The Runtime Encryption® Company

Whitepaper

Fortanix Runtime Encryption® Platform

1. Introduction

Organizations are worried about protecting their data and applications in the cloud. They use a number of security products and best practices to keep malware out. However, data breaches still happen, and security threats from malicious insiders, zero day bugs, and misconfiguration of policies still persist.

For example, a typical 3-tier web application deployed in an organization may look like Figure 1. The application consists of a web frontend and load balancer such as Nginx, application logic implemented in a framework such as Python Flask, and a database such as MySQL. This application runs as a set of containers, and processes PII data such as user information and credit cards. The organization uses mutual TLS to secure communication between containers and issues certificates to the containers.

This application looks secure on the face of it, but there are multiple vulnerabilities as described in Threat Model Blog that can be exploited by a malicious actor. As PII data moves through the different tiers of the application, a bad actor may be able to steal the data while it is at rest (from unencrypted storage, or encrypted storage with unprotected encryption key), in motion (by stealing and manipulating TLS keys and certificates), or in use (through memory scraping).

Runtime Encryption® technology secures the data used by the application by encrypting it at all times. The encryption keys are secured inside Intel® SGX secure enclaves. The only time the data gets decrypted is when it is being processed by the CPU inside the secure enclave. The Runtime Encryption environment for the application protects it all from all external threats, including attacks by root users, compromised network, advanced malware, memory scraping, operating system zero-day bugs, rogue hardware devices, and code-injection based attacks.

As shown in Figure 2, the 3-tier application can be protected using Fortanix Runtime Encryption without changing the application binaries or the user experience. Runtime Encryption environment introduces two components in an enterprise – **EnclaveOS™**, and **Enclave Manager™**. EnclaveOS is a runtime environment for applications to allow them to run inside SGX enclaves. Enclave Manager is a management software which provides visibility into the running enclaves and establishes trust between enclaves using the remote attestation capabilities of Intel® SGX.

