

# SGXonerated: Finding (and Partially Fixing) Privacy Flaws in TEE-based Smart Contract Platforms Without Breaking the TEE

Nerla Jean-Louis  
University of Illinois at  
Urbana-Champaign  
IL, USA  
nerlaj2@illinois.edu

Yunqi Li  
University of Illinois at  
Urbana-Champaign  
IL, USA  
yunqil3@illinois.edu

Yan Ji  
Cornell University  
NY, USA  
yj348@cornell.edu

Harjasleen Malvai  
University of Illinois at  
Urbana-Champaign  
IL, USA  
hmalvai2@illinois.edu

Thomas Yurek  
University of Illinois at  
Urbana-Champaign  
IL, USA  
yurek2@illinois.edu

Sylvain Bellemare  
IC3 (Cornell University)  
NY, USA  
sbellemare@cornell.edu

Andrew Miller  
University of Illinois at  
Urbana-Champaign, IC3  
IL, USA  
soc1024@illinois.edu

## ABSTRACT

TEE-based smart contracts are an emerging blockchain architecture, offering fully programmable privacy with better performance than alternatives like secure multiparty computation. They can also support compatibility with existing smart contract languages, such that existing (plaintext) applications can be readily ported, picking up privacy enhancements automatically. While previous analysis of TEE-based smart contracts have focused on failures of TEE itself, we asked whether other aspects might be understudied. We focused on state consistency, a concern area highlighted by Li et al., as well as new concerns including access pattern leakage and software upgrade mechanisms. We carried out a code review of a cohort of four TEE-based smart contract platforms. These include Secret Network, the first to market with in-use applications, as well as Oasis, Phala, and Obscuro, which have at least released public test networks.

The first and most broadly applicable result is that access pattern leakage occurs when handling persistent contract storage. On Secret Network, its fine-grained access pattern is catastrophic for the transaction privacy of SNIP-20 tokens. If ERC-20 tokens were naively ported to Oasis they would be similarly vulnerable; the others in the cohort leak coarse-grained information at approximately the page level (4 kilobytes). Improving and characterizing this will require adopting techniques from ORAMs or encrypted databases.

Second, the importance of state consistency has been underappreciated, in part because exploiting such vulnerabilities is thought to be impractical. We show they are fully practical by building a proof-of-concept tool that breaks all advertised privacy properties of SNIP-20 tokens, able to query the balance of individual accounts and the token amount of each transfer. We additionally demonstrate MEV attacks against the Sienna Swap application. As a final consequence of lacking state consistency, the developers have inadvertently introduced a decryption backdoor through their software upgrade process. We have helped the Secret developers mitigate

this through a coordinated vulnerability disclosure, after which their state consistency should be roughly on par with the rest.

## 1 INTRODUCTION

Ever since the very first projects in the blockchain space, achieving transaction privacy on a public blockchain has been a pivotal issue. Because the drawbacks of pseudonymity are well-known [31, 46, 61], many privacy-focused projects including Monero and Zcash have emerged with the goal of bringing true usable transaction privacy to end users. However, while these projects can handle basic anonymous payment functionality relatively well, the broader question of how to achieve general private computation through smart contracts remains unanswered. While enhanced privacy for more sophisticated decentralized finance applications is a worthwhile goal in its own right, it is also a promising solution to harmful forms of Miner Extractable Value (MEV) in which publically-visible market orders can be frontrun by block producers.

TEE-based smart contracts are an attractive solution. Compared with purely cryptographic approaches such as zero-knowledge proofs (ZKP) and secure multiparty computation (MPC), trusted execution environments (TEEs) — especially Intel SGX — are more efficient and are very expressive as well, capable of running existing smart contract applications from Ethereum and Cosmos. A line of research works has explored the design of secure TEE-based smart contracts [9, 14, 79, 82]. In this setting, the attacker model includes the untrusted host in possession of the TEE, where the TEE must rely on the host for all network access and to respond to queries about external data, such as clock times. In terms of real-world deployment, TEE-based smart contracts appear poised to grow. Secret Network (formerly known as Enigma [82]) launched the first public deployment of a TEE-based smart contract system in 2020. As of Feb 2023, it has a market cap of approximately \$200M including native and bridged tokens.<sup>1</sup> Other TEE-based smart contract platforms, including Oasis [55], Obscuro [13], and Phala [59], are at varying stages of public launch — each of them has deployed at minimum a public test network with open-source code, often with a considerable backing from investors and interested users<sup>2</sup>.

This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license visit <https://creativecommons.org/licenses/by/4.0/> or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.



*Proceedings on Privacy Enhancing Technologies YYYY(X), 1–18*  
© YYYY Copyright held by the owner/author(s).  
<https://doi.org/XXXXXXXX.XXXXXXX>

<sup>1</sup><https://secretanalytics.xyz/>

<sup>2</sup>We survey the usage and fundraising situations for each in Appendix C

Li, et al. [38] examined the research literature and pre-production industry whitepapers, pointing out several potential areas of concern. Looking at real-world implementations, the discussion of security issues in TEE-based smart contracts has focused largely on compromises of the TEE instantiation itself, specifically SGX, which has had to recover from a sequence of catastrophic vulnerabilities [7, 73, 75, 76]. The exposure of Secret Network to AepicLeak [7] highlighted the importance of compartmentalization and key rotation [77]. Although having systems in place to mitigate risks of TEE failures is crucial, focusing solely on this would miss the bigger picture of other forms of vulnerability.

In this paper, we ask the following: besides hardware vulnerabilities in the TEE itself, are TEE-based smart contracts implemented today vulnerable to non-TEE attacks? We analyze the codebases and documentation of Secret, Oasis, Obscuro, and Phala, especially focusing on the interface between untrusted and trusted code and the interaction between newly written code and software inherited from existing plaintext blockchains. We summarize the main lessons learned:

*Access pattern leakage is a concern for all the systems we reviewed.* This is especially a problem for fungible token applications ported from plaintext blockchains, such as ERC-20 tokens. These platforms support stateful contracts and are built around a high-performance key-value storage library that persists on disk, such as LevelDB. However, the patterns at which these databases are accessed when processing confidential transactions can themselves leak crucial information. In the worst case, this reveals to the untrusted operating system the sender and the receiver in every token transfer, violating the desired transaction privacy guarantees. This is the case for SNIP-20 tokens (privacy-enhanced fungible tokens based on the ERC-20 and CW-20 standards) on Secret Network, the only ones in use today. This leakage is also present in Oasis Sapphire and Cipher, suggesting that ERC-20 tokens that are naively ported would also fail to provide privacy. Obscuro and Phala mitigate this to some extent because their use of block-based paging means that the access pattern is only leaked to the nearest 4 kilobytes.

A comprehensive fix to access pattern leakage could make use of oblivious data structures (ORAM) or encrypted database techniques, found in other TEE-based distributed systems, but have not yet been adapted to smart contracts [4, 17, 26, 47, 70]. The MobileCoin [33] cryptocurrency (developed by Signal [1]) also uses ORAM with SGX to provide privacy-preserving transaction retrieval [43]. These solutions show the viability and scalability of a TEE solution that implements ORAM for interacting with untrusted storage.

*State consistency and replay defenses are critical to privacy but do not apply intrinsically in L1 TEE-backed blockchains.* The need for rollback protection for TEEs in general has been discussed in previous work [42], and has featured prominently in security research papers on TEE-based smart contracts [14, 32, 38]. This seems like a natural synergy for the two technologies – blockchains are good for committing to a sequential ordering of published transactions, and TEE-based smart contracts must rely on this. Li et al. suggested that state consistency and blockchain data confirmation come along as "intrinsic features" for a Layer 1 (L1) architecture, meaning that the consensus protocol to create new blocks is carried out by the same validator nodes that execute smart contracts using TEEs [38]. It turns out that even though Secret Network is considered an L1,

Secret Network's design is incomplete in that it does not incorporate any defense against replay or rollback attacks. The untrusted host is therefore able to simulate or replay transactions in any order, regardless of what appears on-chain. Oasis and Phala have already implemented defenses against these, and it is on Obscuro's list of requirements prior to launch.

*Design flaws in TEE-backed blockchains lead to practical attacks without compromising the TEE.* Since Secret Network is the only TEE-based smart contract system with live running applications, we explore the impact of the above vulnerabilities. SNIP-20 tokens are believed to hide account balances, and the receiver and token amount involved in each transfer transaction [65]. Beyond just identifying vulnerabilities, we show how these can be exploited to break the advertised privacy guarantees. In fact, due to the fine-grained access pattern, just adding print statements to the debug log is enough to trace the senders and receivers in each token transfer.

We design new transaction simulation attacks that enable the forensic analysis of past and present SNIP-20 transfers, as well as query the balances of individual accounts. We also show how partial storage replay can amplify the attack, meaning attackers do not need capital deposits of their own to carry it out. Besides transaction privacy, we also show that replay defenses are critical for MEV prevention, one of the benefits of TEE-based smart contracts. Our attacks can be used by validator nodes to strategically frontrun automated market maker (AMM) swaps. All of our attacks can be performed by an unprivileged node (e.g., no stake deposits or developer keys are needed), without relying on any vulnerabilities of the underlying TEE, and without requiring any kernel patches. Given the threat, we have coordinated with Secret Network's development team on vulnerability disclosure and helped deploy mitigations.

*The need for software updates creates a backdoor hazard.* Every blockchain needs a software upgrade mechanism, though this is notoriously challenging [6] even before TEEs are involved. In order to update software, existing enclaves must transfer key material to enclaves running the new software. In Secret Network, the only policy enforced by the enclaves is code signing. Unfortunately this means that Secret developers have the ability to unilaterally apply updates to the software on their own machines, enabling them to decrypt every transaction - an unintentional backdoor. Oasis and Phala both implement not just a distributed code signing authority, but require proof of on-chain publication of the software binaries to hold them accountable. Obscuro has not implemented such a process but lists it on their requirements prior to mainnet launch.

