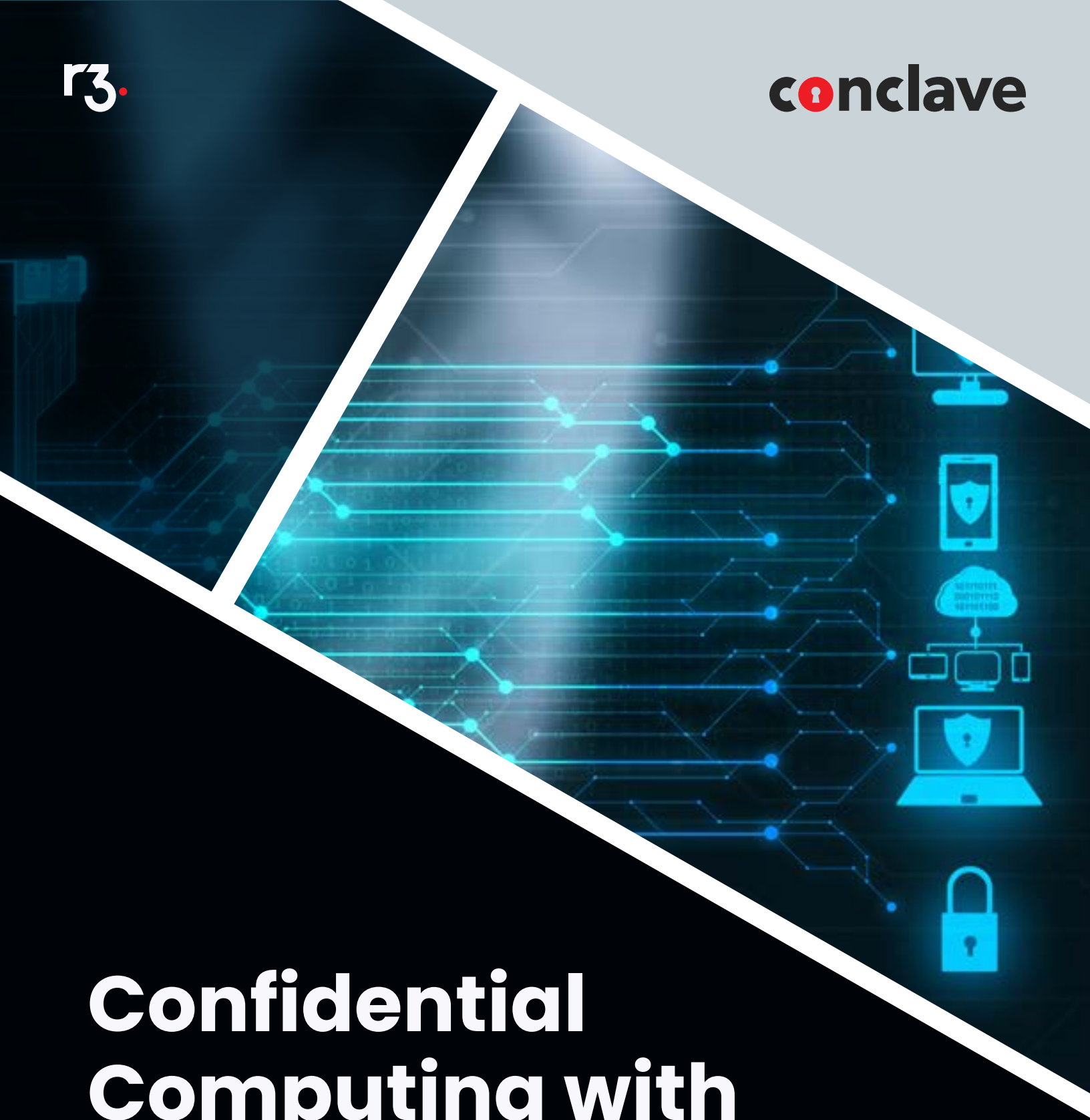
r3.

conclave



Confidential Computing with Conclave

R3's Developer Relations • Ashutosh Meher • August 2021

Index

Data is a valuable asset in the modern world. It has been leveraged to invent and improve technologies that have immensely benefited humanity. Anything this valuable must be protected from misuse. As such, data protection is an utmost priority for all businesses.