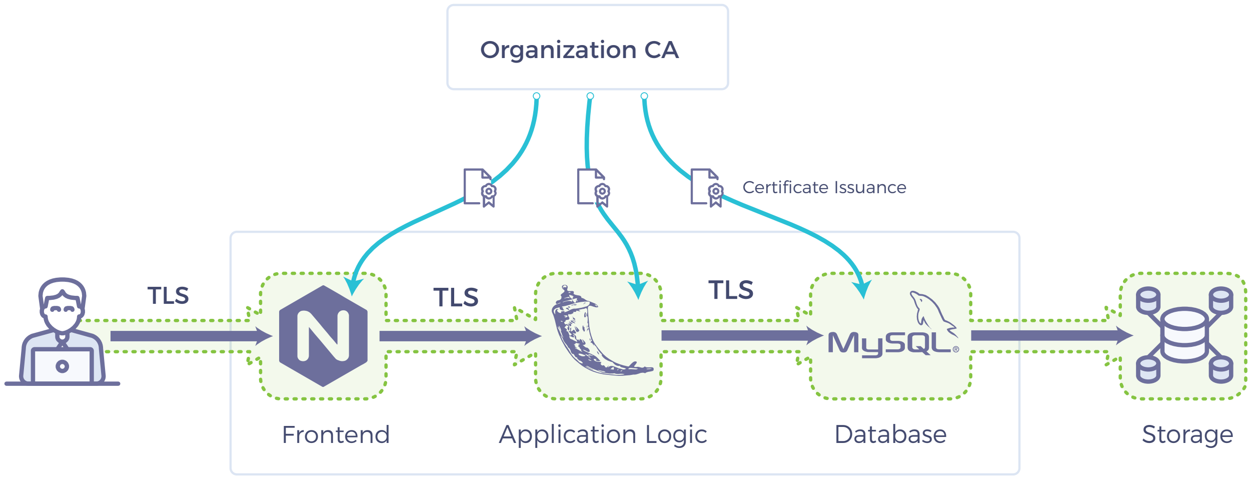

Figure 1: A typical 3-tier enterprise web application

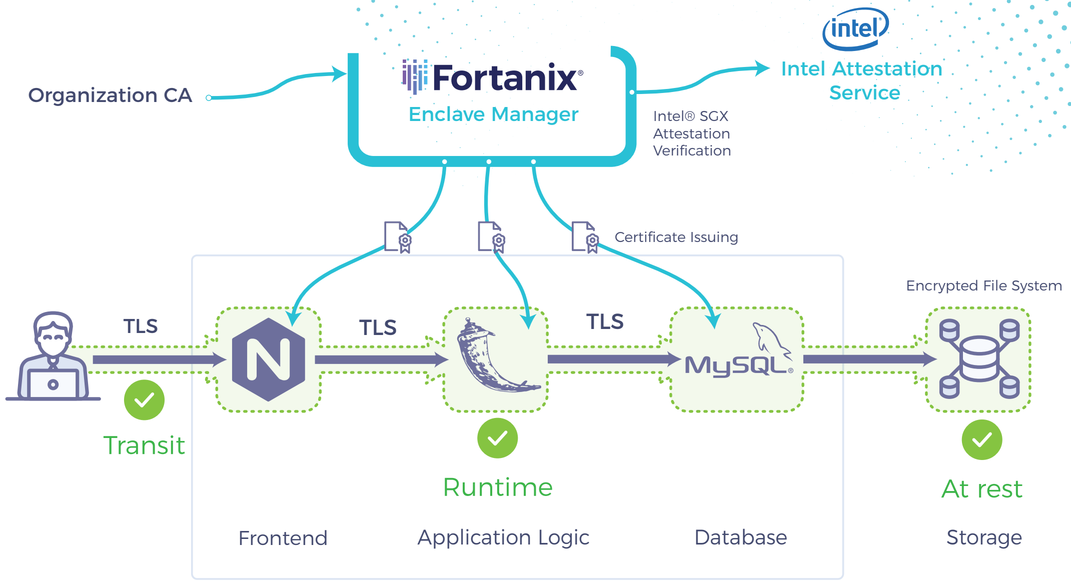

Figure 2: Enterprise 3-tier application secured with Fortanix Runtime Encryption

2. Enclave OS

Fortanix provides a runtime environment for applications, which implements some of the functionality traditionally provided by OS kernel in user space, thus enabling the applications to run unmodified in a secure execution environment. The secure execution environment uses encryption and hardware-enforced security isolation to make applications in this environment completely immune to a wide-range of threats originating in traditional host software including root users, network intruders, malicious insiders, code-injection, cold-boot attacks, and OS zero-day bugs. This is done by establishing a root of trust in the CPU itself and using that to encrypt all the system memory and all other sensitive IO accesses.

2.1 EnclaveOS Architecture

EnclaveOS is the Runtime Encryption environment that goes with every application as shown in Figure 3. The application does not need to be modified.

The application, the required language runtime (e.g., JVM), and EnclaveOS are bundled together.

EnclaveOS uses Intel® SGX to create a secure execution environment. A region of memory is created that is inaccessible to any process, regardless of its privilege level, other than the application itself. All the code and data used by the application is stored in this protected region of memory encrypted with a key. This key is never stored in any persistent storage and is not even present in the RAM. The CPU derives the key on the fly based on a secret already provisioned in the CPU. As the CPU derives the key and performs encryption and decryption of memory without using any support from any of the software (operating system, SMM, BIOS, hypervisor, etc.), the application remains protected even from these higher-privileged processes.

