TEE Rollups: Fixing Access Patterns in TEE-based Smart Contracts with Off-chain Computing

IC3

Follow

--

Listen

Share

This article provides a detailed explanation of the “TEE” project, which won the First Place

of the 2023 IC3 Blockchain Camp Hackathon!

Team Members:

[Andrew Miller

](http://soc1024.ece.illinois.edu/),

[Nerla Jean-Louis

](https://www.ner.la/),

[Yunqi Li

](https://lilione.github.io/), James Austgen, Sylvain Bellemare, Gabriel Arrouye, Bunmi Akintade, Amber Seira, David Hill, Zhengxun Wu

Source Code:

[https://github.com/initc3/tee-offchain

](https://github.com/initc3/tee-offchain)

Overview

Enhancing smart contract privacy is a critical stride towards the development of more useful blockchain applications. Trusted execution environments (TEEs) or secure enclaves are being used in multiple networks (Secret Network, Oasis Network, Obscuro, etc) to enable privacy without significantly increasing computational costs. However, the utilization of TEEs also brings forth challenges, specifically in designing secure network architectures that fully capitalize on the strengths of TEEs while mitigating potential risks. Our recent paper detailing several attacks on these TEE based blockchain networks that broke user privacy guarantees without doing the hard work of breaking into the TEE hardware.

Following the publication of our research on attacks against TEE-based blockchain networks, we concluded that effectively preventing the identified privacy leaks necessitates the use of obfuscation techniques. However, we found that integrating these techniques directly into the “on-chain” execution posed significant challenges due to their high cost and complexity, as the execution would have to be replicated across every node

in the network. To address this, we devised a novel approach by implementing these techniques in the “off-chain” environment, with only a single node needed to run it and post results on-chain later, substantially reducing the costs. At the IC3 2023 Summer Camp, we took this concept further and successfully created proof of concept implementations using our new model on the aforementioned three TEE-based blockchain networks.

Access pattern leakage in TEE-based smart contracts: breaking token transfer receiver privacy

TEEs provide strong hardware-based isolation, ensuring the secure separation of TEE (program running in TEE and data in the TEE’s memory) from the host machine (the operating system and any program running outside of TEE). Through the use of remote attestation a remote user on an entirely different machine can be assured that the correct code is running in the TEE. These two properties align well with the use case of private blockchain networks. The operations performed within the TEE remain concealed from the blockchain node while also providing assurance that the intended code is executing.

Let's explain how TEEs can safeguard privacy in a fungible token application. Secret Network introduces its privacy-preserving fungible tokens, known as SNIP-20 tokens, as an alternative to Ethereum's ERC-20 tokens. The primary objective is to protect the receiver's address, transfer amounts, and user token account balances (although the sender's address cannot be hidden, as it is essential for covering gas fees using the non-private native coin on Secret Network, SCRT). To transfer SNIP-20 tokens to another user, the sender initiates an encrypted transaction containing the receiver's address and the transfer amount. This encrypted transaction is processed within the TEE, where it is decrypted. Subsequently, the sender's and receiver's account encrypted balances are retrieved from external storage, decrypted within TEE, and verified if the sender has sufficient balance to proceed with the transfer. By conducting these operations within the TEE, the transfer amount and user balances remain effectively hidden. The plaintext values are solely exposed within the TEE's isolated memory, ensuring privacy and protecting sensitive data from unauthorized access or visibility on the blockchain.

Although TEEs offer isolation for the code and data running within them, it is crucial to acknowledge that the surrounding infrastructure plays a vital role in maintaining privacy. A well-designed system must consider for example how encrypted data and transactions are loaded into the TEE. In the context of SNIP-20 tokens in Secret Network (the most mature

TEE based blockchain network so far), we've demonstrated in the paper that the receiver's address was not completely hidden due to potential information leakage during the interactions with the external world. By analyzing the encrypted balances loaded into the TEE and correlating them with public sender addresses for token transfers, we could infer the receiver's address.

Preventing access pattern leakage

Preventing access pattern leakage in TEE-based blockchain networks requires the implementation of obfuscation techniques, either at the application layer (smart contract code) or within the blockchain's execution layer. For example, in the token contract, we could do a linear scan over all possible account balances instead of directly loading the sender's and receiver's account balances. This linear scan technique aims to avoid revealing specific addresses accessed by the TEE. During the linear scan, all encrypted balances are loaded and updated, but only the sender and receiver's balances are modified to a different plaintext value.

