Enigma Development Update

Introduction

Our team is excited to have almost reached the second milestone of three on the path to enabling secret contracts. We managed to compile and run an Enigma Blockchain full node with wasmi inside Intel SGX. All basic functionality for executing CosmWasm contracts in a secure enclave is now implemented and tested! We also havesanity tests which are passing in the GitHub CI

Since deploying the Enigma blockchain in February of this year, we have been focused on building the necessary components for private computation inside Trusted Execution Environments (TEEs), namely Intel SGX. This work, led by @assafmo, @toml and @reuven, comprises many specific tasks, organized into these project milestones on GitHub:

- Milestone 1 - Integrating Smart Contracts into the Enigma Blockchain (complete)
- Milestone 2 - Executing CosmWasm Smart Contracts Inside the Enclave (nearly complete)
- Milestone 3 - Adding Encryption for Secret Contracts Running Inside SGX

Generally, milestone 2 involved integrating an internal runtime (wasmi) inside the enclave and changing all the machinery to work with that implementation. We also fixed some CosmWasm issues along the way. This milestone, which we anticipated as challenging, turned out to be even more so, given the original runtime that comes with CosmWasm (Wasmer) is incompatible with SGX. This, along with defining clear boundaries between code that lives outside the enclave (untrusted code), as well as inside the enclave (trusted code), meant that we needed to surgically remove Wasmer from CosmWasm internals and develop our own VM interface that interconnects smart contract executions on our blockchain with the new enclave-enabled wasmi runtime.

While setting up new developer environments, we decided to document the process of installing SGX correctly. Here isthat documentation. Below are details of the Milestone 2 development process.

Approach

Secret Contracts on Enigma Blockchain [Secret Contracts and Secret Apps

](/c/secret-contracts/5)

Secret Contracts on Enigma Blockchain This post describes our current thinking around bringing Secret Contract functionality to the Enigma Mainnet, a proof-of-stake-based blockchain based on Cosmos SDK/Tendermint. This chain is currently transaction-only, and secret contract functionality will be added in the future via proposals. The goal of this post is to provide a window into our current thinking. It is not a final roadmap, proposal, or concrete plan, and is subject to change and improveme…

Dev Work

Step 1: stateless contract execution in SGX

The idea was to first develop a simple toy contract that runs inside SGX without encryption. In this task, we didn't have to worry about an API of going in/out of the enclave (which is required for reading/writing state). Just needed to pull the WASM interpreter into the enclave and properly integrate a single in/out point from/to the enclave in CosmWasm.

Step 2: compile CosmWasm to SGX

(Dev discussion/Issue) WASM implementation [Secret Network

](/c/secret-network/46)

Now that WASM is implemented in testnet (using CosmWasm), we're looking at integrating it into SGX, and it's more challenging than expected. The main concern is that CosmWasm uses Wasmer, which is a JIT compiler. This is great, but it doesn't seem to work inside of SGX currently (might change with SGX2), which means we have to revert back to an interpreter. We're looking at integrating WASMI, which is the interpreter we used in Discovery. WASMI is enclave friendly, but CosmWasm right now is tig…

Following this exploration, our dev team integrated wasmi:

GitHub

GitHub - scrtlabs/wasmi-in-enclave-test

Contribute to scrtlabs/wasmi-in-enclave-test development by creating an account on GitHub.

Step 3: research Rust SGX SDK usage

The Rust SGX SDK provides the ability to write Intel SGX applications in the Rust Programming Language. This is the same low-level SDK we used for Discovery. Check it out:

GitHub

GitHub - apache/incubator-teaclave-sgx-sdk: Apache Teaclave (incubating) SGX...

Apache Teaclave (incubating) SGX SDK helps developers to write Intel SGX applications in the Rust programming language, and also known as Rust SGX SDK. - GitHub - apache/incubator-teaclave-sgx-sdk:...

Step 4: initialize cosmwasm-sgx-vm

Here is Assaf, Tom and Reuven's epic pull request for replacing the wasmer runtime with a wasmi runtime running inside an enclave. They started by removing all uses of wasmer in the CosmWasm code base and replacing them with calls to external functions. Then, they worked on implementing a Wasm runtime using wasmi (inside an enclave) which would be compatible with contracts compiled for the CosmWasm runtime built on wasmer. Based on the needs of both sides of this implementation, they defined the interfaces that will connect the two parts.