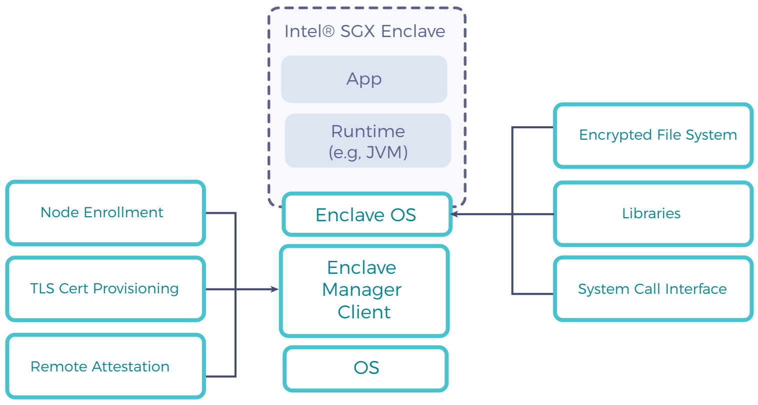

Figure 3: Application running on EnclaveOS

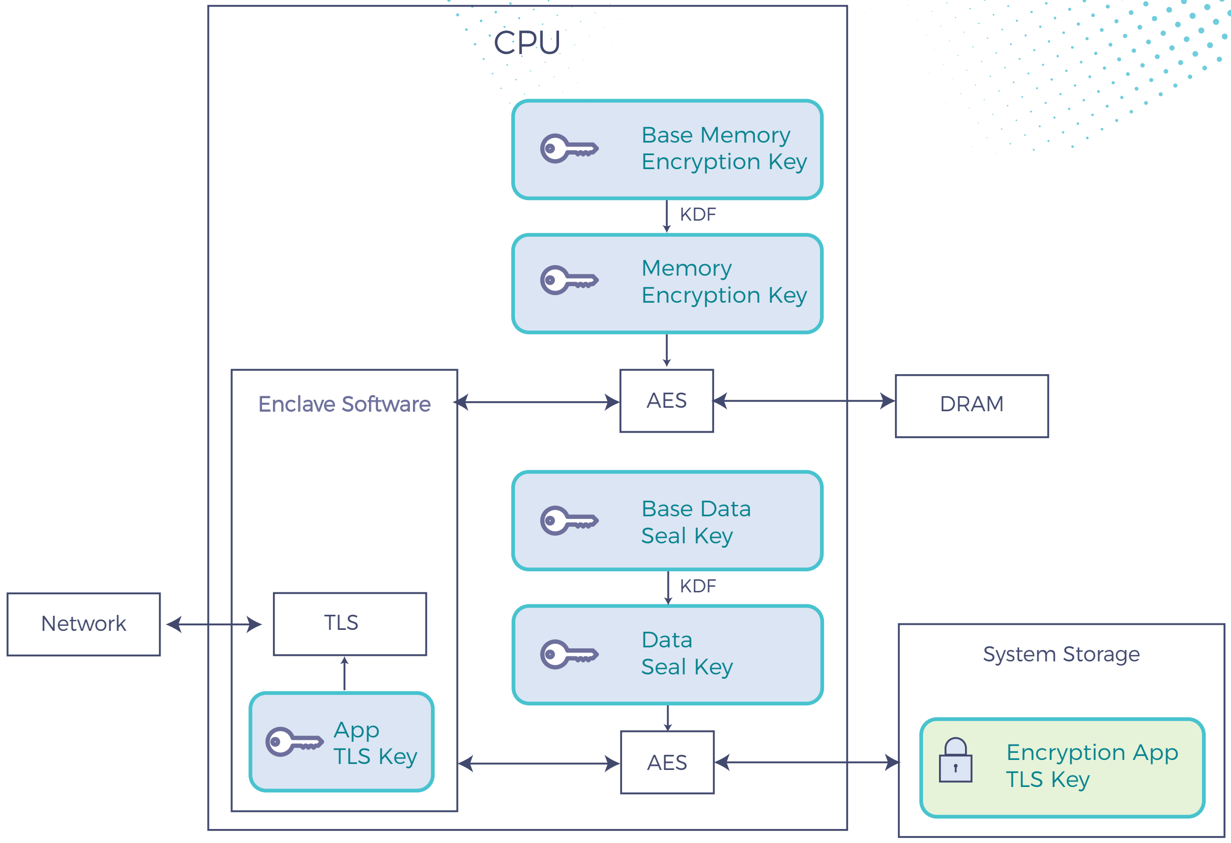

Figure 4: Key hierarchy for storage and network encryption

EnclaveOS encrypt all data which may get written out to the memory (DRAM), to system storage, and to the network.