To summarize our main contributions:

- We analyze the codebases of existing TEE-based smart contract systems that offer at least public test networks. Specifically we focus on issues involving the boundary between the trusted and the untrusted code, and especially the application storage.
- We provide evidence that access pattern leakage from the underlying key-value storage means that ordinary token contracts ported to any of the platforms today would fail to provide strong privacy. We discuss possible mitigations for the future including ORAM or decoys.
- For Secret Network in particular, we demonstrate new transaction simulation attacks, demonstrating that a lack of state consistency leads to breaking all the privacy guarantees of existing tokens. We have conducted a coordinated vulnerability disclosure and

helped developers with mitigations. They have now added replay defenses, although receiver privacy for SNIP-20 tokens is left currently unresolved.

## 2 BACKGROUND

**Blockchains and smart contracts** A blockchain is a decentralized ledger whose state is replicated and agreed upon by mutually untrusting nodes. A smart contract is a program whose execution is recorded on this ledger. Many blockchains, most notably Ethereum [23], support smart contracts. In Ethereum the state of all contracts on the chain is public, and identities are pseudonymous. Many works [46, 50, 60, 61] have shown that these pseudonyms are a weak form of privacy. The public nature of the Ethereum blockchain impedes many applications, though in particular we are interested in decentralized finance. Decentralized finance facilitates manipulation of financial assets without relying on centralized authorities like banks, and its applications include payments, lending, auctions, and other financial derivatives.

**Fungible tokens.** Fungible tokens serve as currency that can be owned and transferred using smart contracts in decentralized finance applications. The pseudocode for the `transfer` function of a fungible token is shown in Figure 1a. These contracts can be customized for specific tokens while maintaining standardized interfaces such as ERC-20 on Ethereum and SNIP-20 on Secret.

**Privacy expectations of fungible tokens.** The appeal of Secret Network is that existing blockchain paradigms from the world of transparent blockchains, such as ERC-20 tokens in Ethereum, can be easily translated over to receive immediate privacy guarantees.

On a transparent blockchain all of the transaction data is viewable on chain. For Alice to send \$5 to Bob, the network must know Alice’s current balance of \$10, Bob’s balance of \$20, the value of the transaction \$5, and Alice’s and Bob’s addresses. We summarize the possible privacy goals for transactions as: *sender privacy* and *receiver privacy* to hide the addresses involved in the transaction and *value privacy* and *balance privacy* to hide the amounts involved in the transaction.

For SNIP-20 tokens, the sender’s address is automatically made public due to the nature of executing transactions on a public blockchain. Transaction fees in Secret are paid using the public native token Secret Token (SCRT), which must be checked for sufficient balance by running nodes. SCRT tokens are converted into the SNIP-20 token called Secret Secret Token (sSCRT) by wrapping it in a smart contract. SNIP-20 tokens like sSCRT are believed to hide receiver address, transaction value, and balances of involved parties. sSCRT is the most widely used smart contract on Secret, with approximately 36,000 token accounts and approximately 500,000 interactions<sup>3</sup>, while other popular tokens include bridged versions of stablecoins like USDT and USDC, Bitcoin, Ethereum, and Monero. **Trusted execution environments.** A *trusted execution environment* (TEE) or secure enclaves are an isolated environment, often implemented with hardware, that provides privacy and integrity guarantees about the code it executes. Intel Software Guard Extensions (SGX) [2, 18, 29, 45] is the basis for Secret and other upcoming systems. It is a hardware TEE that supports remote attestation [63].

Hardware TEEs such as SGX are designed to defend against an attacker with OS kernel-level access. Due to the hardware isolation of these TEEs such an attacker should not be able to tamper with or observe the execution of a running TEE program. This model describes a malicious blockchain node operator running a cloud tenant with a dedicated instance. Because these guarantees only apply to a running TEE, any restart of the TEE will result in data being lost. SGX therefore makes use of sealed data to encrypt state on disk and securely load it into the TEE.

Remote attestation allows a node operator to prove to other network participants that they are running a legitimate TEE with the correct version of the software. At a high-level remote attestation is an interactive protocol where a measurement of the data and software running in the enclave is taken and signed using a secret key hidden in the hardware. This measurement and signature is checked and endorsed by the hardware manufacturer based on their secret knowledge of the hardware keys determined during the manufacturing process.

Following successful remote attestation, the operator’s TEE can access confidential data, such as smart contract states, without requiring the network to trust individual node operators or cloud providers running the TEE. Even with physical access, breaching an Intel CPU is thought to be expensive and difficult [18]. The root of trust remains at the manufacturer (Intel in the case of SGX), responsible for correctly manufacturing hardware and issuing certificates only for appropriately manufactured TEE-enabled CPUs.

Even assuming this root of trust, however, the privacy and integrity guarantees provided by Intel SGX have been broken in practice. Papers such as [7, 11, 76] have identified vulnerabilities in the TEE implementation, resulting in memory and CPU register leaks, as well as the creation of fake attestation reports. Side channel attacks allow attackers to extract secrets from the enclave by observing resources such as caches [10, 15, 20, 27, 48, 64], branch predictions [21, 30, 37], and memory usage [49]. Tools such as SGX-Step [74] provide attackers with instruction level control and increase the efficiency of side-channel attacks.

## 3 OVERVIEW OF TEE-BASED SMART CONTRACT PLATFORMS DESIGN

In this section, we review some general design patterns used to realize TEE-backed blockchains. We focus on explaining Secret Network, though the general design is applicable to others.

**Structure of TEE-based blockchains.** In TEE-based systems, the code is divided into trusted and untrusted portions. The trusted portion runs inside the TEE with integrity and privacy guarantees, while the untrusted portion runs on the host operating system and acts as an interface between 1) the TEE and clients, 2) TEE and the consensus layer, 3) TEE and persistent storage. Therefore, the main threat model for a TEE-based blockchain is that the attacker has full control of the untrusted portion of the nodes.

Both the enclave and the client need access to the encrypted transaction data, necessitating the use of a Diffie-Hellman (DH) key exchange scheme to generate a shared encryption key. This symmetric key is unique to each transaction. The trusted enclave calculates the key using an elliptic-curve DH key derivation protocol, incorporating the io exchange private key (derived from the

<sup>3</sup><https://secretnodes.com/contracts>

```

1 storage balances: mapping { address => uint128 }
2 contract function transfer(sender, receiver, amount):
3     require contract_status == NormalRun
4     require balances[sender] >= amount
5     balances[receiver] += amount
6     require balances[receiver] + amount <= MAX_BAL
7     balances[sender] -= amount
    
```

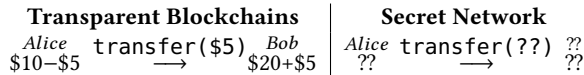
(a) Contract pseudocode for transferring fungible tokens.

```

get 0x6592ff[ . . . ] // contract status
get 0x9e82b0[ . . . ] // sender balance
get 0x3d4982[ . . . ] // receiver balance
    
```

(b) The log of get traces in a SNIP-20 transfer.

**Figure 1:** This figure illustrates the access patterns visible to the untrusted host of an SGX Enclave within which a node runs a token transfer contract. The left hand side includes the operations running in the contract and the right hand side shows the corresponding pattern seen by the host for each step.



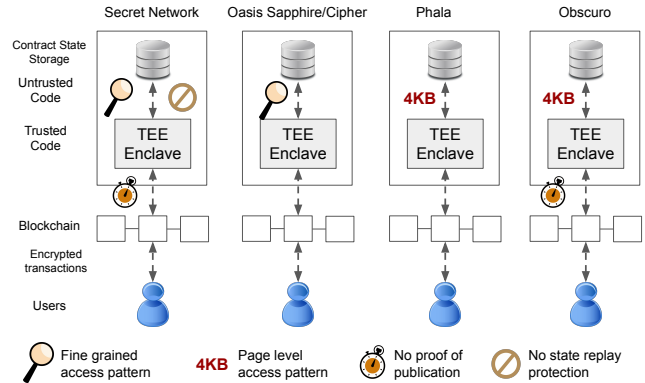
**Figure 2:** Privacy Goals for SNIP-20 tokens versus Transparent tokens such as ERC-20.

setup secret and exclusively accessed by the trusted portion) and the client’s wallet public key. The client inputs their wallet private key and the io exchange public key into the key derivation protocol and acquires the same key. The io exchange public key is stored in the publicly accessible file named *genesis.json* on Secret’s website. When a client submits a transaction to the network, it encrypts the transaction using the unique encryption key. The untrusted host code inputs the encrypted transaction to the enclaves, where it is decrypted, processed, and its results get encrypted for secure transmission back to the client through the untrusted code. Only the client who creates the transaction can decrypt its results.

The relevant threat model for a TEE-based blockchain is that the attacker is in full control of the untrusted portion of the application running on its own nodes. An application using the TEE must assume that the host is malicious and will attempt to learn information about the confidential state using any queries it is permitted to make. The TEE can only help to enforce the policies that the enclave program explicitly writes.

In SGX the interface between the untrusted and the trusted portions of the application are known as `ecall`’s and `ocall`’s which make a function call from the untrusted process to the trusted (`ecall`), or vice versa (`ocall`). Since these are function calls, arguments and return values can be used to pass data between them.

**Blockchains and proofs of publication.** The separation of state machine replication (and execution) from a global consensus layer has been considered in prior work, notably [32] and [14]. [32] considers a more general model, where an untrusted host provides state and services that must be committed to a public ledger. They go on to provide a framework to achieve security against misbehaving hosts by introducing *proofs of publication* to the blockchain before clients accept any request responses from hosts. [14] specializes this model to a scenario similar to that of Secret Network, where a large number of TEE nodes can accept messages and requests, but only a small fraction of these nodes participate in consensus. They utilize proofs of publication to achieve what they call *atomic delivery*, that is, given any client transaction Tx, the output of this



**Figure 3:** High-level architecture of TEE-based smart contracts, focusing on the interfaces between the TEE and external storage and between the TEE and the blockchain.

transaction is revealed to the client if and only if the consensus nodes have provided a valid proof of publication for Tx.