Step 5: implement wasmi runtime inside the enclave as part of cosmwasm-sgx-vm

For this particular issue, we added the functions needed for CosmWasm to interact with the enclave. This meant creating an API for the untrusted code to interact with the trusted code (enclave call). Generally speaking, ecalls mark entry points into the enclave (untrusted–>trusted), whereas ocalls are exit calls from the enclave (trusted–>untrusted). The API we defined is:

- ecall_init
- ecall_handle
- ecall_query
- ecall_allocate
- ocall_read_db
- ocall_write_db
- humanize_address
- canonicalize_address

Adding humanize_address

and canonicalize_address

was done in order to save a round trip from inside to outside the enclave. Stepping outside the enclave would require us to verify the data we will receive from outside the enclave. This is not only painful, but not doing it would increase the attack surface and increase likelihood of bugs and security vulnerabilities. These functions are pure, meaning they only do calculations and not inputs/outputs, so it was easy to relocate them inside the enclave.

Step 6: add gas metering injection to wasm binaries

This required passing the gas limit from the transaction to our wasmi module, enforcing the gas limit while executing Wasm code, and returning used_gas

after the invocation.

Step 7: instantiate and execute a simple contract with wasmi in SGX

Our test involved running a simple ERC20 contract.

Step 8: figure out how to publish SGX to production

In order to do this, we had to check the enclave compilation process. Enigma is a registered service provider with Intel, which means we can sign production enclaves. Now, we are making sure cosmwasm/lib/wasmi-runtime/Enclave.config.xml

is correctly configured.

Step 9: convert CosmWasm automatic tests to work with wasmi in SGX

Assaf wrote a sanity test and added it to our CI, converted the CosmWasm CI to work with wasmi/sgx. Here is the automatic sanity test passing in the GitHub CI.

Step 10: better log errors for Compute module

First, we want to log important events related to the enclave (cosmwasm-sgx-vm, wasmi-runtime modules). We plan to print the following log messages to stdout:

- ecalls invocations
- ocalls invocations
- gas usage
- wasm errors

Next, we are planning to log important events in go-cosmwasm. By design, enclaves cannot perform syscalls (so it cannot work with the machine's I/O) because that's a security hole. Therefore, we have to explicitly and carefully pass logging data outside the enclave in order to log it.

What does this mean for developers?

Soon, developers will have the ability to deploy smart contracts (not secret contracts) on the Enigma blockchain testnet. The compute module gives the ability to run the CosmWasm state machine inside trusted execution environments maintained by node operators. As we continue pushing toward encryption for the purpose of integrating secret contract functionality, we would appreciate community members doing experiments with any kind of persistent scripts.

Tutorial

GitHub

GitHub - scrtlabs/secret-contracts-guide: An initial walkthrough for working...

An initial walkthrough for working with CosmWasm-based secret contracts on Secret Network - GitHub - scrtlabs/secret-contracts-guide: An initial walkthrough for working with CosmWasm-based secret c...

Learn more and help us with testing by sharing feedback here on the forum:

Developer Walkthrough Update -- Testers wanted! [Developer Help

](/c/developer-help/10)

Dev-X Status Update As you know, we're currently hard at work doing the R&D for upgrade proposals to the Enigma Blockchain-- specifically, introducing secret contracts. The first step, however, will be to add smart contracts. The Enigma Blockchain will use cosmwasm as the foundation for smart contracts. Our first goals were to learn:

What's the current process for setting up a developer environment for the Enigma Blockchain? What's the process for writing a smart contract for execution with co…

What does this mean for node runners?

From this point forward, node runners and specifically validators, would generally need to use SGX-enabled hardware to participate in the network. For our upcoming testnets, this is less of an issue, as software mode can be supported as well (a software simulation of SGX that is supported in Intel machines even if they don't directly support SGX).

Next Milestone

CosmWasm Inside SGX With Encryption