safely deploy and manage distributed Confidential Computing solutions across a multi-cloud in a resilient and agile manner that enables upgrade and uniform "policy domain wide" enforcement. All this is done without affecting standard software deployment techniques. In addition, the Certifier Framework for Confidential Computing allows programs

to target any Confidential Computing platform without changing the program or complicating the policy management.

The Certifier Framework for Confidential Computing takes the “devil out of the details” in Confidential Computing and allows organizations to develop, deploy and manage distributed applications, whose security is guaranteed in a principled, verifiable manner with a minimum of effort and welcomed flexibility.

What is Confidential Computing?

Confidential computing provides principled, verifiable security mechanism for distributed computing authenticating a program, protecting the integrity and confidentiality of its processing and the data it receives, generates, stores, or transmits **wherever** the program runs. Protected programs (and their data) cannot be read or modified by other programs on the platform even under the intervention of malicious of careless insiders (e.g., admins).

Four capabilities provide the basis for Confidential Computing:

- Isolation: The program address space is isolated (i.e.-can’t be read or modified) from all other software on the platform. This isolation is verifiable and carries very high assurance.
- Measurement: A unique, unforgeable program identity, computed when the program is initially loaded, typically a cryptographic hash of the entire program (code and data) and any related data that could change program execution.
- Secret storage: The ability to save and store secrets that can only be obtained by the measured program on the platform when isolated. (aka, “sealed storage”).
- Attestation: The ability of the platform to make cryptographically signed statements on behave of an isolated program whose measurement is part of the signed statement. This is the critical mechanism to establish trust between distributed programs in Confidential Computing.

Most people understand the first three properties and how they contribute to secure computing; the final capability, attestation, is less well understood. Attestation within confidential computing addresses the problem of how one can establish trust in the hardware platform as well as the software running in it. The notion of "trust" here does not refer to the intentions of software authors or the correctness of software functional specs, but rather the identity of the software that is running on the system and associated guarantees that the software has not been tampered with in some manner.

Trust negotiation begins with a set of *claims* that are assembled into an evidence statement. Each claim is signed by a key and is hence can be verified. Confidential Computing adds a new kind of claim, called an attestation, this claim includes an assertion (usually a public key), the unforgeable measurement of the program, and details about the underlying hardware; an attestation is signed by the hardware platform. A simple example of an attestation is:

> "The program whose measurement is M[0x3833….44], which has been isolated and measured by a certified TDX platform, asserts that the public RSA key PK [Modulus: x, Exponent: y], authenticates the program." --- Signed hardware attestation key

Upon receipt of evidence, a verifier will examine the submission and compare it against policy to determine whether the entity submitting it should, under the policy, be *trusted*. The policy includes what measurements are trusted, what hardware is trusted, and what permissions programs have once verified. For example, after the evidence, including the above example attestation is verified, the recipient is certain that:

1. Any statement signed by the public key can only come from the indicated program
2. The program, as loaded, has not been modified and no other software on the platform can read or write in its address space
3. The program is isolated
4. The program is trusted under the security policy
5. Secure communications protected using protocols (like TLS) employing the indicated public key are confidential and integrity protected.

Knowing this and given a complete understanding of the subject program, completely provides complete assurance as it relates to program execution as well as the confidentiality, integrity, and reliability of the data it processes, stores or transmits.

Trusted Execution Environments (TEE)

Trusted Execution Environments ("TEE's") are hardware platforms, or enclaves within platforms that implement the four Confidential Computing primitives. There are two classes of TEEs that are common: application-level TEE's (sometimes called "enclaves") and Encrypted Virtual Machines. Application-level TEE's host and isolate a single application (or part of an application) while, as the name suggests, Encrypted Virtual Machines host and isolate a full VM. Application-level TEE's are often written using an SDK, like Open Enclaves or Gramine.

Intel Software Guard Extensions ("SGX") represents the most popular application-level TEE. The Certifier Framework supports SGX using either of two different SDKs: Open Enclaves and Gramine. SGX enabled processors are capable of reserving memory for applications. This memory is protected using encryption and other processor features to provide the four Confidential Computing primitives.