Secret Network, on the other hand, does not require these proofs of publication, breaking the atomic delivery requirement. Any Secret node can unilaterally execute a valid transaction and obtain an output (albeit in part encrypted). While an honest client may not accept this output, the untrusted host of the executing TEE can still glean important information from metadata or through its own transactions as we will soon demonstrate.

**How to pass private outputs to individual users.** Ordinary transparent smart contracts typically define "query" or "getter" methods in order to view the current application state (for example, to check a token account balance). User wallets typically use RPC servers to process these queries on their behalf. Private TEE smart contracts need a way to send query results from a remote enclave to the client, without revealing them to the untrusted host.

There are two main approaches that all of these networks follow: The first approach is based on encrypted log events. As a contract executes a transaction, it may write outputs to a log. This log is included in the public blockchain data. The contract encrypts by associating the intended receiver with a public encryption key. Since users are already identified by their address (a hash of a public key for digital signatures), it makes sense to extend the address format to include a public key for receiving outputs. Oasis and Obscuro

	Access Leakage	Tx Replay Protection	Storage Replay Protection	Upgrade Transparency
Secret <sup>a</sup>	○ key	○ → ●	○ → ☉	○
Oasis <sup>b,c</sup>	○ key	●	●	●
Obscuro <sup>d</sup>	● page	○ → ☉	●	○ → ☉
Phala <sup>e</sup>	● page	●	●	●

<sup>a</sup> SecretNetwork fd2745 <sup>b</sup> oasis-sdk c82319 <sup>c</sup> oasis-core 5302f72

<sup>d</sup> go-obsкуро b5a79e <sup>e</sup> phala-blockchain 55479c

**Table 1: Analysis of security and privacy hazards in existing TEE-based smart contract codebases at time of writing. An arrow and filled circle, → ●, indicates a fix as a result of our disclosure; → ☉ indicates a planned mitigation.**

support atomic delivery so that the outputs would not decrypt until the transaction is committed. An oracle could be used to determine the transaction confirmation.

The second approach uses off-chain queries. Nodes with enclaves are allowed to process “queries” executing the smart contract and retrieving the data from the local database state of the TEE node. A viewing key is shared between the smart contract and a user. The viewing key is stored in the contract’s application state, and the contract query results are encrypted using this key before they are passed on to the host. These off-chain queries are susceptible to access pattern leakage.

## 4 PRIVACY ANALYSIS OF REAL-WORLD TEE-BASED BLOCKCHAINS

We analyze the landscape of TEE-backed blockchain platforms in practice through the codebases of Secret Network, Oasis, Obscuro, and Phala. Table 1 and Figure 3 summarize the different privacy hazards relevant to TEE smart contracts detailed in this section.

### 4.1 Integration with existing blockchain codebases and smart contract languages

**Secret Network.** As we discussed in Sec. 3 Secret Network’s code base, like other SGX applications, is divided into a trusted portion that is executed in the TEE and an untrusted portion that is run on the host operating system. Secret Network’s untrusted code is packaged into two binaries: (1) the `secretd` binary containing a command line interface client to interact with the network and to start a node server, and (2) the `libgo_cosmwasm.so` binary, which provides a programmatic API to `secretd` to interact with contract code and make calls to the TEE code. The code for both `libgo_cosmwasm.so` and `secretd` can be modified by a malicious host.

Much of Secret Network’s codebase is inherited from the Cosmos Software Development Kit (Cosmos SDK) [68]. The Cosmos SDK code handles message authentication and processing of transactions and blocks. Secret Network uses Tendermint [69] to handle consensus and communication between nodes and clients. In order to use the Cosmos SDK and Tendermint frameworks Secret Network extends Cosmos SDK’s default `Application` interface with functions and variables specific to their implementation. This includes the function `DeliverTx` that triggers the execution of a transaction after consensus. The Secret Network contract code is implemented using a Rust-based interface, called `CosmWasm`, which

integrates with Cosmos SDK. `CosmWasm` contracts are compiled into binaries, which are eventually executed in the TEE.

Contract code storage occurs without the use of TEE and is stored unencrypted on disk. To initialize, execute, or query a contract the node server retrieves the stored contract code, a reference to the contract’s key-value stores, and the encrypted message. Then pass these to the `libgo_cosmwasm.so` interface.

`libgo_cosmwasm.so` passes messages to the TEE using the `ecall` interface used for communication between the untrusted and trusted portions of SGX applications. `ecall_init` is used to initialize a contract, `ecall_handle` to execute a transaction on a contract, and `ecall_query` to query a contract. The trusted code is responsible for executing Rust contract code using Secret Network’s `CosmWasm` interface.

`libgo_cosmwasm.so` also defines callback handlers for `ocalls` that are used for communication between the trusted and untrusted code bases. They include `ocall_read_db` and `ocall_write_db` to read and write values to the contract’s key-value store that is stored on the untrusted host OS.

Once a transaction is committed, the transaction data itself, as well as the updated account balances of the sender and receiver are stored in an encrypted database or key-value store. The key-value store is accessible through a storage API but is stored in encrypted form on the untrusted host, due to the TEE’s space constraints. To read or operate on a record, the TEE retrieves it from untrusted storage and decrypts it in the TEE.

**Oasis.** Oasis Network [55] is a Layer 1 (L1) blockchain in that it runs its own standalone consensus layer built on Tendermint (like Secret Network). On top of the consensus layer is an execution layer organized as independent “ParaTimes,” which make use of the consensus layer. They are implemented using the `oasis-runtime-sdk` and are customized for different purposes.

There are two ParaTimes, Sapphire and Cipher, that provide confidential smart contracts. The main difference between the two is that Sapphire is EVM-compatible while Cipher runs Wasm-based contracts. Both do not further modify `oasis-runtime-sdk`. Emerald is an EVM-compatible ParaTime that does not aim to provide any privacy, which we ignored throughout our analysis.

**Obscuro.** Obscuro [13] is an L2 rollup network that aims to address privacy concerns in the transparent blockchain Ethereum. During each epoch aggregator nodes compete to become the leader and generate a new rollup transaction that is posted to the L1 Ethereum chain. This leader election is based on a random nonce that each aggregator generates in their TEE. All contract data is encrypted and added to the rollup. Because the data is replicated on chain they do not rely on the TEE for data integrity.

To participate in the next leader election, aggregator TEEs must validate the previous L1 blocks containing the previous rollup in the sequence. They are therefore reliant on the L1 for safety of the consensus mechanism.

Obscuro is designed to be EVM-compatible for easy porting of smart contracts. While it is the sole framework that encrypts contract bytecode, the degree to which this feature offers explicit privacy protection remains uncertain. In fact, it is possible for attackers to probe the contract code via side-channel attacks like SGX-Step [74].

**Phala.** Phala [59, 80] (Phat Contract) is an L1 blockchain implemented as a Polkadot parachain. Unlike the others, their focus is to act as middleware for other blockchains. Their goal is to enable serverless smart contracts called "Phat Contracts" similar to Amazon Lambda functions that can handle concurrent user requests and do not rely on fast updates to a global state.

They note their goal is not to replace existing smart contract infrastructure: "Instead of implementing an ERC-20 token with Phat Contract (whose balance has to be stored on-chain), we recommend deploying your ERC-20 contract on Ethereum and using a Phat Contract to operate it"<sup>4</sup>. Depending on the amount of stake held by the contract owner, a set of worker nodes are assigned to a contract. Although they encourage stateless smart contracts, they also support "vanilla on-chain states and transaction processing" for applications that need it, hence such contracts would be subject to the same state consistency hazards we consider.

Phat Contracts are written in Rust-based Ink! <sup>5</sup> language, compiled into Web Assembly and executed in TEE using Parity substrate<sup>6</sup>. The contract code and the initial parameters are public.

## 4.2 Storage access pattern leakage

All of these blockchains inherit a key-value storage programming model for their existing smart contract languages. This is backed by encrypted data stored locally on the node performing the execution. In the example of a private token, each account balance may be stored as a separate record.

None of the systems whose codebases we evaluate hides access pattern, and so an untrusted host OS can potentially learn private information (such as receiver addresses in the private token example) by monitoring accesses to the key-value storage. This is most pronounced in Secret and Oasis, which leak the exact key that is being accessed. Obscuro uses a TEE-based database library called EdgelessDB, which reads and writes to the filesystem using 4KB blocks. Phala uses an entirely in-memory data structure for contract state, thus it would leak to the untrusted OS page fault handler which memory page is being accessed. To address this issue, it is crucial to employ effective obfuscation techniques such as the use of oblivious RAM/data structures or encrypted databases.

**Secret Network.** Each Secret Network contract has access to a key-value storage, where it implements an application-specific state. Each value is encrypted using an authenticated encryption mode, using the plaintext key as a tag. In this way, encrypted values cannot be replayed on different keys (important for Sec. 5.5).

A separate encryption key is derived for every contract. Each plaintext key-value store record is encrypted using the contract's encryption key and a fixed initialization vector (IV). The IV must be fixed, i.e., the encryption must be deterministic since the encrypted field name is used to retrieve the encrypted value.

**Oasis.** Oasis, like Secret, also uses an underlying database in the untrusted host, in this case BadgerDB. This similarly leaks the access pattern of each accessed record.

**Obscuro/EdgelessDB.** In addition to storing all contract data on chain Obscuro uses EdgelessDB to store and retrieve data during

the execution of the smart contract. EdgelessDB runs in a separate TEE and uses a RocksDB backend. The data is stored in sorted string table (SST) files, and accessed in 4KB blocks.

**Phala.** While Phala encourages mainly stateless applications, persistent contract state is nonetheless supported. Since significant usage is not anticipated, they use an in-memory data structure, periodically saved in its entirety to disk as a "checkpoint." Compared with the alternatives like LevelDB, this incurs greater cost when saving and restoring from these checkpoints.