Figure 4 shows the primary keys involved in these encryption operations. The following keys are used:

For encrypting data written to DRAM:

A memory encryption key is randomly generated by the processor on every boot and is used by the processor to encrypt and decrypt Intel® SGX enclave memory. The memory encryption key is usable only by the hardware. Enclave software does not have the ability to directly request encryption or decryption using this key.

For encrypting data written to system storage:

A data seal key is used for this. A base seal key is fused into each processor during manufacturing of Intel® SGX-capable CPU, and is unknown even to Intel®. At runtime, enclave-specific keys are derived from this base key using a key derivation function (KDF). The input to the KDF include the identity of the application and the identity of the signer. The identity of the application is obtained from the cryptographic hash of the enclave binary, and the identity of the signer is obtained from the cryptographic hash of the public portion of the key used to sign the enclave.

For encrypting data written to network:

The application uses one or more keys to authenticate itself to network clients, services, or other external entities. Commonly, these are TLS keys and they are generated by the application. A TLS key for the application running on EnclaveOS is present in plaintext only within the secure enclave. At rest, the TLS key is encrypted with one of the application's data sealing keys prior to being stored in system storage.

2.2 System Call Interface

EnclaveOS provides a Linux-compatible system call interface to support running any application designed to run on Linux. Figure 5 shows an example of how data may be present in the CPU core, CPU cache, and RAM during the lifecycle of a single system call. When the application makes a call to the write() function, the call is intercepted by EnclaveOS, which fetches the pages required for the `write()` call from enclave pages in the CPU cache. Note that only an enclave process with the CPU running in the enclave context can access these pages. At this time, the requested pages may be fetched from the memory or the swap, in which case, the memory pages will be decrypted before they are brought to the cache. The data is then encrypted inside the EnclaveOS, which then writes the encrypted data to the user (non-enclave) pages in the CPU cache. Again, this data may be sent out to the RAM or swap space. The EnclaveOS then makes an outward call (OCall) to the user (non-enclave) portion of the EnclaveOS , which gets the pages from the CPU cache, and calls the regular Linux `write()` system call.

2.3 Libraries

EnclaveOS provides modified versions of standard libraries (libc, libm, libpthread, librt, etc.) that invoke EnclaveOS functions rather than making system calls to the Linux kernel. These libraries are automatically substituted when launching dynamically linked applications. Other system call interception strategies are also supported.

2.4 Encrypted File System

EnclaveOS ensures that the application data is secured not just in the memory but also when it is written on the disk. It mounts an encrypted file system with each application such that application can transparently encrypt all data that gets written out to the disk. The encryption key is associated with the application and is protected from all other applications, including the host operating system. This key is derived from the data seal key described in Section 2.1.

