Using the Certifier Framework for Confidential Computing

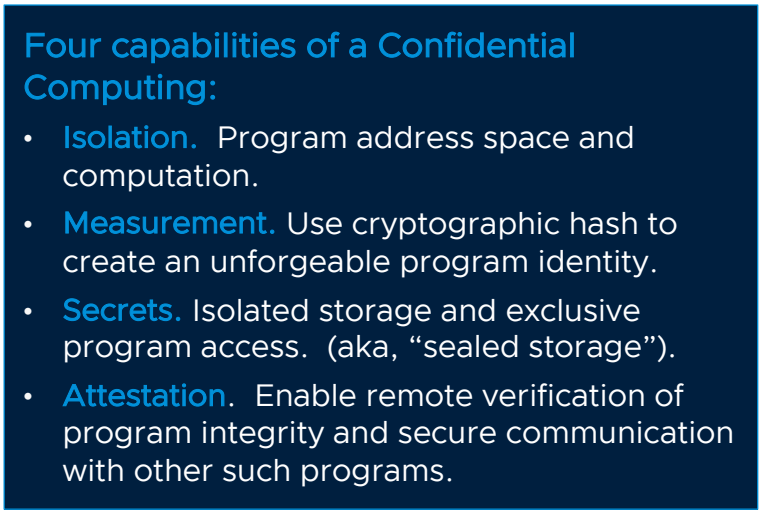
Major Concepts

The Certifier Framework for Confidential Computing consists of two major software elements:

- The Certifier API, which is a small API designed to allow you to use Confidential Computing with a minimum of effort.
- The Certifier Service, which allows Confidential Computing programs to be deployed and managed in a simple, scalable manner.

Confidential Computing relies on isolation of Confidential Computing programs from all other programs, the unforgeable identity of application elements (measurements), secure key management (using Seal/Unseal) and rigorous verification of other Confidential Computing programs under a specific machine enforced policy as a basis for collaboration (Attestation and policy control).

The foundation for Confidential Computing is complete knowledge of each Confidential Computing program in a security domain. Programs can only act in accordance with verified program properties. In addition, trust decisions are rooted in a policy key, in the control of the security domain owner. All program decisions related to what hardware and programs to trust is rooted in the policy key. The policy key signs approved policy and only verified policy signed (or policy key delegated) policy is used in trust or access decisions.



Confidential Computing Properties at a glance

This means that secure hardware and securely written Confidential Computing programs are unconditionally protected. Confidential Computing provides a principled, verifiable security mechanism for distributed computing (across the multi-cloud!); it protects the integrity and confidentiality of processing **wherever** programs run from other malicious programs or even malicious insiders (e.g., admins).