However, implementing the linear scan on-chain can be prohibitively expensive. In the case of the most popular SNIP-20 token, sSCRT, this approach would incur the loading and updating of over 36,000 records on-chain for each token transfer. As the user base expands, the associated costs increase accordingly.

The linear scan mentioned here is a trivial construction of Oblivious RAM (ORAM), a data structure designed explicitly to conceal storage access patterns. While more complex ORAM constructions can optimize the process, the considerable number of storage accesses and updates remains a challenge.

"Free" TEE execution

Implementing the linear scan technique or other Oblivious RAMs (ORAMs) on-chain would significantly escalate the gas cost for even simple token transfers, not to mention more complex transactions. The additional operations required for storage accesses and updates must be executed by all nodes in the network, leading to higher gas fees for each transaction. To mitigate the problem, we implement the linear scan in the off-chain view only mode, where users can query blockchain states without sending transactions to the network.

Compared to public blockchain where anyone can freely read chain state, in TEE-based blockchain, only authenticated enclaves are capable of decrypting private smart contract state and providing query results to authorized users. For instance, when users need to check their SNIP-20 token balance, they send an off-chain query to a TEE node. The TEE node retrieves their encrypted balance from external storage, decrypts the value, and sends back the re-encrypted value, ensuring that only the querier can decrypt it. Such queries are "view-only" as they do not alter the blockchain state and are "off-chain" because the query transactions are not recorded/ordered on the blockchain. The execution of an off-chain query occurs on a single TEE node and is not replicated across the network. For the networks we looked at– Secret Network, Oasis, and Obscuro–these queries are currently FREE, but there is a per query gas limit imposed to restrict the number of operations performed in a single query. It is anticipated that when TEE-based networks adopt this off-chain query mechanism we introduce, they will implement more effective access control mechanisms to prevent denial-of-service attacks, making off-chain queries not free anymore however still way cheaper than on-chain implementation.

While off-chain queries do not directly update the blockchain state, there is nothing stopping us from using this infrastructure to conduct useful computations with the privacy and integrity

guarantees provided by TEEs. Our aim during the week-long hackathon was to show how this strategy could be used to implement the obfuscation techniques that could hide the recipient address of a token transfer application in an inexpensive–basically free–way.

Explaining TEE Rollups with a toy application: Iterated

Hashing.off-chain hash machine

In our initial demonstration, we showcase the remarkable capability of off-chain queries to conduct computations offline using the current blockchain states. By harnessing this approach, we effectively move the costly hashing computing from on-chain to off-chain in a hash machine application. The hash machine HashMachine(x,n) takes a private input x and a public number of iterations n to compute hash(x)=x1, then hash(x1)=x2, repeating this pattern n times to generate the private output. This computation is non-parallelizable and produces unpredictable output, making it ideal as a Verifiable Delay Function. Another valuable use case for the hash machine application is that in the audiTEE library, it serves to demonstrate the chain of trust for Intel SGX TEE and showcases how to outsource an expensive computation to an SGX enclave while gaining confidence in the output without executing the computation locally.

In our implementation of applications using off-chain queries, we prioritize user experience by hiding the complex underlying processes from them. Users are only required to provide private inputs at the start and receive the private outputs upon computation completion. This ensures a seamless experience for users, maintaining the same interfaces as the existing fully on-chain implementation.

To achieve this, we employ a dedicated worker responsible for submitting off-chain queries to trigger the off-chain computing process. The worker node stores intermediate results until they post the final private output back on-chain. With the use of the worker node, we must ensure two things, 1) privacy — the worker cannot learn anything about the private inputs, intermediate results, and final private output; 2) integrity — the output posted on-chain is indeed obtained from off-chain queries carried out by TEE. For these purposes, we use an authenticated encryption scheme to encrypt the off-chain queries outputs, ensuring both privacy protection and prevention of forgery.

The hash machine example runs as follows. During the instantiation phase, the smart contract generates a private authenticated (symmetric) encryption (AEAD) key on-chain, accessible only to TEE nodes. This key enables encryption of values, ensuring that data originates from smart contract function invocation within the TEE and can only be decrypted by TEE.

Figure 1 shows the hash machine execution in our model. A user posts their input string

through an

on-chain transaction. The worker node calls the view-only function getState()

in the smart contract, employing the AEAD key to encrypt the user input into encInput

, which can only be decrypted by TEE.

The worker initiates an off-chain query by invoking the view-only function processNext

(), passing encInput

as the input. The TEE decrypts the value, performs hashing operations (the number of which is determined by the gas limit of the off-chain query), re-encrypts the result as encHash0

, and returns it to the worker. The processNext

() function is iteratively called with the output from the last call until the desired number of hashes is obtained.