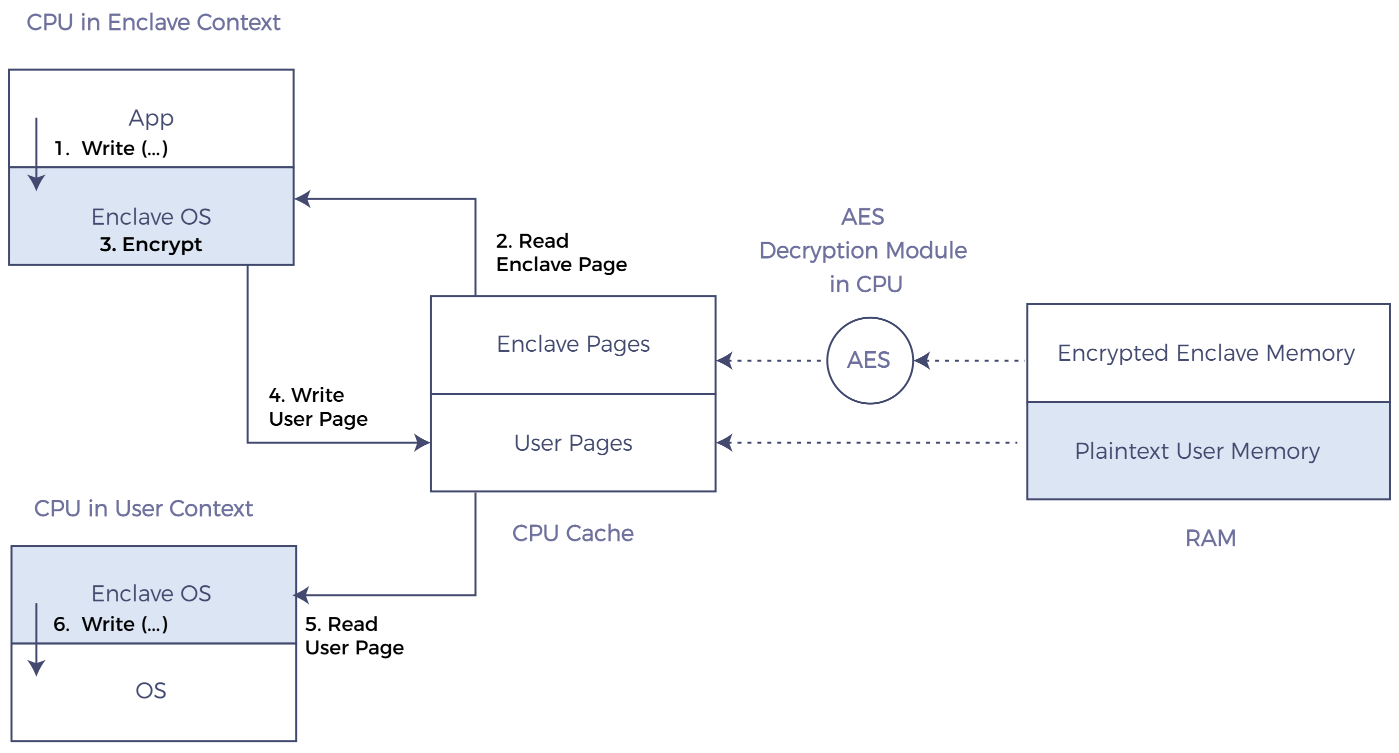

Figure 5: Transparent encryption and integrity protection of memory

3. Trust model for communication between applications

This section describes how communication between applications running in EnclaveOS is secured. It starts with a brief review of the standard Linux container security model. Figure 6 shows a sample container environment (e.g., Docker or CoreOS). Each container may contain one or more processes. Processes within a container can communicate using the usual Linux IPC mechanisms (pipes, Unix domain sockets, shared memory). Processes in different containers have more limited communication options. They can communicate via IP sockets. It is also possible to bind-mount filesystems from the host into the container. If the same filesystem is mounted into multiple containers, they can communicate via files.

In the EnclaveOS environment, a container corresponds to an application manifest and a filesystem image. At launch, EnclaveOS starts a single process as defined by the container's entry point definition. That process may in turn launch other processes. Processes launched in the context of a single EnclaveOS application manifest (i.e. a single container) are assumed to cooperatively implement the functionality of a single application. The EnclaveOS platform does not cryptographically isolate different processes belonging to a single application.

The identity of an EnclaveOS application is determined by the hash of the application image. The manifest is part of the application image. By placing the initial filesystem hash in the manifest, the filesystem is also part of the application identity. Processes launched from the same application manifest share the same identity, and are thus able to derive the same keys. Communication between processes in the same EnclaveOS application (i.e. in the same container) is secured using these shared keys.

Processes launched by different EnclaveOS applications (i.e. in different containers) are assumed by the EnclaveOS platform to be mutually distrusting. The applications have different identities, and thus one application is not able to derive the other's keys. Typically, communication between two different applications is via a mechanism like TLS. This is facilitated in Fortanix Runtime Encryption environment by Enclave Manager, which is described in the next section.