In addition to this contract state Phala allows contract owners to bring their own external database services that the contract can connect to via HTTP requests. This model requires contract owners to handle access patterns hiding themselves. They can choose to store and load data in blocks rather than fine-grained key-value pairs, but they are still subject 4KB limit for loading pages in a TEE. Contract owners must also ensure that the HTTP queries themselves do not uniquely leak the data block or key being accessed. Amazon S3 is one of the supported external database services in Phala. To perform a request to S3, the URL contains a unique identifier for the object and bucket being accessed. The access pattern could be inferred from logs viewable by the S3 bucket owner <sup>7</sup>. We use a half circle in Table 1 since Phala ensures storage replay protection for contract state but not external database accesses.

## 4.3 State consistency and blockchain data confirmation

The ability to perform transaction simulations can undermine privacy goals for TEE-based smart contracts. Even if transactions are encrypted and private their execution can have some side effects that change some public states, e.g. the public pool balance of a token will change depending on the amount of tokens traded in a private transaction. In the expected case, 1) the TEE only executes a transaction after it has already been committed/finalized by the validators' consensus, and 2) the execution permanently modifies the stored records on disk. However, these rules are not enforced by the TEE and are only expressed in the untrusted code. The result is that a validator can run *simulations* of a block prior to consensus, varying the subset of included transactions and their execution order to maximize profit.

Ensuring the validity and up-to-dateness of external storage data is crucial for secure TEE-based smart contract platform design. Failure to do so may result in storage replay attacks, which can lead to the exposure of private information.

**Secret Network.** In Secret Network, there is no Proof of Publication mechanism, so uncommitted transactions can be run against the local TEE. To carry out a simulation attack, the untrusted code is modified so that a) the `call_handle` can be invoked directly to execute a transaction without waiting for network consensus, and b) the `call_read_db` and `call_write_db` handlers provide a "snapshot"/"restore" mechanism to reset the simulated contract state. The potential for transaction replay is acknowledged in Secret Network developer documentation as "Theoretical Attacks" but is not considered a practical concern [22].

<sup>4</sup><https://wiki.phala.network/en-us/build/general/intro/>

<sup>5</sup><https://paritytech.github.io/ink/>

<sup>6</sup><https://www.parity.io/technologies/substrate/>

<sup>7</sup><https://github.com/Phala-Network/phala-blockchain/blob/a6b6c7f/crates/pink-libs/s3/src/lib.rs#L123>



As mentioned, Secret Network stores records with encrypted field names and encrypted values. In more detail:

```
(Enc_key(field_name, Tag=""),
 Enc_key(value, Tag = Enc_key(field_name, Tag=""))).
```

While each record has an authentication tag (specifically AES-SIV authenticated encryption), such that values cannot be replayed across different records, this does not prevent the replay of previous values for the same record.

**Oasis.** Oasis uses a Proof of Publication similar to that described in Ekiden, in which the enclave verifies that a transaction is committed before executing. A slight difference is that in Oasis [55], transactions are executed before confirmation, however the decryption key to their results and events is not released until they are committed on-chain.

Oasis comes with a Merklized Key-Value Store (MKVS). All data provided to the enclave is authenticated by the current root hash of the state Merkle tree stored inside the enclave.

**Obscuro.** As of writing this paper, Obscuro does not implement any method to prevent simulation attacks, but they propose strategies to defend against these attacks in their design documentation<sup>8</sup>. Obscuro uses Sequencer nodes to order transactions before they are included in a rollup transaction. In order for the Sequencer to simulate different orderings of transactions it would have to be restarted using an old state. They plan to require the sequencer to be delayed or re-register to the network after restarting so they cannot execute multiple simulations in the same epoch before proposing a block.

As mentioned in Section 4.2, Obscuro makes use of EdgelessDB which uses sorted string table (SST) files. These SST files consist of 4KB blocks and are append-only (meaning blocks are not modified after being written). EdgelessDB notably aims to provide integrity guarantees but does not aim to prevent rewinding the entire database to a previous state. The position in the block and a file-specific key is used as a parameter for encryption to get position-dependent authenticated encryption. This prevents fine-grained replays because blocks cannot be swapped between files and values cannot be changed in a block [16].

**Phala.** The contract state in the Phala blockchain is determined by a sequence of transactions executed in order in a TEE. The TEE also validates the blocks containing these transactions before writing the expected outputs back to the blockchains in "checkpoints" [80]. The transaction order is determined before execution.

Replay attacks on contract state are not possible in Phala. As described in Section 4.3 transaction ordering occurs before execution and this ordering is validated in the TEE. If a contract makes use of an additional external database it falls to the contract owner to ensure that a malicious worker node cannot replay old database values when the contract makes an HTTP request to the server. This can be done by including a random nonce in the requests that are signed by the server in the response.

#### 4.4 Software update transparency

Software upgrades are notoriously challenging for blockchains [44], even before we consider privacy and TEEs. In short, to avoid partitioning the network, it is important that all nodes participating

in the network can apply updates simultaneously. Additionally, developers should not have the ability to unilaterally change the rules, e.g., to print more coins.

The standard process involves several steps. First developers publish a proposed software update. The upgraded software initially behaves the same as existing software so that node operators can gradually update without risk of partition. An on-chain vote is collected, such as through miners or validators signaling support for/against the blocks they produce, and with enough votes it is approved. The upgraded software includes a "flag day" timestamp (or block number), after which the software switches to the new behavior; at this point, any nodes that have not yet updated will find themselves partitioned from the network.

The additional challenge in TEE-based smart contracts is that enclaves running the old software must somehow transfer key material to enclaves running the upgraded software, but only after the update has been approved on-chain. In Secret Network, although software updates follow the on-chain approval process outlined above, the enclaves do not enforce this policy. Instead, key material is transferred between versions through the use of SGX-sealed data.

As mentioned in Section 2, TEEs use sealing to survive power resets and to make use of on-disk storage. Sealing in SGX takes one of two modes: MRENCLAVE, which means that only the exact same TEE that created the sealed file can unseal it; and MRSIGNER, which means that any enclave code signed by the developers can load the sealed file. Although using MRSIGNER simplifies the upgrade process, it means that the developers have the unilateral authority to secretly decrypt every transaction on the network, simply by code-signing a TEE that outputs the master key in plaintext and running that on their own node. In other words, the upgrade process has introduced an unwanted backdoor. Furthermore, as this could be carried out offline, there is no way for developers to provide evidence they have not exploited this.

Obscuro similarly uses MRSIGNER to seal enclave data on their public testnet. As a result of our disclosure, they have committed to transition to MRENCLAVE prior to their mainnet launch.<sup>9</sup> They propose to do these updates by creating a transaction on the network to propose the upgrade and allow for disputes/objections to the update. After the upgrade is accepted all enclaves will be required to use the new version. Secret keys are transferred between versions of the enclave via an RPC call.<sup>10</sup>

Phala and Oasis already use MRENCLAVE sealing. For upgrades, their enclaves check that an update has been approved on-chain. As this relies on proof of publication, Obscuro and Secret will need to fix this first before changing their update process.

## 5 FLESHING OUT THE ATTACKS ON SNIP-20 TOKENS IN SECRET

It is clear that the vulnerabilities noted in the previous section may have some effect on the privacy properties of the systems in which they are present. But are they real-world issues? To evaluate this, we analyzed Secret Network, since it is the only of the TEE-based systems advertising strong privacy guarantees with currently

<sup>8</sup>[https://github.com/obscuronet/go-obscuro/blob/e626db/design/fast\\_finality.md#possible-designs-for-preventing-value-extraction](https://github.com/obscuronet/go-obscuro/blob/e626db/design/fast_finality.md#possible-designs-for-preventing-value-extraction)

<sup>9</sup><https://github.com/obscuronet/go-obscuro/pull/1065>

<sup>10</sup>[https://github.com/obscuronet/go-obscuro/blob/bacb60/design/security/Upgrade\\_Design.md](https://github.com/obscuronet/go-obscuro/blob/bacb60/design/security/Upgrade_Design.md)

in-production-use applications. Especially popular are its fungible privacy tokens, known as SNIP-20. SNIP-20 tokens have nearly identical functionality to the ERC-20 tokens from plaintext blockchains, following the pseudocode mentioned in Sec. 2. The balances are stored as smart contract application states in the encrypted key-value storage. Users can transfer a token balance by including a "transfer" message in the transaction payload. In order to query the account balance, users need to set a viewing key through an on-chain transaction, but afterward they can query their balance with the help of any network node. At the time of writing, while the market cap of circulating SCRT (i.e., the native asset that is not "staked" in the consensus protocol) is \$169 million, of that a fraction \$5.87 million is wrapped in sSCRT<sup>11</sup>. As explained in Section 2 the sender address in SNIP 20 tokens are public, therefore we can query the number of unique addresses that have interacted with the token contracts as well as the number of transactions sent to each contract in Appendix Table 3. The most popular SNIP-20 tokens we investigated have a combined market cap of \$20.2 million<sup>12</sup>, over 1 million successful transactions, and 44.9 thousand unique senders.

### 5.1 Tracing attacks on SNIP-20 Transfers

Before getting to transaction simulation attacks, we start by simply adding logging statements to the untrusted software with to observe the storage access patterns. This is enough to invalidate the Receiver Privacy guarantees of the SNIP-20 tokens. In particular, the receiver and the amount are both intended to be confidential.

The combination of encrypted transaction requests and encrypted storage is the basis for expecting receiver privacy and value privacy for transactions. As discussed in Section 4.2, however, users' balances are maintained by a key-value store and the encrypted key for each account remains the same. By logging the keys fetched from the storage, we can infer the receiver of a SNIP-20 transfer.

We modified the untrusted code, specifically `ocall_read_db`, to simply print the `get` traces of encrypted keys fetched while executing a contract program. For instance, an abridged trace resulting from executing a SNIP-20 transfer is illustrated in Figure 1b, each trace corresponding to a smart contract instruction in Figure 1a. The same sender or receiver in different transfer transactions will always leave a `get` trace of the same encrypted key, e.g., `0x3d4982[...]` in Figure 1b. We carried out experiments on public testnet by sending transactions between our own addresses to confirm our hypothesis.