Background

When we think of data it can generally exist in three different states. There are different methods we can use to protect the data in each of these states:

Data at rest

This refers to data stored at a physical location in a file system. We use technologies like file encryption or hardware encryption to protect data at rest from unauthorized access.

Data in transit

This refers to data being transferred from one location to another via a network. We protect data in transit using encryption technologies like TLS (Transport Layer Protection).

Data in use

This refers to data being used, that is the data being processed to provide a result. Data itself is not of much use unless it's processed to compute a result. While this is an important step in the life cycle of data, data protection in this step has been majorly ignored over the years.

To understand this further, let's take an example of a tender processing use case. Suppose an organization issues an RFP (Request for Proposal) to invite bids for a particular project. While different interested parties submit their bids/proposals in a confidential manner, they are still at the mercy of the person/organization handling the entire process to keep their bids confidential. It is possible that someone having access to this confidential information might leak it for their own benefit.

Confidential Computing is an emerging technology which focuses on data protection while it is in use. It is a hardware-based approach which requires special hardware or CPU to protect data while it is being processed.

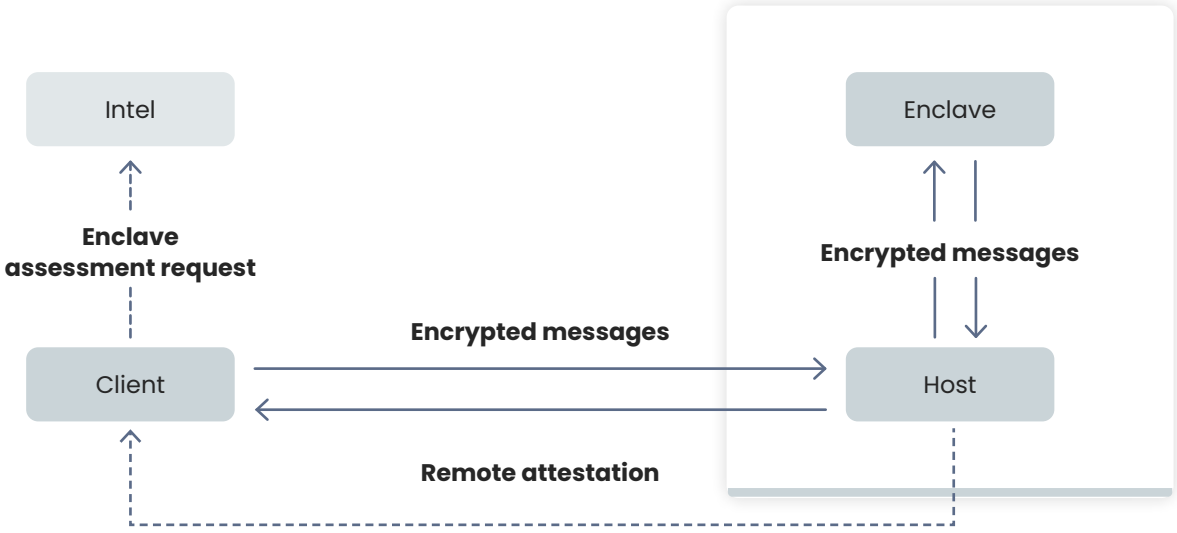
Intel® SGX

Intel® SGX is a hardware technology that protects data in use using secure enclaves or Trusted Execution Environments (TEEs). An enclave is a small piece of software that runs in an isolated region in the memory. Access to this region of memory is blocked to everyone, even privileged software like kernel and BIOS. Thus, code and data on the enclave can't be read/ tempered by anyone, not even by the owner of the computer in which it runs.