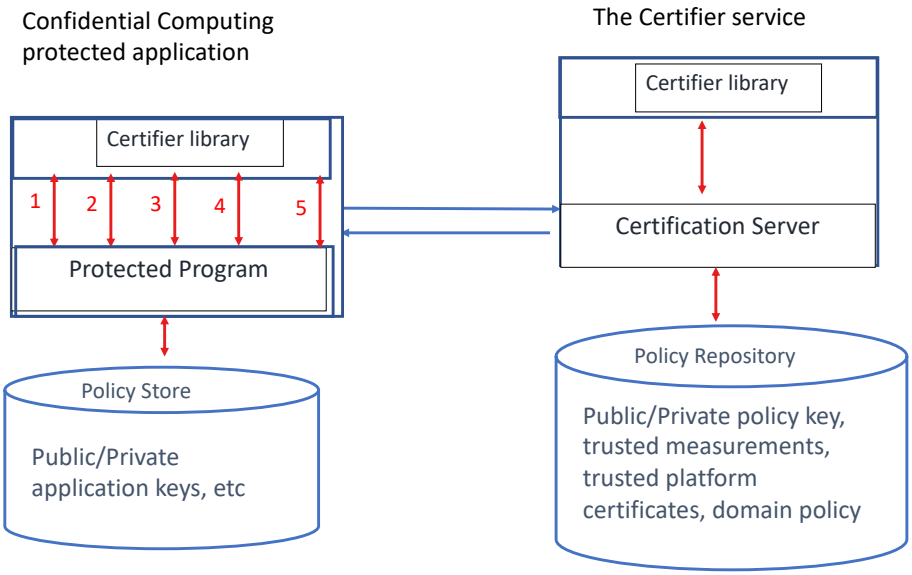


Figure 1: Certifier API and Certifier Service

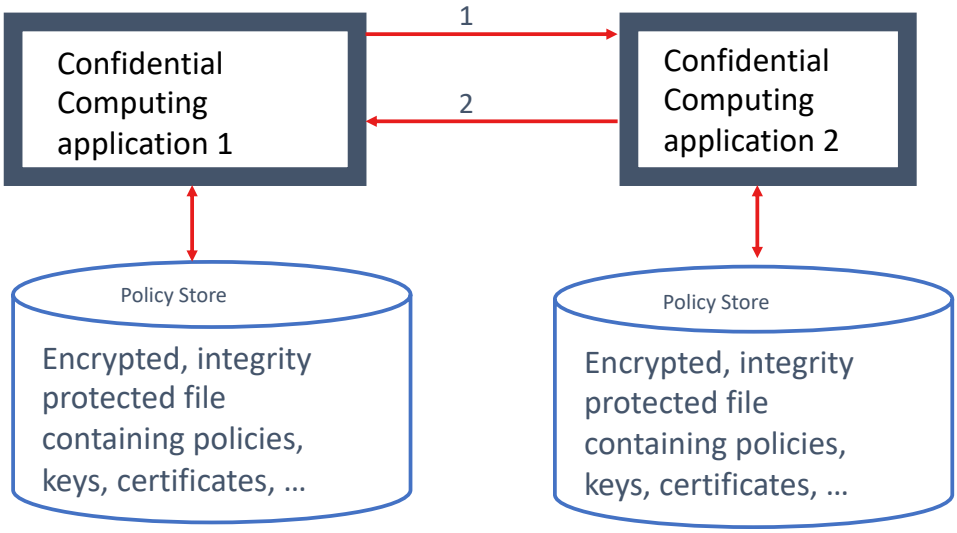


Figure 2: Two mutually authenticated Confidential Computing Applications communicating over a secure channel. The thick box indicates isolation.

Environments

The certifier runs identically in different environments.

Firstly, it can use SGX, SEV and in the future other "hardware enforcement" mechanisms supporting the Isolation, Measurement, Sealing, Unsealing and Attestation without modifying the program. In addition, the certifier comes with a "simulated-enclave" so you can develop and test on platforms without special hardware. Many of the sample applications us it to illustrate the API.

On "enforcing hardware," a client can run in the following ways:

1. In an SGX enclave under Open Enclaves, Asylo, or Gramine. Here the application enclave is the isolated a measured application enclave.
2. In a "application service enclave" in an encrypted virtual machine under SEV or TDX. Here the entire kernel and initramfs is the isolated and measured security principle. The "application service" provides service for OS-wide actions.
3. In an application (process) in an encrypted virtual machine under SEV or TDX. Here, the process is the isolated and measured principal. An OS-wide "application service" provides Isolation, Measurement, Sealing, Unsealing and Attestation services for client application. The application service itself is protected by the secure encrypted virtual machine platform and provides Confidential Computing services (using its own protected keys) for client programs isolated by OS level process isolation. All such protected programs are descendants (e.g-children) of the "application service" and enjoy the same certifier API and certification as programs protected at the platform level. In this case, the application service enforces trust policy using attestation and other services performed by the hardware and the process level attestation performed by an application program enforces trust policy securely provided by the application_service. Notes on using the application service and writing applications that run under it are in the application_service directory.