The worker sends the final result to the TEE node and creates an on-chain transaction using the commit()

function to post the result on-chain. After this the user can get the result as they would in a regular application.

We implemented this hash machine example in the TEE based blockchain networks Secret Network, Oasis Sapphire, and Obscuro.

Figure 1: Off-Chain Hash Machine Architecture

Truly private token transfer

The same architecture can help with solving a more intricate problem, the recipient address leakage in privacy-preserving token contracts. As previously noted, the main functionality of a privacy-preserving token contract is to enable private transfers, which involves hiding the transfer amount, recipient's address, and token account balances. The sender's address is made public as they are responsible for paying the transaction fee using the chain's public native coin. However, the amounts involved in deposit and withdrawal transactions are not hidden, as these can be inferred by observing changes in the public token account balances.

Similar to the hash machine application, users submit transaction requests (deposit, transfer, or withdraw) much like they would with current privacy-preserving token contracts, while also providing private inputs to the invoked function. However,

these transactions are not processed instantly. Instead, they are ordered in a sequence and recorded on-chain for subsequent processing. A worker node then processes these transactions using off-chain queries, in the order dictated by the blockchain. At the end, the worker node posts updated account balances back to the blockchain.

In contrast to the hash machine application, which only requires data confidentiality, the token contract additionally mandates recipient privacy. Maintaining this is challenging, as it's possible to ascertain the recipient's address by analyzing the external storage access pattern when the TEE retrieves sender and receiver account balances (see more details in our paper). To prevent this, we incorporate the linear scan functionality, as outlined in the "Preventing access pattern leakage" section, into a view-only function in the token smart contract. By implementing the linear scan in view-only functions, we allow the costly computations to be performed by a single TEE node. This strategy not only minimizes extra transaction fees incurred from the complex linear scan but also maintains privacy and integrity.

To facilitate the transfer of off-chain computation responsibilities between worker nodes and avoid new workers to sync everything from scratch, we implement a checkpointing mechanism in the token contract. The checkpoint state contains a list of all account balances and a sequence number of last transaction processed to get those balances. The worker node gets the encrypted checkpoint state from the view-only function, getState. Because the checkpoint state is encrypted using authenticated encryption the worker cannot manipulate or read the user balances.

The worker consistently submits off-chain queries to invoke the view-only function, processNext, providing the latest encrypted checkpoint state as function input. The function executes the subsequent user transaction request in the sequence and updates the provided checkpoint state accordingly (updating account balances and incrementing sequence number).

In addition to posting the most recent checkpoint state (including updated account balances and sequence number) on-chain after processing a batch of transactions, the worker also must ensure the "side effects" of transaction execution take effect. In essence, a withdraw request could trigger a token transfer from the token contract to a different account or contract, which must be managed when the worker is posting the checkpoint.

We implemented a prototype of the proposed privacy-preserving token contract in both Secret Network and Oasis Sapphire.

Prototype implementations

For our prototype implementations, we worked with three TEE-based smart contract platforms, Secret Network, Oasis Sapphire, and Obscuro. Secret network is a WASM compatible platform that was launched in 2020. Sapphire is the EVM-compatible "branch" of Oasis, which launched their mainnet in 2023. Obscuro is a layer 2 TEE based rollup network that has not officially launched its mainnet launch yet.

We developed the prototypes in the local private network that we set up. Perhaps most interesting of all, we have deployed the privacy-preserving token contract to Oasis Sapphire's public test network. Here, we exemplified a common pattern of wrapped tokens, which involves three sequential steps: 1) depositing tokens to the contract, 2) performing a private token transfer to a second account, and 3) withdrawing tokens from that second account.

In our demo, we have the first account deposit 0.1 ROSE to the contract, process the next state off-chain, and commit the deposit response; request a transfer of 0.05 ROSE to the second account, process the next state, and commit the transfer response; and finally from the second account request a withdrawal of 0.05 ROSE, process the next state, and commit the withdrawal. In the end, the first account retains 0.05 ROSE within the token contract, while the second account holds 0 ROSE in the token contract but 0.05 ROSE in another public token contract.

Corresponding Authors:

[Nerla Jean-Louis

](https://www.ner.la/)(nerlaj2@illinois.edu),

[Yunqi Li

](https://lilione.github.io/)(yunqil3@illinois.edu)

Editor:

[Bria Han

](https://www.linkedin.com/in/jia-bria-h-29113a12a/)(IC3, jh2584@cornell.edu)