To enable IPC between different applications, the trust relationships must be manually specified. For example, the manifest for application/container A can specify that it wants to share a filesystem with application B, and that it wants to create shared memory with application C. If the manifests for applications B and C also specify that they want to share these resources with application A, the applications can establish secure channels using DH key exchange authenticated by the application identity at runtime. The encryption keys for filesystems and shared memory are exchanged over these channels. The EnclaveOS platform does not currently provide revocation for these trust arrangements. Since shared memory is inherently volatile, it is enough to release new versions of the applications without the trust specifications. When the applications are restarted, the new versions will no longer allow exchange of shared memory encryption keys.

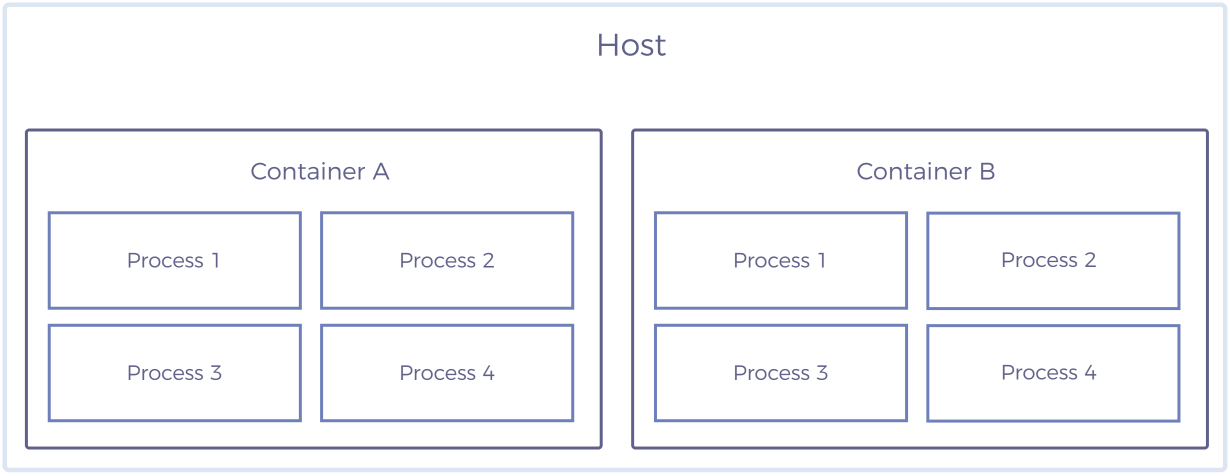

Figure 6: Containers in Linux

4. Enclave Manager

Fortanix Enclave Manager is a management and orchestration software which is useful for provisioning SGX capable machines and SGX applications, and for managing trust relationships between them.

4.1 Enclave Manager Architecture

Figure 7 shows the trust relationships involved in provisioning keys to the application. The purpose of the provisioning flow is to ensure that application keys are provisioned only to genuine instances of the application running in the Intel® SGX environment. Figure 7 illustrates the relationships between the following entities:

The Application Node, which is the compute node running the application.

An Attestation Service. The Enclave Manager queries the attestation service to verify attestations produced by application nodes, in order to confirm that the nodes are genuine. This is typically the Intel® Attestation Service.

A Certificate Authority, which issues TLS certs used to authenticate applications to other services in the deployed environment. This is typically a component of the Enclave Manager, but can also be an organization's CA.

The Fortanix Quoting and Provisioning Enclave (FQPE), which is a Fortanix-provided enclave service that runs on each application node. The FQPE manages a node private key, and has a role in the attestation process.

4.2 Node Enrollment

This is a one-time action for every Application Node. Before any Application Node runs, it must be enrolled into Enclave Manager. The Application Node sends an attestation to Enclave Manager, which verifies the attestation with the Intel® Attestation Service to confirm that the Application Node is a genuine Intel® SGX machine. Once this has been confirmed, the Enclave Manager can provision a secret with FQPE such that the future requests for attesting EnclaveOS applications can be satisfied by the Enclave Manager, and there is no further need to contact Intel® for future attestations.