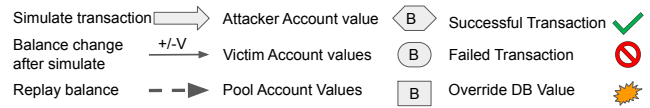
As the above discussion implies, the balances of the parties (sender/receiver) involved in a transaction are accessed using fixed keys for each address. By correlating two transactions, Tx1 and Tx2, such that the sender of Tx1 is the receiver of Tx2, we were able to deanonymize the *supposedly private* receiver of Tx2. Over time, a graph of the transaction history can be built, and used to fully de-anonymized users.

### 5.2 Implementing transaction simulation

We now describe how we modified (the untrusted portion of) the existing Secret Network codebase, to build a forensic analysis node capable of carrying out offline simulation attacks.

<sup>11</sup><https://secretnodes.com/>

<sup>12</sup><https://secretanalytics.xyz/bridge>, <https://www.coingecko.com>



**Figure 4: Legend for transaction simulation schematics. Elements with the same shape have the same key and can be replaced with one another. Elements with the same color have the same representation as encrypted values.**

We focus on adding a wrapper around the `ecall_handle` function described in Section 4.1. Secret's untrusted codebase includes `DeliverTx` and `SimulateTx`, two high level wrappers that call this function. `DeliverTx` is intended to be used after a transaction has been committed in a block — though lacking a Proof of Publication, the enclave cannot ensure this. `SimulateTx` is used to estimate gas costs. While either could be used, we found `SimulateTx` to be the easiest code to modify. We then added the following new RPC calls to support transaction simulation:

- **Fork()**: (Re-)Initializes a new in-memory cache, `SimState`, for persisting state changes between transaction simulations.
- **Simulate(Tx) → [ck=cv, ...]**: Simulates one transaction, modifying only `SimState`. During contract execution, interactions with key-value storage (through `ocall_get`, `ocall_set`) are diverted to `SimState`. Returns a transcript `[ck=cv, ...]` resulting from interacting with key-value storage.
- **Replay (ck=cv, ...)**: Replays previously stored mappings from encrypted keys `ck` to encrypted values `cv`. Since the empty string is a default value, this can be replayed for any key regardless of whether it has been previously observed.

### 5.3 Inferring SNIP-20 Transfer Amounts

As our first transaction simulation attack, we can infer the amount of tokens transferred in a SNIP-20 transaction. While access pattern leakage from Sec. 5 enabled us to determine the receiver's address, the number of tokens transferred remains encrypted.

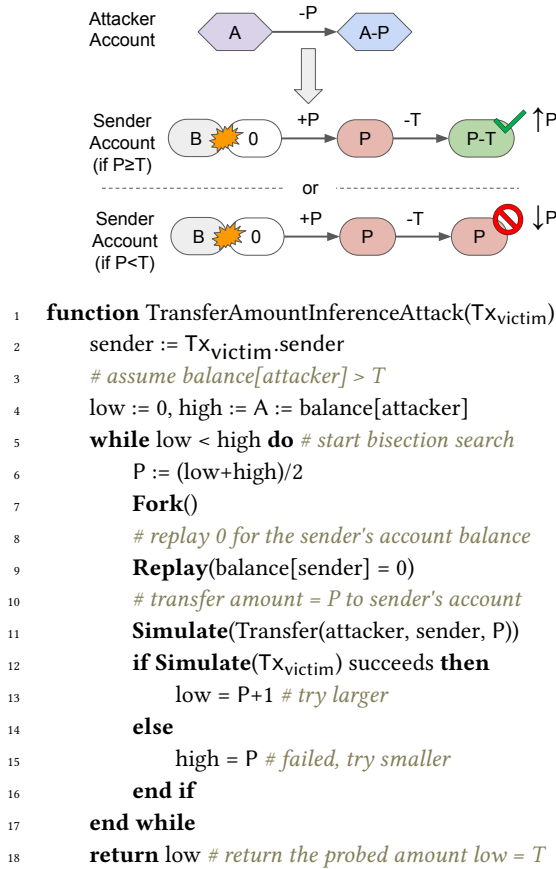
Assume that our goal is to infer the transfer amount  $T$  of a victim transaction  $Tx_{victim}$ . For now, let us assume the attacker has an account with balance  $A$  that we know is greater than  $T$ . We'll design a simulation attack that probes the victim transaction by constructing a comparison oracle, i.e., for each choice of a probe value  $P$ , we learn whether  $T < P$ .

In Fig. 5 we give a schematic for our approach along with a pseudocode description. The legend in Fig. 4 should help for following the schematic. We start by **Replaying** `balance[sender]=0`, where the encrypted field name corresponding to `balance[sender]` can be inferred from the access pattern analysis. Next we generate and **Simulate** the execution of a transaction that transfers the probe amount  $P$  from the attacker's balance to the sender's balance. Finally, we **Simulate** the victim's transaction  $Tx_{victim}$ . As a result of transaction simulation, we learn whether or not the transaction succeeds. If it succeeds, we've learned that  $P \geq T$ , and if it fails then  $P < T$ . We can simply rewind the simulation to the beginning by calling **Fork**, choosing a different probe value via binary search.

### 5.4 Surveillance without having to buy tokens

We previously assumed that the attacker's balance  $A$  at the beginning of the attack was larger than the transaction they sought to





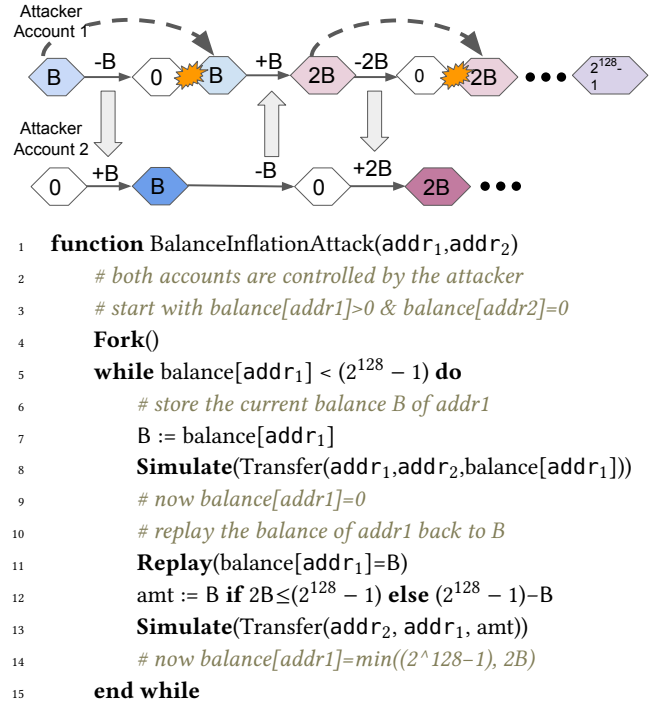
**Figure 5: Schematic and Pseudocode for Transaction Simulation Attack to Infer SNIP-20 Transfer Amount**

analyze, meaning that the attack had large upfront investment requirements. We now explain how to get around this assumption by starting with a very small balance and *simulating the inflation of the attacker's account*. Pseudocode and a schematic are given in Fig. 6. The main idea is that the attacker simulates transferring a balance between two of her own accounts, replenishing each empty account with a replayed value. Since this algorithm doubles the balance of an adversarial account at each step, it takes 128 iterations to reach the maximum representable value. This only needs to be carried out once per token type since the resulting encrypted value can be stored for later use.

Note that this inflation is only occurring within the attacker's local simulation and does not in any way imply that the *integrity* of the overall blockchain is at risk. The reason for this is that even if there were no enclaves at all, the integrity of the blockchain would still be ensured by the *consensus* among validator nodes, just as in any transparent blockchain.

### 5.5 Querying SNIP-20 account balances

We now present an attack that allows us to probe the account balance of an arbitrary victim, just given their address. We assume the attacker begins with a maximal account balance of  $2^{128} - 1$  (obtained in Section 5.4). SNIP-20 tokens throw an exception when



**Figure 6: Schematic and pseudocode for a transaction simulation attack that inflates the attacker's (simulated) balance**

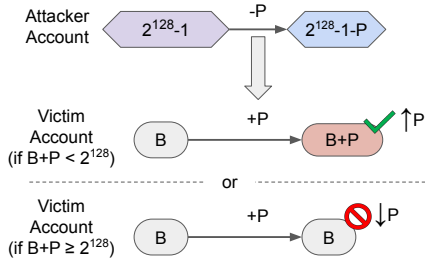
adding two u128 elements that overflow. We use this as our next oracle to probe whether a victim's account with balance  $B$  is less than a parameter  $2^{128} - 1 - P$ . Following along with the illustration in Fig. 5, we simulate a transaction that transfers  $P$  from the attacker's account to the victim's account. This transaction succeeds if  $B + P < 2^{128}$ , and fails otherwise. Using the **Replay** capability and a binary search we can recover the exact value after 128 iterations.

### 5.6 Forensic analysis of past SNIP-20 transfers

For reasons involving details of Secret Network's implementation, it is difficult to obtain a snapshot of the blockchain state from very far back. State snapshots are cycled on a rolling basis, and while archive nodes provide most old blocks, we found we were unable to sync a node by replaying blocks from "genesis." So, how would we break receiver privacy for transactions prior to the last available state checkpoint?

The first challenge is that Secret Network, like most blockchains, implements a form of replay prevention based on "sequence numbers." Transactions are rejected unless they have a sequence number that matches a counter associated with each account. The counter for an account is incremented each time a transaction from that account is committed. However, this mechanism is implemented only within the untrusted codebase, so we simply disable the checks on the sequence number of a transaction before `call_handle`.