The Certifier provides full support for SEV-SNP natively. AMD Secure Execution Virtualization ("SEV") and Intel Trust Domain Extension ("TDX") provide hardware support for isolated virtual machines. These VMs are isolated from the hypervisor and other untrusted software on the platform and, again, "isolated VM's" provide the four Confidential Computing primitives using slightly different approaches:

AMD EPYC processors contain a Secure Processor that provides cryptographic functionality for secure key generation and management. They also contain an AES-128 encryption engine embedded in the memory controller. The engine can automatically encrypt and decrypt data in main memory when an appropriate key is provided.

In Intel TDX, hardware isolated VMs are called Trust Domains (TDs). These TDs include:

- Secure-Arbitration Mode (SEAM) – a new mode of the CPU that can host digitally-signed, security services module
- Multi-key, total-memory-encryption (MKTME) engine to provide memory encryption (AES-128-XTS) and integrity using 28-bit MAC and TD ownership bit
- Secure EPT[1] to translate private guest physical address with integrity
- Physical-address-metadata table (PAMT) to track page allocation, initiation and TLB consistency

New Applications enabled

An illustrative example

We start with an example application that would difficult, if not impossible, without Confidential Computing.

Suppose three parties want to analyze all their fraud data to spot fraud trends, but,

- No bank wants another bank else to see its fraud data.
- No bank completely trusts any third party.

Confidential computing to the rescue: All three parties get together and agree on the analysis and what can be released from the analysis. The analysis is done by an Analysis program created/selected by the three parties in collaboration protected by Confidential Computing. Each can examine the Analysis program to ensure security properties are met. Each bank "measures" the analysis program for later reference and the public key of each bank is embedded in the analysis program so it can verify and safeguard communications with the bank.

The analysis program is started on a confidential computing platform. When it starts, the analysis program

- Generate a public/private key pair (PK/pK).
- Generate (symmetric) storage keys (SK).
- Seal these the private keys and store sealed material for later use

[1] See, for example, Intel® Architecture Instruction Set Extensions and Future Features Programming Reference for definitions and details.

Each bank communicates with the analysis program which provides an attestation naming PK. The bank checks the measurement in the attestation to make sure it's the right program. If so, The bank authenticates itself to the analysis program using its public key, generates an encryption key, K, and encrypts K with PK and sends it. The analysis program decrypts the bank data and re-encrypts and stores the data for later use. Finally, the bank encrypts all its data with K and sends that to the analysis program. The security implication is

1. Each bank verifies the analysis program before data transfer
2. Data is encrypted at all times (at rest and in transit)
3. Only the analysis program can recover unencrypted data