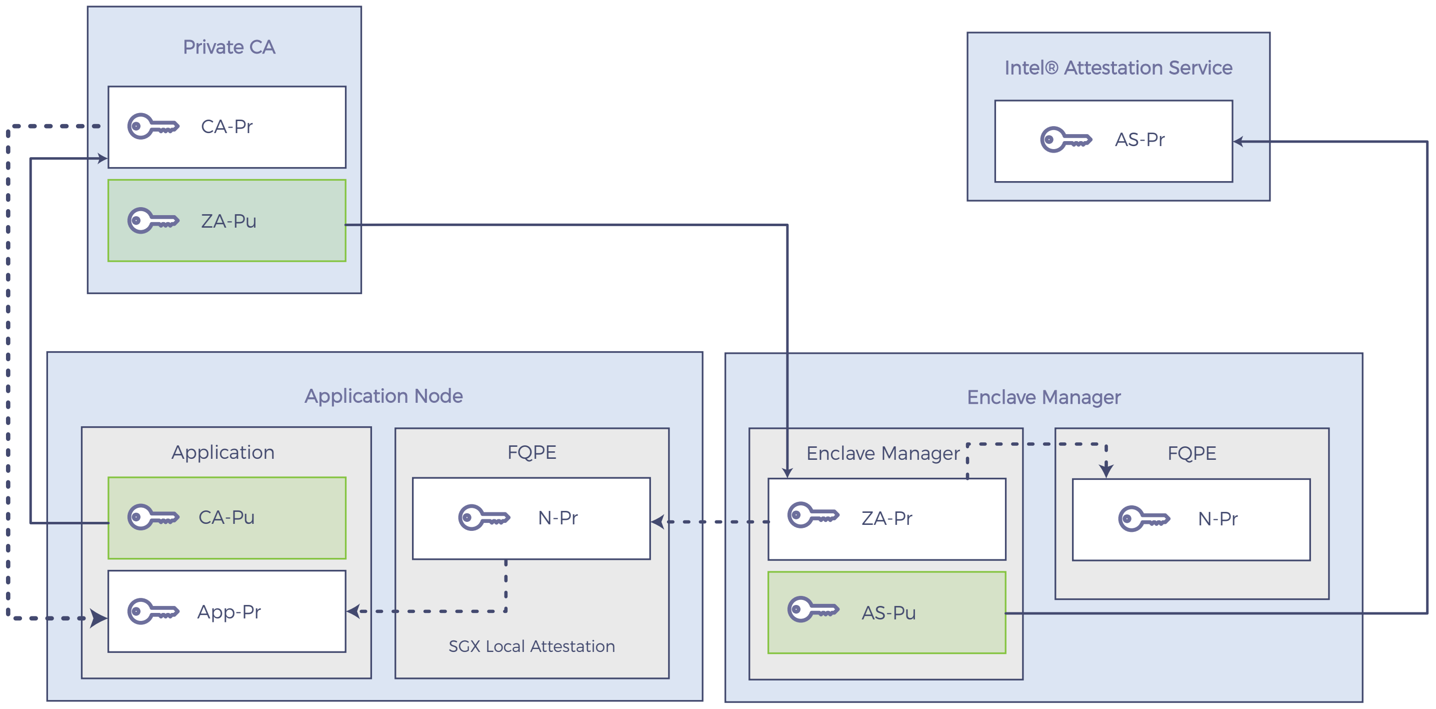

Figure 7: Trust Relationships in Enclave Manager

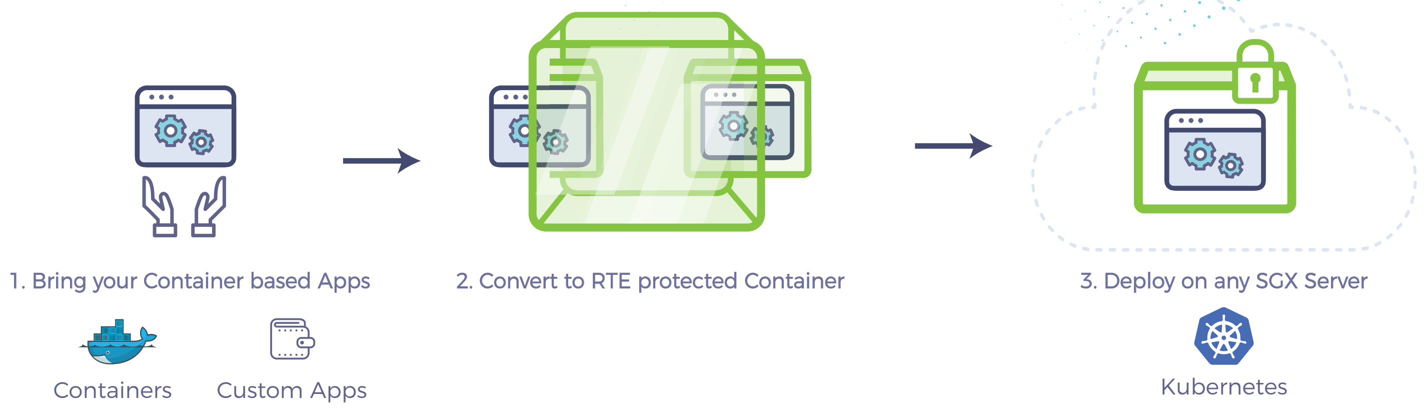
1. Bring your Container based Apps
Containers
Custom Apps
2. Convert to RTE protected Container
3. Deploy on any SGX Server
Kubernetes

Figure 8: One-click integration of container images to run on EnclaveOS™

4.3 Application Whitelisting

Enclave Manager can whitelist EnclaveOS applications. The enclave related properties of the application are included for the application while whitelisting. This includes the identity or hash of the enclave (MRENCLAVE), identity of the signer of the enclave (MRSIGNER), product identifier (ISVPRODID), security version number (ISVSVN). When the enclave runs and presents its attestation to the Enclave Manager, all of these values are included in the attestation report, which can be used by the Enclave Manger to determine whether to accept the attestation.

4.4 TLS certificate provisioning

The Certificate Authority issues TLS certs to EnclaveOS applications on verifying their remote attestation signed by the FQPE. The attestation step can be added either manually or automatically in an existing private CA, or using Enclave Manager to issue TLS certificates. Applications can use this TLS certificate as a server certificate (e.g., web servers, databases, etc.), or a client certificate, or both.

Securing applications using Fortanix Runtime Encryption Platform:

Fortanix Runtime Encryption Platform is currently available as IBM Cloud Data Shield, a service running on IBM Cloud. This service runs on top of IBM Kubernetes Service (IKS), and thus supports running applications as containerized services.

On installation, IBM Cloud Data Shield instantiates the Enclave Manager. As an immediate first step, the worker nodes which form the Kubernetes cluster in IKS are enrolled into the Enclave Manager. These nodes, along with their SGX attestation information, can be viewed in the web console of Enclave Manager.

To run an application on IBM Cloud Data Shield, a container conversion tool is provided to convert an existing Docker container image into an SGX capable container image. This inserts EnclaveOS into the Docker container image and sets the required parameters to run the original application on EnclaveOS. The modified container image can now be whitelisted with Enclave Manager. After whitelisting, the container can be launched using standard Kubernetes interface. Enclave Manager can verify attestation and issue TLS certificates that can be used by the application.

For more details, please visit
https://www.ibm.com/cloud/data-shield

or send us an email at
support@fortanix.com

or join our Runtime Encryption® forum on Slack at
https://fortanix.com/runtime-encryption-slack

Connect with us

LinkedIn Twitter Facebook Slack