The certifier API and Certifier Service works the same way in each of these cases. We refer to any of these Confidential Computing protected programs as "applications" below without distinction.

Writing Applications

The certifier API makes converting a well written application into a Confidential Computing enabled application easy.

The certifier performs a number of functions:

1. It abstracts the underlying isolate, measure, seal, unseal and attest primitives so they have the same interface in any environment.
2. It provides a secure store which can be securely saved and recovered (in one statement!). The store will contain keys, public keys for authentication, policies,

symmetric keys for encrypting and integrity protecting files and certificates and tokens acquired by the program to carry out its functions.

3. It provides a policy language, evidence formats and policy evaluation to help a Confidential Computing application determine when another Confidential Computing application should be trusted according to signed policy. Evidence submitted and evaluated includes attestation reports from the platform(s).
4. It contains a mechanism to establish secure channel (encrypted, integrity protected bi-directional channels with authenticated trusted enclave named by their measurements).
5. It contains "helpers" APIs, for example file encryption, file protection using application file keys (so one needn't decrypt files to transfer them to another Confidential Computing application), policy language manipulation, human readable proofs of trust decisions.
6. Mechanisms to establish trust bilaterally between two Confidential Computing applications and, more usefully a mechanism for a Confidential Computing application to prove its trustworthiness within a security domain (defined by policy) to the Certifier Service which provides a one-stop "admission certificate" to establish trust thereafter with any Confidential Computing application in the security domain. This mechanism allows for scalable applications with the ability to upgrade without redistributing application.
7. Utilities to generate keys and write policy.

The process of modifying a well written application to make it a Confidential Computing application is rather thoroughly illustrated in the "sample_app." In the sample_app, a single executable compiled from `example_app.cc` acts both as a Confidential Computing application client (from the point of view of TLS) and as a server. The Certifier Service runs on one or more servers. The Certifier Service evaluates, or certifies, all the policy for the applications that run in a security domain (`example_app.cc`, in this case). This evaluation results in an "Admission Certificate" within the security domain (a domain complying with policy identified by a "policy key"), applications need know nothing about the policy details; indeed, the applications can run securely in any properly configured security domain even as policy changes.

Here are some comments about the process.

1. A Confidential Computing application must have an associated policy key which is a public/private key pair. If a Confidential Computing application wishes to use the Certifier Service, the Certifier generates the key-pair (i.e.- the policy key) and a self-signed cert for the key. The private portion of the policy key is used only by the Certifier Service.
2. If a Confidential Computing application wishes to use the Certifier Service, the policy key, which roots all decisions must be embedded in the applications. As described, in `sample_app/policy_key_notes.txt`, there are three ways to do this, but the easiest involves providing the Certifier Service provisioned self-signed policy key certificate to a utility (`embed_policy_key.exe`) which puts it in the

example_app.exe application. In this case, the policy key is part of the measurement of the Confidential Computing application.