Conclave

Conclave is an application development toolkit that eases the development of enclave programs. It builds on Intel® SGX and hence running a secure Conclave application requires Intel® SGX supported CPUs. Conclave runs a JVM inside and enclave, it uses GraalVM, thus allowing developers to write enclave programs using high-level languages like Java, Kotlin and JavaScript.

Conclave application architecture



A Conclave application typically has 3 major components and provides a testing framework which helps in unit testing of enclaves:

1. **Enclave**
2. **Host**
3. **Client**

1 Enclave

An enclave is a piece of program which runs in the protected memory space of the Intel® SGX machines and is isolated from the outside world. It processes private data confidentially and provides processed results to the clients. They are loaded onto a dedicate sub-JVM (GraalVM) that does not share the same JVM as the host.

2 Host

Since enclaves are isolated and can't be accessed from the outside world, the host machine runs a host program which does the following:

- Load the enclave programs
- Provide the necessary resources required to the enclave
- Serve as a proxy between the client and the enclave

The clients can't directly access the enclaves, they rely on the host to deliver any message to the enclave. It is important to note that the host is always considered untrusted, and hence all confidential information exchanged between the host and the client, or the host and the enclave must be encrypted.

3 Client

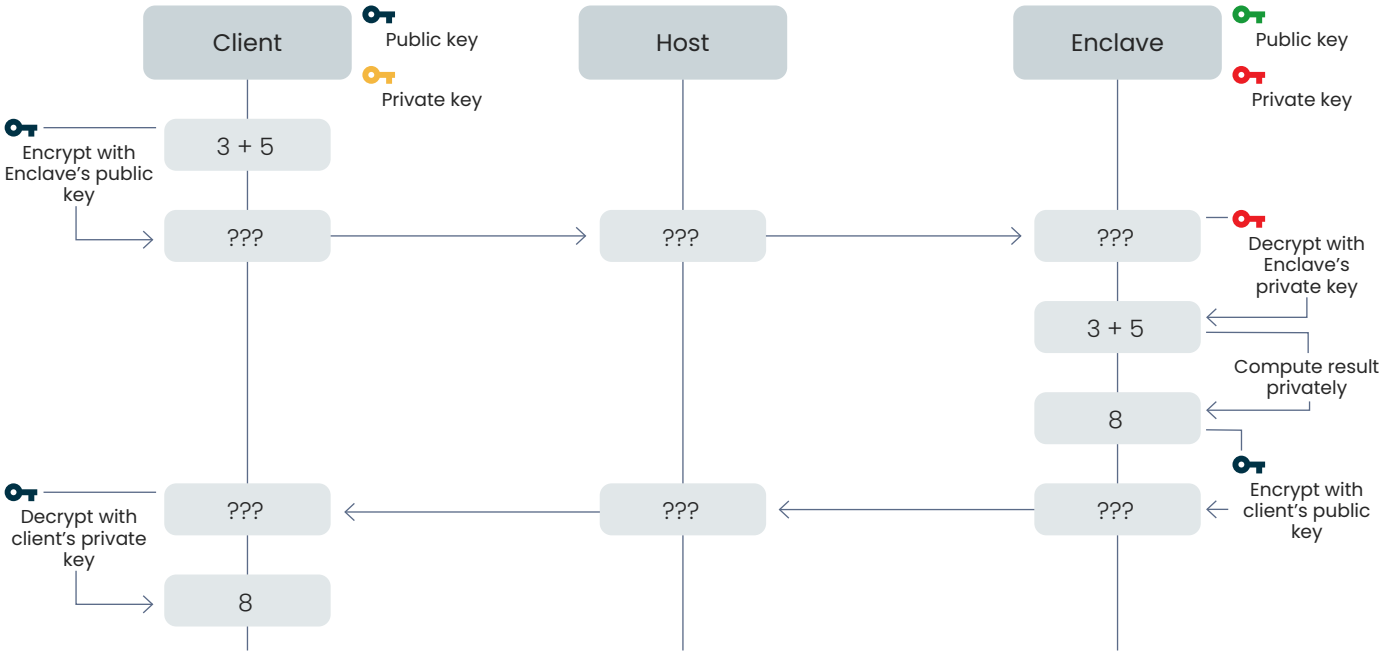
Clients are programs which send confidential data to the enclave for processing via the host. Conclave comes with the Mail API to ease the communication between enclaves and clients.

