slug: geth-inside-sgx title: "Running Geth within SGX: Our Experience, Learnings and Code" authors: [metachris, frieder] tags: [privacy, sgx] image: /img/posts/sgx-og.jpg hide_table_of_contents: false description: We are happy to publish our efforts and a number of key learnings about running Geth inside SGX, for others to reuse, experiment with, and build upon. forum_link: https://collective.flashbots.net/t/running-geth-within-sgx-our-experience-learnings-and-code-flashbots/938

At Flashbots, we are exploring trusted execution environments (TEEs) such as SGX, as well as other privacy technologies including Multi-Party Computation and Homomorphic Encryption, as important building blocks for trustless collaboration along the transaction supply chain. This is particularly relevant for applications such as decentralized block building and sharing private orderflow.

Today we are happy to publish our efforts and a number of key learnings about running Geth inside SGX, for others to reuse, experiment with, and build upon.

In collaboration with konVera, a software development company with strong expertise in confidential computing, we have achieved a fully working prototype of Geth running inside SGX and syncing with Ethereum mainnet.

You can find the code at https://github.com/flashbots/geth-sgx-gramine.

Working with SGX

Historically the idea of TEEs such as SGX was to isolate only a small critical part of your application within the confinements of an enclave. For this, it is necessary to adapt the code to make it SGX aware. Only later on did people want to use SGX and run existing applications in isolation and do so without modification of the source code.

For this, libOSes such as Gramine, Occlum and EGo have been developed, to provide an abstraction layer to applications which expect to be run in Linux-like environments. We chose Gramine because it is the most mature and stable, and is well documented.

Issues and Challenges

As we began this process, we encountered several challenges and issues that we had to address.

Storing the State

One major problem we faced was the slow performance of Gramine's encrypted file mounts, which made it difficult for Geth to keep up with Ethereum mainnet. This is likely a bug and should be investigated at some point.

To address this, we experimented with using different libOS frameworks, but ultimately found that these alternatives were not suitable for our needs:

- Occlum: performance was promising but it was highly unstable and we couldn't get Geth running with encrypted state (many syscalls missing, unstable, lack of documentation).
- EGo: easy to setup, but much slower than Gramine and did not support encrypted files at all.

Initial Sync

The initial sync process of the chain can take quite a long time, and currently requires up to 800GB of storage.

We chose to copy the unencrypted database from the host into SGX, rather than run the sync within SGX. Copying the database on startup also encrypts the data, which takes about 3 hours because of the data size and cryptographic operations.

Information Leakage

Another issue we explored was the problem of information leakage, which occurs when the host system can extract information about what data is being accessed within the SGX enclave. In the case of Geth, this could potentially leak information about the keys being accessed from the database due to IO and memory access patterns.

This research paper explores the issue of IO leakage in more detail: "*SGX-MR: Regulating Dataflows for Protecting Access Patterns of Data-Intensive SGX Applications*" (Alam, Sharma, Chen, 2021)

In-Memory Approach

In light of these challenges, we decided to try a different approach: storing the entire Geth database in memory and using encrypted swap space to make the approach less resource intensive.

To store the entire chain in-memory, we would need a system with at least 1TB of memory, which is hardware resource intensive and expensive. Another downside of this approach is that there is no persistence, and the state is lost if the Geth application stops.

Luckily, not all of the 1TB need to be accessed by the application at once, a lot of it can be encrypted by the SGX kernel driver and swapped out.

Our initial attempt was to run geth on a 384GB machine with 256GB EPC (SGX protected) memory and 1TB swap, and it worked well (~150 mgasps [million gas per second]). But those machines are costly and hard to come by (Azure only).

We found a trick though: you can use a server with large regular memory and treat it as buffer space. We ended up running on a machine with 500 GBs RAM, 1TB of SSD swap and 64 GBs of protected memory, which are much cheaper and more widely available. It performed at around 130 mgasps which is good enough.

While this approach does have some limitations, we have found it to be a viable option for running Geth within SGX.

Summary

Running Geth within SGX is entirely possible, but both a resource- and time-consuming process. It needs a large amount of memory, has a startup time of around 3 hours, and state is easily lost.

While storing the database in memory and using encrypted swap space provides good performance, it may still be vulnerable to certain types of information leakage. On the other hand, using encrypted file systems within SGX may be the straight forward, less resource intensive way, but it can introduce performance issues and information leakage is even harder to avoid.

Ultimately, it is important to carefully assess the needs of your specific application and choose the solution that best meets those needs.

By publishing our code and learnings, we hope to provide a reference implementation and to stimulate broader exploration and collaboration!

Future work

- Reproducible builds (enable trust into which code is running in SGX)
- Add `flock` system call to Gramine
- Explore slowness of Gramine encrypted FS mounts
- Explore possible IO leakage and implications
- Verify attestations on-chain

Thanks

Special shoutout to Mateusz Morusiewicz and Tomasz K. Stańczak for their support and creativity in making this happen and solving a bunch of problems along the way, as well as to Alejo Salles, Jonathan Passerat-Palmbach and Robert Miller for reviewing the draft of this post.

References

- You can find our code for running Geth inside SGX atgithub.com/flashbots/geth-sgx-gramine
- Join us in the forum, to discuss and collaborate, at https://collective.flashbots.net/t/running-geth-within-sgx-our-experience-learnings-and-code-flashbots/938
- See also the second post in this series:Block Building inside SGX