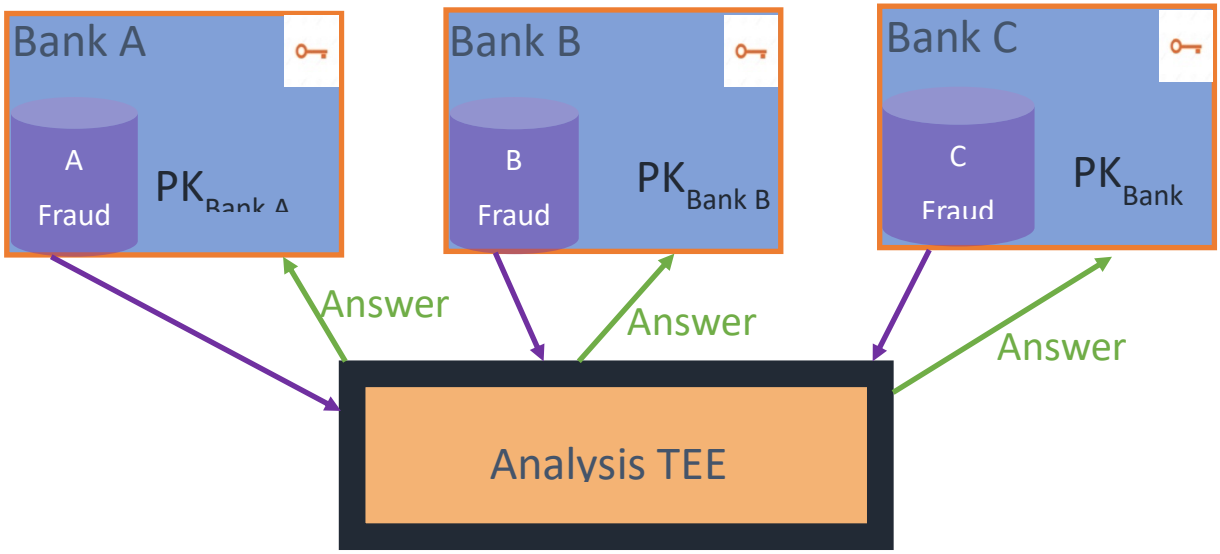

Figure: Bank fraud analysis

Once the data from all the banks is received, the analysis program does the fraud analysis an transmits (encrypted to each banks public key) the results. Easy!

More applications

Secure Kubernetes Containers Anywhere: The "pods" that host Kubernetes containers can be protected by Confidential Computing. This provides secure container management in the multi-cloud.

Secure Authentication, Authorization and Auditing: Critical authentication and authorization services can be securely hosted anywhere in the multi-cloud avoiding vulnerabilities such as Solar Winds.

Soft Hardware Secure Modules: We have implemented a soft-HSM using Confidential Computing. This provides HSM functionality and protection on capable hardware platforms.

Secure Collaborative machine learning: In the "data economy," programs acting on behalf of different parties use shared data repositories or data repositories belonging to other parties for learning. Confidential Computing provides a mechanism to ensure data integrity and provides technical guards to enforce restrictions on data use on behalf of the data owner. The "bank" example above is a simple example of this.

Secure data distribution: Confidential Computing provides the technical safeguards to share sensitive data while ensuring only authorized use. Examples include personal information used in delivering services or health information.

Secure Transactions: Confidential Computing can provide high assurance guards that control critical information whose correct use many parties rely on. Examples are high frequency trading, distributed ledgers and payments.

Classified data processing: Confidential computing provides principled protection for classified data processing.

Authorized action and audit guards: Confidential computing can serve as a host to control sensitive actions like controlling dangerous equipment or ensuring audit events are posted securely when required.

Just make everything safer: Almost all well written programs will enjoy improved security by making them "confidential computing capable."

Why does Confidential Computing change the game?

Prior to confidential computing running programs securely on machines not in the direct control of a party (or even in the control) was either not possible at all or extremely error prone. Indeed, to run a program securely, one must:

1. Write the program correctly (no exploitable vulnerabilities)
2. Deploy the program safely (no changes)
3. Configure the operating environment correctly
4. Ensure other programs can't (or don't) interfere with safe program execution
5. Generate and deploy keys safely
6. Protect keys during use and storage
7. Ensure data is not visible to adversaries and can't be changed in transmission or storage
8. Ensure trust infrastructure is reliable
9. Audit to verify this all happened

While number 1 above is in the control of a diligent developer, the remainder relies on the good intentions and error free execution of the platform operator (The IT group, Amazon, Azure, Google, etc). So, there is really not a principled reason to believe these programs (or their data), as run, are secure.

With confidential computing, provided the program is correctly, one can provide exactly the security assurance desired.

However, writing secure Confidential Computing Platforms can present challenges:

1. A program must be custom written for each platform (SGX, SEV, TDX, etc).
2. One must manage program migration and data migration; one must provide for general error-free policy which allows one confidential computing protected program to "trust" and interact with another one.
3. One must security store and recover program secrets, one must ensure a secure, authenticated, mechanism to communicate with other "trusted programs" in its security domain.
4. One must provide for (secure, resilient) data distribution, cryptographic support, scalable, efficient distribution, and management (including the ability to upgrade programs).

This involves thousands of lines of security code and is fraught with the potential for errors. For example, if one "hard codes" trust policy in a program, the program can only be used in a single policy domain, the program must be changed and redeployed every time a new trusted program is introduced to the environment and then one must deal with platform differences.

The Certifier Framework for Confidential Computing takes "the devil out of the detail" by providing library and service support to achieve all the Confidential Computing promise for a well written program by adding about half a dozen calls to our libraries and using our service.

The Certifier Framework for Confidential Computing components examined

The Guide in this Doc directory contains more detailed information about writing an application, deploying it and managing it with the Certifier Service.