Conclave doesn't mandate any particular network protocol for client-host communication. It's up to the developer to choose the protocol of their choice.

Encrypted messaging

Encryption techniques are used to avoid the risk of confidential information being compromised in a Conclave-powered application. The enclave and the client use key-pairs to encrypt and decrypt information exchanged between them.

The client encrypts information sent to the enclave using the enclaves public key so it can only be decrypted using the enclave's private key. Thus, the host responsible for relaying the information sent by the client to the enclave can't read/temper with the information. Similarly, the enclave encrypts the processed result using the client's public key hence it can only be decrypted and read by the client.



Remote attestation

Any program could use a random key-pair and pretend to be an enclave. It's important for the client to verify that a piece of software advertising itself as an enclave is actually an enclave, before sending any confidential information to it. To verify the authenticity of the enclave 'remote attestation' is used.

Remote attestation is a piece of data that contains important information that can be used to verify an enclave. Among other information, it contains something called a measurement.

A measurement is a unique hash generated using a special tool, the hash is generated using the fat-jar that's loaded onto the enclave. The measurement can be verified by compiling the enclave source code. Conclave takes care of the fact that the multiple builds of the same source-code produce the same measurement.

In addition to remote attestation, clients can also request Intel® SGX for an assessment of the enclave to verify if it is secure. Intel® SGX provides important information about the enclave hardware like if it's using an updated version or if it has any security vulnerability etc.

It's important to understand that any upgrade to the enclave code will result in a change in measurement. This will result in failure since the client will no longer be able to identify the enclave. A potential solution is to maintain a whitelist of acceptable hashes.

Alternatively, a signing key could be used and as long as the enclave is signed with the key, it could be deemed as authentic.

Conclave mail

The Conclave toolkit comes with the Mail API which facilitates developers with communication between clients and enclaves. In addition, it provides the below useful features:

Encryption
End-to-end encryption of messages between the client and the enclave using AES encryption.

Authentication
Mail allows a message to prove it came from the owner of a particular key, as well as being encrypted to a destination public key.

Headers
Mail provides unencrypted and tamper proof headers, which can be used to link messages together. It uses topics and sequence numbers which can provide protection against reordering and dropped messages.

Framing
Mail delimits messages such that the beginning and ending can be easily identified, without the need of implementing any framing on top of it.

Enclave running modes

Conclave requires specialized hardware (Intel® SGX enabled machine) to run applications securely in production environments. However, to reduce cost and enhance ease of development an enclave can run in various modes, some of which does not required specialized hardware or operating system. An enclave can be run in the below modes:

Release mode

This mode is used for production environments. It provides the highest level of security and requires Intel® SGX enabled machine & Linux operating systems. There is no back door, and the enclaves runs in complete isolation.

Debug mode

This mode is similar to release mode and requires an Intel SGX enabled machine & Linux operating systems. However, it provides a back door to the enclave which allows debugging. It can used for UAT/pre-production environment where a production like environment is required with the ability to debug. It's not completely secure as it provides a back door into the enclave.

Simulation mode

This mode does not require Intel® SGX machines, it runs an enclave using software simulation. However, a Linux operating system is still required to run an enclave in this mode. This mode is insecure as it uses a software simulated enclave instead of a real one. Important to note that the enclave is still loaded into its separate dedicated JVM.

Mock mode

This mode is least secure, it neither requires a specialized hardware or operating system. The enclave runs within the host JVM. This mode can be considered very useful for development purposes.