3. When a Confidential Computing application starts for the first time it generates an authentication key, called an enclave-key, which allows it to authenticate itself to other programs in the security domain and puts the private and public portion of the enclave-key in the policy store. This is done in cold_init_trust_info() in example_app.cc. That routine also generates some (optional) symmetric keys that can be used to encrypt, and integrity protect files; those keys are also stored in the policy store.
4. Next, example_app.cc, requests an attestation, naming the enclave public key and recovers some additional evidence supporting a trust decision from the platform. It assembles these into a trust_request_message and sends it to the Certifier Service. The Certifier Service evaluates the evidence in conjunction with security domain policy and sends back a trust_response_message indicating the evaluation was successful and including an "admission certificate" which names the Confidential Computing application measurement and its public enclave-key. This is done in the routine certify_me in example_app.cc.
5. The Confidential Computing application extracts the admission certificate from the response and stores it in the policy store. It saves the policy store so all the information in it can be retrieved whenever the program restarts. The policy store is automatically encrypted, and integrity protected with keys that are sealed to example_app.exe on the platform.
6. At this point, example_app.exe can either continue or restart later. If it decides to restart later, it recovers its policy store (which is decrypted, and integrity checked by the certifier API) and retrieves its enclave public/private key pair as well as its admission certificate. This is done in warm_restart.
7. At this point, the Confidential Computing application example_app.exe is fully initialized and proceeds with normal processing. If it wishes to contact another Confidential Computing application in the security domain (in this case, an instance of the very same example_app.exe on another machine), it uses its enclave key and admission certificate to open a bidirectional channel with the other Confidential Computing application (which symmetricly uses its enclave key and admission certificate in the channel negotiation). The client side of this is performed by one of the participants using client_auth_client(SSL* ssl) and the server side is performed by the other participant using server_application(SSL* ssl). Very likely, you can copy and use all these routines in your program with no changes.
8. During execution, a Confidential Computing application may transmit secrets or data to another trusted Confidential Computing application within the security domain using this secure channel. When it does so, it knows that only the identified (by its measurement) authenticated program complying with the domain security policy can get it.
9. The Confidential Computing application, during execution, may also obtain keys to shared distributed files or wish to securely write, or read previously securely written

files and the certifier provides a one-step way to do that as well as other common functions.

That's it! You can now imagine converting any program or service into on protected by Confidential Computing rather quickly with the certifier API.

The code in sample_app and the instructions there provide a complete step by step guide to writing Confidential Computing Applications and deploying them. There are a few other useful applications provided: a sample machine learning enclave that analyzes data and a keystore that provides keys and tokens for applications in the security domain.

Configuring Policy and running the Certifier Service

Policy is expressed in a declarative policy language rooted in a policy key. The Confidential Computing Framework provides tools to author, read, and distribute policy. Confidential Computing programs that other Confidential Computing programs wish to rely on (trust) must first submit evidence (including attestations) to establish their trustworthiness as well as have an unforgeable way to authenticate themselves. This trust decision is made in a mathematically rigorous way using only the evidence, policy, and logic.

This process is described in detail in the sample application provided with this repository in `sample_app/instructions.txt`.

Proofs from the Certifier Service

Trust decisions are accompanied by short, human readable proof. We have an internal evidence and policy format based on the Lampson-Abadi SPKI/SDSI formalism with constrained delegation. The internal format consists of simple predicates with key or measurement-based principals; these statements are called "`vse-clauses`." You can also use other claim formats (like certs) or substitute another policy evaluation engine, like Datalog or OPA as the indicated in the code.

The internal evidence format has the advantage that it is simple and easily read (by humans!). We haven't come across a policy we can't express rapidly in this format. Policy is produced by the policy tools in the utilities directory. Consult `sample_app/intructions.txt` for further information.

Here is an example "proof" that uses policy and application provided evidence (including an attestation). Remember, the goal is to prove the enclave-key can be trusted for authentication within the security domain based on policy and evidence.

Proof

```
1. Key[rsa, policyKey, c9d16649…] is-trusted
   and
   Key[rsa, policyKey, c9d1664…] says Measurement[cdf3590…]is-trusted
   imply via rule 3
   Measurement[cdf35…] is-trusted

2. Key[rsa, policyKey, c9d16649…] is-trusted
   and
   Key[rsa, policyKey, c9d16649…2] says Key[rsa, platformKey, e59709…]
   is-trusted-for-attestation
   imply via rule 5
   Key[rsa, platformKey, e59709bae…] is-trusted-for-attestation

3. Key[rsa, platformKey, e59709bae…] is-trusted-for-attestation
   and
   Key[rsa, platformKey, e59709bae4…] says Key[rsa, attestKey,
   e3f0bbd20a…] is-trusted-for-attestation
   imply via rule 5
   Key[rsa, attestKey, e3f0bbd2…] is-trusted-for-attestation

4. Key[rsa, attestKey, e3f0bbd2…] is-trusted-for-attestation
   and
   Key[rsa, attestKey, e3f0bbd2…] says Key[rsa, app-auth-key,
    b86447b…] speaks-for Measurement[cdf359…1]
   imply via rule 6
   Key[rsa, app-auth-key, b86447b71e…] speaks-for
   Measurement[cdf359089b4…]

5. Measurement[cdf3590…1] is-trusted
   and
   Key[rsa, app-auth-key, b86447b71e…] speaks-for
   Measurement[cdf35908…]
   imply via rule 1
   Key[rsa, app-auth-key, b86447b7…] is-trusted-for-authentication
```