Performance

The certifier library itself is fairly small and consists of about 290 KB of compiled code. Of that, 200 KB is machine generated serialization code (i.e., protobuf support). The certifier adds almost no overhead during program execution except during initialization. Overhead during initialization is dominated by the execution time needed for a dozen or so public key cryptographic operations essential to the attestation framework.

The Certifier Service

The *Certifier Service* consists of, possibly many instances of the *Certifier Service* server (written in Go). This server has the private policy key. It accepts connections from programs which send

"evidence packages" for evaluation; this is called "certification". The server verifies the signatures and proofs and, if acceptable, issues an X509 certificate. Note that upgrading an application is easy: simply recertify with the new version of the program and use the certificate to retrieve storage keys for any existing data from the old version of the program or a secure centralized repository protected by Confidential Computing.

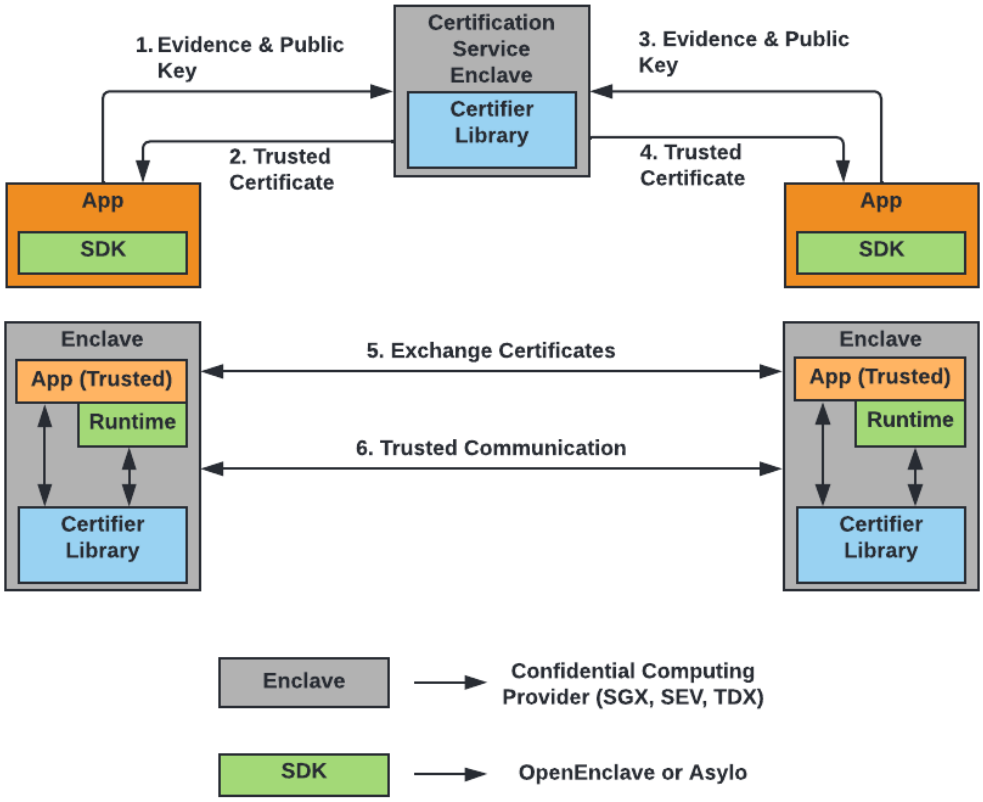

Figure: Certifier Service Architecture. Note: Encrypted Virtual Machines do not require an SDK.

Scalability can be achieved simply by increasing the number of distributed *Certifier Service* instances. Deployment models remain unchanged. So, almost any program distribution or resource assignment regimes could be effectively used with the *Certifier Service* model.

Certifier Service servers can also be grouped and employ group signatures so no single compromised server can destroy the assurance. The *Certifier Service* instance will have information about all the trusted hardware and trusted client programs. It can be used for revocation after certificates are issued but it is much more common to issue "short term" certificates and re-certify frequently since certification is very cheap and scalable.

Certifier Service servers can also be deployed in many "resilience zones" to avoid issues related to server or infrastructure failures.