Conclave hello world application

The Conclave SDK comes with a hello world application to help developers understand the basics of an application built on Conclave.. The hello-world application uses a very simple enclave to reverse any string that is passed to it.

You could download the latest version of Conclave SDK from:
developer.r3.com/conclave

Configuring the project

settings.gradle

Conclave uses the gradle build system, so the three components of the application built on Conclave need to be configured as modules. This is done with the 'settings.gradle' file found at the root of the project.

```
pluginManagement {
    repositories {
        maven {
            def repoPath = file(rootDir.relativePath(file(conclaveRepo)))
            if (repoPath == null)
                throw new Exception("Make sure the 'conclaveRepo' setting exists in gradle.
                properties, or your \$HOME/gradle.properties file. See the Conclave
                tutorial on https://docs.conclave.net")
            else if (!new File(repoPath, "com").isDirectory())
                throw new Exception("The $repoPath directory doesn't seem to exist or isn't
                a Maven repository; it should be the SDK 'repo' subdirectory. See the
                Conclave tutorial on https://docs.conclave.net")
                url = repoPath
              }
              // Add standard repositories back
              gradlePluginPortal()
              jcenter()
              mavenCentral()
          }

          plugins {
              id 'com.r3.conclave.enclave' version conclaveVersion apply false
          }
}

include 'enclave'
include 'host'
include 'client'
```

Apart from configuring the modules, the setting.gradle file also defines the pluginManagement block which is used to configure gradle to find the Conclave SDK in your file system.

gradle.properties

The path to the Conclave SDK and its version is required to be configured in the 'gradle.properties' file as shown below.

```
conclaveRepo=/path/to/sdk/repo
conclaveVersion=1.1
```

Root build.gradle

Import the Conclave SDK repository to the root build.gradle.

```
subprojects {
    repositories {
        maven {
            url = rootProject.file(conclaveRepo)
    }
        mavenCentral()
    }
}
```

Configuring individual modules

Host module

The below line of code in the host's build.gradle, helps developers to choose the enclave mode from the command line. The enclave runs in simulation mode by default.

```
def mode = findProperty("enclaveMode")?.toString()?.toLowerCase() ?: "simulation"
```

The application gradle plugin and the mainClassName allows the host module to be run from the command line.

```
plugins {
    id 'java'
    id 'application'
}

application {
    mainClassName = "com.r3.conclave.helloworld.host.Host"
}
```

Finally, the required dependencies are added.

```
dependencies {
    implementation "com.r3.conclave:conclave-host:$conclaveVersion"
    runtimeOnly project(path: ":enclave", configuration: mode)

    runtimeOnly "org.slf4j:slf4j-simple:1.7.30"
}
```

Enclave module

The Conclave gradle plugin and dependencies are added to the enclave's build.gradle file. The version of Conclave is not required in the dependencies as the plugin will set it up its own.

```
plugins {
    id 'com.r3.conclave.enclave'
}

dependencies {
    implementation "com.r3.conclave:conclave-enclave"
}
```

You could also configure signing keys for different Conclave running modes.
To learn more visit: docs.conclave.net/writing-hello-world.html#signing-keys

Client module

The client module just needs the application plugin with the mainClassName to run it from the command-line, and the dependencies.

```
plugin {
    id 'java'
    id 'application'
}

application {
    mainClassName = "com.r3.conclave.helloworld.client.Client"

dependencies {
    implementation = "com.r3.conclave:conclave-client:$conclaveVersion"
}
```

Develop individual components

Enclave class

The enclave in need to perform a simple task. Receive a string in an encrypted mail from the host, reverse the received string, encrypt it and send it back to the host.