Next, the sender's balance at the snapshot time may be lower than it was when the past transaction of interest was committed. However, the techniques we described earlier enable us to control the sender's simulated balance, as we make use of when inferring



```

1 function BalanceInferenceAttack(victim)
2   # begin with inflated balance[attacker]=2^128-1
3   low = 0, high = 2^128 - 1
4   while low < high do
5     P := (low+high)/2
6     Fork()
7     if Simulate(Transfer(attacker, victim, P))
8       succeeds then
9         low = P+1 # try larger
10    else
11      high = P # failed, try smaller
12    end if
13  end while
14  return (2^128 - 1) - low # return the probed balance[victim]

```

Figure 7: Schematic and Pseudocode for Transaction Simulation Attack to Probe the Balance of Arbitrary Address

the transaction amounts. Therefore, forensic analysis of prior transactions does not require syncing a node to old application state.

## 6 MEV ON TEE-BASED NETWORKS

Though the privacy of SNIP-20 tokens is the most important issue for Secret Network, this is not the only application at risk. We also explain how transaction simulation attacks can undermine defenses decentralized exchanges (DEXes) built on top of TEE-backed platforms against front-running.

### 6.1 Background on MEV

**MEV and Frontrunning attacks.** Any operations that clients want to be executed on the blockchain must be committed in a block by the validator nodes running a consensus protocol. Since the validators must collect pending transactions before assembling them into a block, they can insert their own transactions (such as when *frontrunning*) and reorder the pending ones in whatever order they see fit. In blockchains, the notion of validators benefiting economically from the ability to determine transaction order is called *miner extractable value* (MEV) [71].

**MEV in AMMs.** Automated Market Maker (AMM) is one of the most popular DEXes. Most well-known AMMs, e.g., Uniswap [36], are constant product AMM. An AMM is meant to help trade any number of token pairs, which we will generalize to calling *tokenA* and *tokenB*. Let *PoolAmtA* and *PoolAmtB* represent the total amount of tokens in the liquidity pool for this token pair (tokens are provided by liquidity providers who make profits by charging transaction fees). When trading one token for another, the price is

```

1 contract function UniswapTrade(amtA, slippage_limit)
2   amtB = calcB(PoolAmtA, PoolAmtB, amtA)
3   if amtB < slippage_limit then
4     # no state change but the transaction is committed.
5   else
6     # changes to caller account balances as well as the
7     # contract's variables are committed to chain.
8     PoolAmtB -= amtB
9     PoolAmtA += amtA
10  end if
11 contract function calcB(PoolAmtA, PoolAmtB, amtA)
12   amtB = PoolAmtB * PoolAmtA / (PoolAmtA + amtA)

```

Figure 8: Contract Pseudocode for Uniswap Trade.

determined as shown line 10 in Figure 8, which keeps the product *PoolAmtA\*PoolAmtB* invariant. The *amtA* refers to the amount of *tokenA* the client wants to trade and *amtB* is the amount of *tokenB* they will receive back.

Upon seeing that a client wants to buy *tokenB* using *tokenA*, the miner generates a transaction buying *tokenB*, ahead of the client, artificially inflating the demand for *tokenB*. This means the client would end up getting less *tokenB* with the same amount of *tokenA*. Subsequently, the price of *tokenB* would inflate even more and the adversarial miner could now “sell” *tokenB* to get more of *tokenA* than it originally spent buying it. Thus, the miner benefits from sandwiching the client transaction. For a more detailed discussion of MEV and the other perils to the blockchain introduced by it, see [19].

Note that to know the frontrun amount, the adversarial miner needs to see the amounts in the client’s transaction. This is because if the miner increases *PoolAmtA* (raises the price to buy *tokenB*) too little it will not profit, and if it increases it too much, the *slippage\_limit* (the minimum amount of *tokenB* the client would like to receive) parameter of the client causes the target transaction to fail altogether. Thus, if the miner were unable to access the amounts input by the target transaction caller, they would not be able to mount MEV attacks effectively.

### 6.2 Sandwich attacking a private swap

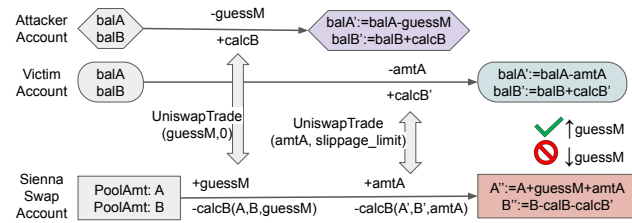
SiennaSwap<sup>13</sup> and SecretSwap<sup>14</sup> have gained significant popularity as AMMs built on Secret. They have cumulated trading volumes of \$1.07B and \$1.26B, support 149 and 184 trading pairs, attract 8.94K and 24.22K users respectively<sup>15</sup>. Although they all claimed that they are front-running resistant as they are based on a privacy-preserving platform, we demonstrate a sandwich attack does exist.

Consider a victim transaction *Trade(amtA, slippage\_limit)*. The value of *amtA* can be inferred from the change in *PoolAmtA* after execution, but *slippage\_limit* here would be secret. The main parameter of a sandwich attack is how large the frontrun transaction should be. The attacker’s profit is maximized when the victim transaction succeeds but just barely within the slippage limit.

<sup>13</sup><https://sienna.network/swap/>

<sup>14</sup><https://secretswap.net/swap>

<sup>15</sup><https://archive.is/A8fTk>, <https://archive.is/NcY9F>



```

1  function FindMaxFrontrunAmt(TXvictim)
2      low := 0, high := balA
3      while low < high do
4          guessM := (low+high)/2
5          Fork( )
6          Simulate(Tradeattacker(guessM, 0))
7          if Simulate(TXvictim) succeeded then
8              low := guessM+1 # try larger
9          else
10             high := guessM # failed, try smaller
11         end if
12     end while
13     return low # the maximal frontrun amount

```

**Figure 9: Schematic and Pseudocode for Transaction Simulation Attack to Determine the Maximal Frontrun Amount**

Optimizing a sandwich attack thus requires determining the victim’s slippage limit, which can be done through a “bisection” search, as shown in Figure 9. This bisection search iteratively reduces the range of possible values of the slippage limit by determining larger minimum and smaller maximum for the value. Once the bisection search ends and the `slippage_limit` is revealed the attacker can frontrun of the transaction on the real network.

Note that the attacker’s ability to front run is limited by their available capital. Thus while the attacker could use simulated balance inflation as described earlier to infer the slippage limit even for larger trades, this would not help them to front run.

### 6.3 Validating our implementation and artifact

There are two main ways we validate our implementation. The first is through the use of container-based local networks, which are used by Secret Network for integration tests. This requires us to compile both the Secret enclave as well as the smart contracts from the sSCRT implementation from source code. This is ideal for reproducing our demonstrations, so we plan to publish an artifact based on this. We benchmark the cost of conducting the sandwich attack in this setting. As the sandwich attack is based on the transaction simulation attack, no transaction is broadcast to the network so there’s no monetary cost to find the optimal profit-making transactions. The primary bottleneck for launching this attack lies in the bisection search process. Each iteration of the bisection search takes about 0.40s. Given the attacker’s constrained ability to execute only a limited number of iterations/simulations during the block time to explore the slippage limit of the victim transaction, there’s a tradeoff between the range and precision of the bisection search. The success of the attack hinges on the target

value falling within the search range; thus, a broader range ensures the target’s inclusion. However, a wider search range sacrifices precision, preventing the attacker from optimally extracting profits. We provide more detailed microbenchmarks in Appendix A and attack accuracy analysis in Appendix B.

The second is through the use of Secret’s public test network, `pulsar-2`. The test network runs an enclave binary that was signed by the developers, so it is ideal for demonstrating that the vulnerabilities we rely on were not introduced by our own modifications. The production versions of Secret Swap and Sienna Swap also provide development versions on the test network.

## 7 MITIGATIONS

### 7.1 Coordinated vulnerability disclosure and mitigating retroactive SNIP-20 privacy leaks

We discussed our preliminary findings with developers from all four projects. As only Secret Network had already launched their production network, so we focused on coordinating with them on a public disclosure and mitigations. These vulnerabilities are more easily exploited than those related to AepicLeak [77], as they do not require a special hardware configuration to exploit - even cloud tenants without hypervisor privileges or physical could carry them out. Despite the much larger attack surface, several principles for deploying mitigations are the same.

The most immediate intervention developers could take, as they did for AepicLeak, is to disable remote attestation by revoking the API key associated with their IAS account. This is a “registration freeze” that prevents new nodes from joining the network. Once developers have updated their software with mitigations (e.g., proofs of publication and storage replay defense), they can reenact registration by generating a new IAS API key. In this case, developers instead deployed a proof-of-publication defense against transaction replay attacks. We coordinated our public disclosure to follow this upgrade. Regardless, existing nodes that had already registered on the network prior to the intervention are still able to carry out these attacks at any time in the future.

### 7.2 Preventing future SNIP-20 leaks

As mentioned, in response to our disclosure, Secret Network has implemented proofs-of-publication to mitigate transaction replay attacks [66]. Essentially, the updated mechanism incorporates a Tendermint verifier in the trusted enclave code. However, it is important to note that the current implementation is not yet complete as it lacks database rollback protection. Specifically, values loaded from the database are not verified against the proof-of-publication commitment, such as the state Merkle root, published in the latest block. As a result, it is still possible to rewind the state to an earlier version, either entirely or partially. Exploring the privacy implications of this incomplete implementation of proofs-of-publication remains an interesting open problem for future work.

Additionally, developers have taken initial steps towards mitigating access pattern leakage. Every method of SNIP-20 tokens that accesses an account balance is now extended to support additional “decoy” parameters, inspired by Monero [54], as we describe below.

The smart contract, called SNIP-25, now accesses the actual account as well as the decoys in sorted order, thus obfuscating which is the real one.<sup>16</sup> Although this is supported at the smart contract level, there is not yet guidance on how to choose these decoys in an appropriate way.

**Access pattern decoys.** Here, we propose and analyze a straw-man strategy for adding privacy to SNIP-20 tokens, using decoys à la Monero [54]. To implement this decoy strategy, the token contract maintains a list  $L$  of all addresses that interact with the contract. Prior to creating a transaction, a client queries the contract to get  $L$ . The client samples<sup>17</sup> an array  $D$  of  $k$  random addresses from  $L$  to use as decoys.  $D$  included in the encrypted parameters to the `Transfer` function. This protocol requires the use of real addresses for the decoys rather than random strings due to the fact that if an accessed key does not exist, an error is thrown in full view of the untrusted host OS, revealing that it was a decoy access.