The conclusion of step 5 (in bold) is what we were after.

Notes and observations

1. The Certificate Service and Certifier API are format rule agnostic. Any tokens or formats you use in an application or service work the same way they used to. No need for token translation or a change in application authorization logic.
2. Provisioning of keys and data requires almost no change to existing applications. Basic keys are either generated by the application or transmitted via a secure channel from a

trusted application in the security domain. Data is provisioned as before, except through a secure channel.

3. The certifier does not rely on root key store, application actions are entirely controlled by signed policy from the policy key.
4. Neither the Certifier nor the Certifier Service requires any changes in application provisioning or deployment. Any existing mechanism continues to work.
5. Confidential Computing applications can be written in C, C++ or Go and via shims all the other popular languages.
6. The Certifier Service can add or upgrade individual Confidential Computing applications without redeploying exiting ones.
7. The entity controlling the Certifier Service is in complete control of the security domain. No action can be taken, no data can be changed, modified, or read unless it conforms to policy. You can run the Certifier Service yourself (with minimal overhead and resilience and availability) consuming minor server resource or you can have someone run it on your behalf.
8. Admission to the security domain relies on a trust decision (usually supported by code inspection) of applications "admitted" to the security domain. Confidentiality and integrity of processing depends only on the Confidential Computing applications (which you either wrote or had an opportunity to review in its entirety or had a third party do so) and hardware enforcement. There is no dependency on third parties or service providers for these properties. Neither improper configuration within a service provider (or on your own machines!), nor malicious administrators, nor malware can compromise your Confidential Computing applications.
9. You can use this framework for collaborative Confidential Computing workloads without disclosing data to other participants.
10. When programming a Confidential Computing program in an encrypted virtual machine, ordinary Linux service calls work in a manner that programmers are familiar with so no additional training is required for programmers who know how to write secure applications. When programming in an SGX enclave, platform calls are provided by an SDK like Open Enclaves or Gramine.

Some Applications

Here are some applications, several of which we have implemented to make sure the Certifier Framework for Confidential Computing is easy (and safe) to use:

1. Hardware secure module
2. Secure key store and token generation
3. Secure motion planning as a service
4. Secure collaborative machine learning
5. Secure auctions
6. Secure real-time trading services

7. Secure Kubernetes container management (via secure Spiffie/Spire)
8. Secure federated identity management
9. Secure databases
10. gRpc
11. Secure document sharing
12. Secure sensor collection
13. Secure caching services
14. Standard platform components (storage, logging, time, IAM)

Advice

We strongly recommend following the instructions and reviewing the code in `sample_app` which gives a complete picture of all aspects of using the certifier API and certifier service.

Using the Certifier Framework for Confidential Computing

Suggestions and contributions are warmly welcomed. The repository is at github.com/vmware-research/certifier-framework-for-confidential-computing

Appendix --- API

The API is specified in two include files, support.h and certifier.h in the include directory. In addition, there is an automatically generated header file for the protobufs and you will find the protobuf definitions in src/certifier.proto. You will likely use only a few of these calls in any application, namely, the Confidential Computing Primitives and the Policy Store and these are all illustrated in the example code. You may also want to use some of the "helper" routines in `sample_app/example_app.cc` (like `cold_init`, `warm_init`, and `certify_me`); these should help you get you apps working quickly. You may also want to use the procedures in these samples to run prototype certifier_service instances. Have fun!