The enclave class for the hello-world application is named as ReverseEnclave. This must be a subclass of Enclave class.

```
public class ReverseEnclave extends Enclave {
}
```

The reverse enclave will receive encrypted mails in the recieveMail method. This method accepts 3 parameters:

- **Id:** Used to acknowledge the receipt of mail (if needed, ignored in this case) by the enclave to the host.
- **mail:** The encrypted mail itself.
- **routingHint:** A simple string which helps the enclave to keep track to different clients when dealing with multiple clients. This is a random string created by the host.

```
public class ReverseEnclave extends Enclave {

// The sample code download with the SDK will have this method as
// receiveFromUntrustedHost. Its changed here for simplicity
    public byte[] reverse(byte[] bytes) {
        byte[] result = new byte[bytes.length];
        for (int i = 0; i < bytes.length; i++) {
                result[i] = bytes[bytes.length - 1 - i];
        }
        return result;
    }

@Override
protected void receiveMail(long id, EnclaveMail mail, String routingHint) {
        byte[] reversed = reverse(mail.getBodyAsBytes());
        byte[] responseBytes = postOffice(mail).encryptMail(reversed);
        postMail(responseBytes, routingHint);
    }
}
```

The reversed bytes are then encrypted using the EnclavePostOffice's (The instance is obtained by the postOffice() method) encrypt() method. Finally the response is posed to the host using the postMail() method.

PostOffice API is a Conclave feature which helps developers to use mails easily.

Host class

The host performs the following tasks:

1. Verify if the platform being run supports the enclave
2. Load the enclave
3. Register callback to receive enclave response
4. Listen to client connection request
5. Send attestation to connected clients
6. Receive encrypted mails from clients and deliver them to the enclave
7. Deliver the enclave response to the client

The host first checks whether the platform its being run on supports enclaves.

```
try {
    EnclaveHost.checkPlatformSupportsEnclaves(true);
    System.out.println("This platform supports enclaves in simulation, debug and release mode.");
} catch (EnclaveLoadException e) {
    System.out.println("This platform does not support hardware enclaves: " + e.getMessage());
}
```

Then, it loads the enclave.

```
EnclaveHost enclave = EnclaveHost.load("com.r3.conclave.sample.enclave.ReverseEnclave");
```

Once loaded, it starts the enclave and registers a callback to receive the response from the enclave. From within the callback the response is sent back to the client.

```
enclave.start(new AttestationParameters.DCAP(), (commands) -> {
    for (MailCommand command : commands) {
        if (command instanceof MailCommand.PostMail) {
            try {
                sendArray(output, ((MailCommand.PostMail) command).getEncryptedBytes());
            } catch (IOException e) {
                System.err.println("Failed to send reply to client.");
                e.printStackTrace();
            }
        }
    }
});
```

```
private static void
sendArray(DataOutputStream stream,
byte[] bytes) throws IOException {
    stream.writeInt(bytes.length);
    stream.write(bytes);
    stream.flush();
}
```

The attestation information can be obtained from the enclave instance using the getEnclaveInstanceInfo() method. The attestation is sent back to the client for verification.

```
final EnclaveInstanceInfo attestation = enclave.
getEnclaveInstanceInfo();
final byte[] attestationBytes = attestation.serialize();

sendArray(output, attestationBytes);
```

Upon successful verification of the constraint the client sends the string it needs to reverse using the Conclave mail, which is forwarded to the enclave using the deliverMail() method.

```
DataInputStream input = new DataInputStream(connection.getInputStream());
byte[] mailBytes = new byte[input.readInt()];
input.readFully(mailBytes);

enclave.deliverMail(1, mailBytes, “routingHint”);
```

Client class

The client needs to perform the following tasks:

1. Connect to the host
2. Receive attestation from the host and verify if the enclave is legit
3. Send a string to the host using encrypted mail
4. Receive response form the host, decrypt, and read the response

Connect to the host using a simple Socket and receive attestation from the host.

```
Socket socket = new Socket("localhost", 9999);

DataInputStream fromHost = new DataInputStream(socket.getInputStream());
byte[] attestationBytes = new byte[fromHost.readInt()];
fromHost.readFully(attestationBytes);

EnclaveInstanceInfo attestation = EnclaveInstanceInfo.deserialize(attestationBytes);
```