As of the SNIP-25 update, the token transfer contract generates a seed by hashing the transaction, uses this to deterministically sample a random number  $0 < r < k$ . To update the receiver balance the contract inserts the real receiver address at position  $r$  in  $D$ . The contract then reads and all of the addresses in  $D$  and only updates the value of the balance of the address at position  $r$ . The contract re-randomizes and writes back the balances of the decoy addresses.

We provide pseudocode for a SNIP-25-like transfer method that makes several decoy accesses along with the actual access in Fig. 10. The developers acknowledge that this is not a comprehensive solution as it relies on choosing decoy addresses effectively<sup>18</sup>. There is a minimum  $1/len(X)$  chance that the attacker can guess the real receiver's address – a criticism that extends to Monero [54].

The private cryptocurrency Monero also employs this decoy strategy [52]. Several attacks described on Monero would also apply to this proposed strategy and can be used to raise the attacker's chances. For example, if the decoys are chosen uniformly at random, more frequently used receiver addresses, such as a token pool contract, have a higher chance of being the real address.

An attacker can also create a large amount of addresses with minimal balance to poison the list of potential decoy addresses [57]. This would require some gas fees but could allow them to eliminate some decoys. As of writing this paper there are 36k unique addresses interacting with the most popular token contract on Secret Network (sSCRT). An attacker could add 10k more addresses, subsequently eliminating about 20% of the decoy addresses on a given transaction, paying only 175 SCRT in transaction fees or \$52.71.

The privacy offered by this scheme is akin to the notion of  $k$ -anonymity [67] where  $k$  is chosen as denoted above. The effectiveness of this protocol depends on the number of decoys, but more decoys increases the transaction fees, leading to a dilemma.

**Block Based Paging.** Next, we discuss the effectiveness of a block-based paging strategy that is a simplification of the implementations used in Obscuro and Phala. The exact leakage from the

access patterns of these schemes depends on several implementation factors but the following construction provides intuition.

Suppose, token contracts store their users' account balances in 4KB blocks. In order to access the account balance the entire block must be loaded into enclave memory. Depending on the storage space required to store the data associated with an account, a fixed number of accounts fits in a 4KB block. Even assuming the mapping of addresses to block locations is fully private, some data is leaked by each access. Essentially, this simple block based strategy means that the same block (and corresponding set of addresses) are loaded each time a particular address is accessed. This is essentially  $k$ -anonymity with the same anonymity set, for each time an address interacts with a contract. The size of the  $k$ -anonymity set can be increased through policies such as using larger blocks.

**ORAM.** Oblivious RAM (ORAM), originally proposed by Goldreich and Ostrovsky [24, 25, 56], is a functionality for hiding storage access patterns for an outsourced, encrypted database. In the ORAM model, a trusted client can temporarily hold and operate on a subset of a vast database, while primarily delegating the storage responsibilities to an untrusted server. ORAM solutions essentially separate the logical access pattern of a program from the physical access pattern, hiding the locations being accessed. In blockchains such as Secret Network, the SGX enclave is the client which outsources most of its state on an untrusted host. Thus, if an ORAM solution were to be deployed for SNIP-20 contracts, the untrusted host would not be able to distinguish between two different `Transact` executions based on the access pattern. While Secret Network has expressed an interest in ORAM as a future strategy, at the moment there is "no clear path forward".<sup>18</sup>

One way to integrate ORAM into SNIP-20 contracts is to create a wrapped version of the `get/set` instructions to obliviously read/write. This seamlessly benefits existing contracts without modification. However, all ORAM solutions add overhead, with a memory of  $N$  blocks requiring the client to fetch  $O(\log(N))$  values [78] per access. Since not all storage accesses are sensitive and require privacy, applying ORAM everywhere is potentially inefficient. Another challenge of merging existing ORAM constructions is that Secret Network (and others) relies on off-chain queries for faster/cheaper read operations, such as checking account balances. Existing ORAM models ensure privacy by assuming sequential reads and writes, resulting in permanent alterations to the untrusted database post each access. Consequently, users would have to submit balance check requests on-chain as transactions, greatly impacting usability.

Implementing ORAM at the application level is an alternative. Insights from previous research on oblivious programming, which recommends breaking down an application into smaller ORAMs based on the preferred leakage model [40], could be instructive. In principle, smart contracts could serve as the trusted ORAM client, utilizing the current key-value interface as the untrusted ORAM server. However, this approach could substantially increase on-chain transaction costs relative to an integrated solution. A subsequent study<sup>19</sup> utilizing off-chain queries to perform ORAM on an individual node and later posting the results on-chain may offer a cost-effective alternative.

<sup>16</sup><https://github.com/scrtlabs/snip20-reference-impl/blob/92ff4bd5/README.md#snip-25-security-update>

<sup>17</sup>Ideally the subset  $D$  is selected uniformly at random, but the distribution from which the client samples is up to the client and trust assumptions based on it are brittle, as we discuss later.

<sup>18</sup><https://github.com/scrtlabs/snip20-reference-impl/tree/ea9fb0#future-work>

<sup>19</sup>[https://medium.com/@dsl\\_uiuc/tee-rollups-fixing-access-patterns-in-tee-based-smart-contracts-with-off-chain-computing-b2f3acfe335f](https://medium.com/@dsl_uiuc/tee-rollups-fixing-access-patterns-in-tee-based-smart-contracts-with-off-chain-computing-b2f3acfe335f)

```

1  contract function send(address[] decoys,
2      address receiver, uint amount)
3      seed := Hash(decoys || receiver || amount)
4      r := RandInt(0, len(decoys), seed)
5      combined := decoys
6      combined.insert(r, receiver)
7      for each address a in combined do
8          balances[a] := balances[a] + (amount * (a == receiver))
9      end for

```

**Figure 10: A decoy-based mitigation for access pattern leakage inspired by Monero**

### 7.3 Ethical principles

Beyond coordinating our vulnerability disclosure with developers, our main concern has been to avoid contributing to the data breach hazard created by this vulnerability. We did not collect a dataset or carry out a measurement study by probing balances and transactions on the production network.

Previous blockchain deanonymization and tracing attacks [31, 46, 52] have only required the public dataset as input, i.e., the blocks and transactions, which are widely replicated. They publish their tracing analysis concept along with a measurement study. While the Secret blockchain itself is a public dataset, executing transactions requires a sealed copy of the master key, which can only be obtained through remote attestation. Thus while we could have retained the ability to carry out measurements, having our authority "grandfathered in," other researchers are unable to obtain this authority for themselves.

We have reset our authority to baseline by erasing our node's sealed key, such that we have to upgrade and re-register our node like anyone else. However, as part of our public disclosure, we plan to provide an "access pattern dashboard" service that displays the access pattern logs that remain leaked to the untrusted host by an ordinary node post-mitigation.

## 8 RELATED WORK

**Security issues in TEE-based smart contracts.** Our work is most closely related to Li et al. [38] in aiming to systematize privacy hazards in TEE-based smart contracts. Whereas they consider frameworks including research proposals, we focus on production systems and the specific design choices they make. While they point out potential hazards, our goal is to demonstrate how these hazards translate into real world application failures. Previous real world attacks have also been demonstrated, again using Secret Network as the concrete example due to its in-use applications. Specifically Van Schaik et al. [77] showed that Secret Network was vulnerable to AepicLeak [7] (the most recent in a long line of SGX vulnerabilities [11, 73, 76]). Although TEE breaks threaten any system that relies on it, Phala and Oasis limit access to the key material to nodes who are trusted or at least have committed financial stakes; in contrast, Secret Network, as well as Obscuro, allow anyone with a compatible processor to register and obtain the ability to execute contract transactions in an enclave. Our attacks are different in that they do not involve a failure of the TEE itself, but rather the integration of the TEEs within a blockchain architecture.

**Smart contract security and privacy.** While smart contracts have enabled many innovative decentralized applications, their history could easily be characterized by the many high-profile hacks and losses of funds that have taken place over the years. Consequently, much research effort has been devoted to surveying these risks [3, 58] and creating tools to reduce them [41, 51, 72, 81]. At the same time, researchers have begun to explore the intersection of private computation and public blockchains. For example, Zether [12] proposes an Ethereum-compatible confidential payments contract, Hawk [34] creates a framework for private smart contract development, and Zexe [8] extends this idea to also obfuscate which functions are being executed. More recently, Aztec [35] with a hybrid zk-rollup, and Zama [28] with FHE, are both planning devnet deployments for private smart contracts. Relative to these works, however, TEE-backed chains offer high performance along with the features developers expect from transparent smart contracts.

**Secure systems based on TEEs.** The need for defense against replay and rollback attacks is well understood in the security literature. Systems such as Rote [42] and Microsoft Confidential Consortium Framework [62] create a distributed network of TEEs such that an honest subset remains online. Since today's TEEs cannot offer reliable non-volatile monotonic counters, this network implements a monotonic counter and keeps track of the most recent version of the sealed data output by the enclave and signs the input along with a hash of the sealed data. On restart, the TEE must verify that signature, ensuring that it can only execute one input using that sealed data. Narrator [53] similarly uses a distributed system of TEEs but application enclaves send a state update message to the system before revealing the output of the computation. Our work reinforces these efforts by demonstrating the consequences of foregoing these defenses.

Besides the blockchain itself, TEEs have also been proposed for blockchain-adjacent services; here too rollback prevention is a central focus. Bite [43] implements a privacy preserving light client for Bitcoin that can receive payments and construct transactions requests without running a full Bitcoin node. It uses ORAM to hide data access patterns and Rote [42] to prevent rollback attacks. TEEChain [39] uses TEEs to create a private payment channel system and requires hardware monotonic counters to protect against rollback attacks. Tesseract [5] implements a private real-time cryptocurrency exchange using TEEs and requires an honest subset of nodes participating in the protocol to be online to prevent rollbacks.