Further comments

Since "economy of code and function" is fundamental to the effective use of secure enclaves, one may wonder whether cross-platform support increases the attack surface of the certifier. Broadly, we believe the answer is no. Each program will use a single SDK and platform at execution time, and only the relevant SDK and platform interface will be used during execution. The *certifier* is almost entirely independent from any given SDK. Thus, the certifier's cross-platform support does not add additional attack surface to a through an expanded trusted computing base (TCB).

The safest Confidential Computing programs make use of protected modules of limited size and complexity to reduce attack surfaces and the potential for exploitable vulnerabilities. This usually means that they don't load additional code during execution and only use well-curated libraries. Under these circumstances, there is little benefit in continuously attesting enclaves since, by construction, they are immutable. New functionality can be achieved either by adding new enclaves or by restarting an enclave after it is upgraded and using the certifier service's trust infrastructure to allow the new version to obtain existing secrets from the older version of the enclave.

An astute observer will note that "scaled deployment" depends on a highly secure *Certifier Service* infrastructure, and that a compromised certifier service could be catastrophic. Fortunately, the certifier service can make use of modern cryptographic techniques including quorum-based signing of certificates (so that a single compromised server becomes harmless) or partitioned service authorization zones that constrain which servers can "authenticate" which services. Finally, all certifier actions result in concrete cryptographic artifacts which can be logged, monitored, and analyzed. So, security surveillance can be lightweight and comprehensive.

The certifier framework comes with several utilities to generate service keys, and to format and sign policy statements.

The safest Confidential Computing programs make use of protected modules of limited size and complexity to reduce attack surfaces and the potential for exploitable vulnerabilities. This usually means that they don't load additional code during execution and only use well-curated libraries. Under these circumstances, there is little benefit in continuously attesting enclaves since, by construction, they are immutable. New functionality can be achieved either by adding new enclaves or by restarting an enclave after it is upgraded and using the certifier service's trust infrastructure to allow the new version to obtain existing secrets from the older version of the enclave.

Both the *certifier* and the *Certifier Service* come with a policy language and policy evaluation engine. Our policy language based loosely on the Lampson-Abadi “says/speaks-for” framework. We support many claim formats and, if you prefer, you can “plug in” an alternative policy

engine that uses a different policy language. You can also employ other verification frameworks like OPA or Datalog as noted in the code.

Here is a simple example of a trust policy together with evidence supporting a proof. The policy consists of statements 1, 2 and 5. Statements 3 and 4 are evidence. Statement 4 is an attestation in our stylized format.

1. Key[rsa, policy-key, be...]is-trusted
2. Key[rsa, policy-key, be...] says Key[rsa, intel-key, ab...]is-trusted-for-attestation
3. Key[rsa, intel-key, ab...] says Key[rsa, local-attestation-key, d0...]is-trusted-for-attestation
4. Key[rsa, local-attestation-key, d0...] says Key[rsa, enclave-key, da...]speaks-for Measurement[...]
5. Key[rsa, policy-key, be...] says Measurement[...]is-trusted

Here is a proof verification from the policy evaluation engine.

1. Key[rsa, policy-key, be...] is-trusted AND Key[rsa, policy-key, be...]says Key[rsa, intel-key, ab...] is-trusted-for-attestation --> Key[rsa, intel-key, ab...] is-trusted-for-attestation
2. Key[rsa, intel-key, ab...] is-trusted-for-attestation AND Key[rsa, intel-key, ab...] says Key[rsa, local-attestation-key, d0...] is-trusted-for-attestation --> Key[rsa, local-attestation-key, d0...] is-trusted-for-attestation
3. Key[rsa, local-attestation-key, d0...] is-trusted-for-attestation AND Key[rsa, local-attestation-key, d0...] says Key[rsa, enclave-key, da...] speaks-for Measurement[...] --> Key[rsa, enclave-key, ...] speaks-for Measurement[...]
4. Key[rsa, policy-key, ...] is-trusted AND Key[rsa, policy-key, ...] says Measurement[0001...] is-trusted --> Measurement[0001...] is-trusted
5. Measurement[0001...] is-trusted AND Key[rsa, enclave-key, bd...] speaks-for Measurement[...] --> Key[rsa, enclave-key, da...] is-trusted-for-authentication

We have built several applications with and without this framework and found it to be extremely useful. It vastly reduces the barrier to building and deploying Confidential Computing programs.