To verify the attestation information the below code is used. The below method will throw an exception if the constraint is not acceptable.

```
EnclaveConstraint.parse(<constraint_string>).check(attestation);
```

To learn more about constraint refer:
docs.conclave.net/writing-hello-world.html#constraints

A key-pair to encrypt the mail before it can be sent to the host. Thus, a random key-pair is generated and the string 'hello world' is encrypted using the PostOffice API.

```
PrivateKey myKey = Curve25519PrivateKey.random();

PostOffice postOffice = attestation.createPostOffice(myKey, "hello world");
byte[] encryptedMail = postOffice.encryptMail(toReverse.getBytes(StandardCharsets.UTF_8));
```

Finally, the encrypted message is sent to the host and a response is received, decrypted and printed.

```
// Send mail to host
DataOutputStream toHost = new DataOutputStream(socket.getOutputStream());
toHost.writeInt(encryptedMail.length);
toHost.write(encryptedMail);

// Send enclave response
byte[] encryptedReply = new byte[fromHost.readInt()];
System.out.println("Reading reply mail of length " + encryptedReply.length + " bytes.");
fromHost.readFully(encryptedReply);

// Decrypt enclave response and print
EnclaveMail reply = postOffice.decryptMail(encryptedReply);
System.out.println("Enclave reversed '" + toReverse + "' and gave us the answer '" + new
String(reply.getBodyAsBytes()) + "'");
```

For a more detailed explanation of the hello-world code refer to the below link:
docs.conclave.net/writing-hello-world.html

Running the Conclave "Hello World" application

Download Conclave SDK

In order to compile this sample successfully, kindly download the Conclave SDK and update the location in gradle.properties.

```
conclaveRepo=<path_to_conclave_sdk>
```

Conclave SDK can be downloaded from here:
developer.r3.com/conclave

Make sure you have Java 8 or 11 and Docker installed and running and then follow the following tutorial to setup your machine: docs.conclave.net/tutorial.html#setting-up-your-machine.

Assemble the host

The assemble task will build the app in simulation mode.

```
./gradlew host:assemble
```

To use a different mode use the -PenclaveMode argument.

```
./gradlew host:assemble -PenclaveMode=mock
```

Running the host (Linux)

The below commands will build and run the host app.

```
./gradlew host:installDist
cd host/build/install
./host/bin/host
```

Running the host (MacOS/ Windows)

The below commands will build the host app and mount it into a Linux container.

```
./gradlew host:installDist (For windows use gradlew.bat)

docker run -it --rm -p 9999:9999 -v $PWD:/project -w /project --user $(id -u):$(id -g)
conclave-build /bin/bash
```

Run the below commands from within the docker container to run the host app.

```
cd host/build/install
./host/bin/host
```

Running the client

To run the client use the below gradle command (use gradlew.bat for Windows).

```
./gradlew client:run --args=”Reverse me”
```

You should be able to see the reversed string returned from the enclave in the client terminal along with the attestation information.

Further reading & references

Conclave.net

Conclave developers

Conclave events

Conclave docs

Conclave blog

Conclave videos

R3 is a leading provider of enterprise technology and services that enable direct, digital collaboration in regulated industries where trust is critical. Multi-party solutions developed on our platforms harness the "Power of 3"—R3's trust technology, connected networks and regulated markets expertise—to drive market innovation and improve processes in banking, capital markets, global trade and insurance.

As one of the first companies to deliver both a private, distributed ledger technology (DLT) application platform and confidential computing technology, R3 empowers institutions to realize the full potential of direct digital collaboration. We maintain one of the largest DLT production ecosystems in the world connecting over 400 institutions, including global systems integrators, cloud providers, technology firms, software vendors, corporates, regulators, and financial institutions from the public and private sectors.

For more information, visit
www.r3.com and developer.r3.com

r3.com • developer.r3.com