## 9 CONCLUSION

The usage of TEEs is a promising way to realize practical, performant, and private general computation on blockchains. While it is well-known that TEE compromises are a possibility, this is just one of the attack surfaces to contend with. Previous discussions on replay and access pattern hazards have been somewhat abstract, making them easy to underestimate when transitioning research to practice. In this work we bring clarity to this issue by demonstrating the first replay and access pattern attacks on in-use TEE-based smart contract systems. The impact of our vulnerability disclosure has already led to rapid deployment of mitigations for Secret Network's in-use SNIP-20 tokens, as well as to influence the development plans of other mainnet launches.



## ACKNOWLEDGMENTS

This work was funded in part by the National Science Foundation under grants #1943499 and #2112726, and by IC3 industry partners.

## REFERENCES

- [1] 2017. Technology preview: Private contact discovery for Signal. <https://signal.org/blog/private-contact-discovery/>.
- [2] Ittai Anati, Shay Gueron, Simon Johnson, and Vincent Scarlata. 2013. Innovative technology for CPU based attestation and sealing. In *Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy*, Vol. 13. Intel, ACM New York, NY, USA, 7 pages.
- [3] Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. 2017. A survey of attacks on ethereum smart contracts (sok). In *Principles of Security and Trust: 6th International Conference, POST 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings 6*. Springer, 164–186.
- [4] Pierre-Louis Aublin, Florian Kelbert, Dan O’Keeffe, Divya Muthukumaran, Christian Priebe, Joshua Lind, Robert Krahn, Christof Fetzter, David Eyers, and Peter Pietzuch. 2018. LibSEAL: Revealing Service Integrity Violations Using Trusted Execution. In *Proceedings of the Thirteenth EuroSys Conference (Porto, Portugal) (EuroSys ’18)*. Association for Computing Machinery, New York, NY, USA, Article 24, 15 pages. <https://doi.org/10.1145/3190508.3190547>
- [5] Iddo Bentov, Yan Ji, Fan Zhang, Lorenz Breidenbach, Philip Daian, and Ari Juels. 2019. Tesseract: Real-Time Cryptocurrency Exchange Using Trusted Hardware. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (London, United Kingdom) (CCS ’19)*. Association for Computing Machinery, New York, NY, USA, 1521–1538. <https://doi.org/10.1145/3319535.3363221>
- [6] Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A. Kroll, and Edward W. Felten. 2015. SoK: Research Perspectives and Challenges for Bitcoin and Cryptocurrencies. In *2015 IEEE Symposium on Security and Privacy*. IEEE, San Jose, CA, USA, 104–121. <https://doi.org/10.1109/SP.2015.14>
- [7] Pietro Borrello, Andreas Kogler, Martin Schwarzl, Moritz Lipp, Daniel Gruss, and Michael Schwarz. 2022. {ÆPIC} Leak: Architecturally Leaking Uninitialized Data from the Microarchitecture. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, BOSTON, MA, USA, 3917–3934.
- [8] Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. 2020. Zeke: Enabling decentralized private computation. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 947–964.
- [9] Marcus Brandenburger, Christian Cachin, Rüdiger Kapitza, and Alessandro Sorniotti. 2018. Blockchain and Trusted Computing: Problems, Pitfalls, and a Solution for Hyperledger Fabric. <https://doi.org/10.48550/ARXIV.1805.08541>
- [10] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiaainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. 2017. Software Grand Exposure: SGX Cache Attacks Are Practical.. In *WOOT*. USENIX Association, VANCOUVER, BC, CANADA, 11–11.
- [11] Jo Van Bulck, Marina Minkin, Ofir Weiss, Daniel Genkin, Baris Kasicki, Frank Piessens, Mark Silberstein, Thomas F. Wernisch, Yuval Yarom, and Raouk Strackx. 2018. Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution. In *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association, Baltimore, MD, 991–1008. <https://www.usenix.org/conference/usenixsecurity18/presentation/bulck>
- [12] Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. 2020. Zether: Towards privacy in a smart contract world. In *Financial Cryptography and Data Security: 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10–14, 2020 Revised Selected Papers*. Springer, 423–443.
- [13] James Carlyle, Tudor Malene, Cais Manai, Neal Shah, Gavin Thomas, and Roger Willi. 2021. Obscuro White Paper. <https://whitepaper.obscuro.ro/assets/images/obscuro-whitepaper-0-10-0.pdf>
- [14] Raymond Cheng, Fan Zhang, Jernej Kos, Warren He, Nicholas Hynes, Noah Johnson, Ari Juels, Andrew Miller, and Dawn Song. 2019. Ekiden: A Platform for Confidentiality-Preserving, Trustworthy, and Performant Smart Contracts. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, Stockholm, Sweden, 185–200. <https://doi.org/10.1109/EuroSP.2019.00023>
- [15] Chitchanok Chuengsatiansup, Daniel Genkin, Yuval Yarom, and Zhiyuan Zhang. 2022. Side-channeling the Kalyna key expansion. In *Topics in Cryptology—CT-RSA 2022: Cryptographers’ Track at the RSA Conference 2022, Virtual Event, March 1–2, 2022, Proceedings*. Springer, Springer, San Francisco, CA, USA, 272–296.
- [16] Jenny Cook. 2021. Introducing EdgelessDB: A Database Designed for Confidential Computing. <https://techcommunity.microsoft.com/t5/azure-confidential-computing/introducing-edgelessdb-a-database-designed-for-confidential-ba-p/2813631>
- [17] Manuel Costa, Lawrence Esswood, Olga Ohrimenko, Felix Schuster, and Sameer Wagh. 2017. The Pyramid Scheme: Oblivious RAM for Trusted Processors. <https://doi.org/10.48550/ARXIV.1712.07882>
- [18] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. *Cryptology ePrint Archive*, Paper 2016/086. <https://eprint.iacr.org/2016/086> <https://eprint.iacr.org/2016/086>
- [19] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. 2020. Flash Boys 2.0: Frontrunning in Decentralized Exchanges, Miner Extractable Value, and Consensus Instability. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, San Francisco, CA, USA, 910–927. <https://doi.org/10.1109/SP40000.2020.00040>
- [20] Fergus Dall, Gabrielle De Micheli, Thomas Eisenbarth, Daniel Genkin, Nadia Heninger, Ahmad Moghimi, and Yuval Yarom. 2018. CacheQuote: Efficiently Recovering Long-term Secrets of SGX EPID via Cache Attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2018, 2 (May 2018), 171–191. <https://doi.org/10.13154/tches.v2018.i2.171-191>
- [21] Dmitry Evtvushkin, Ryan Riley, Nael CSE Abu-Ghazaleh, ECE, and Dmitry Ponomarev. 2018. BranchScope: A New Side-Channel Attack on Directional Branch Predictor. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems (Williamsburg, VA, USA) (ASPLOS ’18)*. Association for Computing Machinery, New York, NY, USA, 693–707. <https://doi.org/10.1145/3173162.3173204>
- [22] Leor Fishman. 2020. Secret Network documentation: Theoretical Attacks. <https://github.com/scrtlabs/SecretNetwork/blob/61cfc4/docs/protocol/encryption-specs.md>
- [23] Ethereum Foundation. 2023. Ethereum: Blockchain App Platform. <https://ethereum.org/>. Accessed: 2023-01-22.
- [24] Oded Goldreich. 1987. Towards a theory of software protection and simulation by oblivious RAMs. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*. 182–194.
- [25] Oded Goldreich and Rafail Ostrovsky. 1996. Software protection and simulation by oblivious RAMs. *Journal of the ACM (JACM)* 43, 3 (1996), 431–473.
- [26] Daniel Gruss, Julian Lettner, Felix Schuster, Olya Ohrimenko, Istvan Haller, and Manuel Costa. 2017. Strong and Efficient Cache {Side-Channel} Protection using Hardware Transactional Memory. In *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, Vancouver, BC, Canada, 217–233.
- [27] Marcus Hähnel, Weidong Cui, and Marcus Peinado. 2017. High-Resolution Side Channels for Untrusted Operating Systems. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. USENIX Association, Santa Clara, CA, 299–312. <https://www.usenix.org/conference/atc17/technical-sessions/presentation/hahnel>
- [28] Rand Hindi. 2023. Private Smart Contracts Using Homomorphic Encryption. <https://www.zama.ai/post/private-smart-contracts-using-homomorphic-encryption>.
- [29] Matthew Hoekstra, Reshma Lal, Pradeep Pappachan, Vinay Phegade, and Juan Del Cuvillo. 2013. Using innovative instructions to create trustworthy software solutions. *HASP@ ISCA 11*, 10.1145 (2013), 2487726–2488370.
- [30] Tianlin Huo, Xiaoni Meng, Wenhao Wang, Chunliang Hao, Pei Zhao, Jian Zhai, and Mingshu Li. 2019. Bluetheater: A 2-level Directional Privector Based Side-Channel Attack against SGX. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2020, 1 (Nov. 2019), 321–347. <https://doi.org/10.13154/tches.v2020.i1.321-347>
- [31] George Kappos, Haaron Yousaf, Rainer Stütz, Sofia Rollet, Bernhard Haslhofer, and Sarah Meiklejohn. 2022. How to Peel a Million: Validating and Expanding Bitcoin Clusters. In *31st USENIX Security Symposium (USENIX Security 22)*. Usenix Association, Boston, MA, 2207–2223.
- [32] Gabriel Kapthchuk, Ian Miers, and Matthew Green. 2017. Giving State to the Stateless: Augmenting Trustworthy Computation with Ledgers. *Cryptology ePrint Archive*, Paper 2017/201. <https://eprint.iacr.org/2017/201> <https://eprint.iacr.org/2017/201>
- [33] koe. 2022. Mechanics of MobileCoin. <https://mobilecoin.com/files-uploads/2022/09/Mechanics-of-MobileCoin-v0-0-39-preview-10-11.pdf>.
- [34] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamantou. 2016. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *2016 IEEE symposium on security and privacy (SP)*. IEEE, 839–858.
- [35] Aztec Labs. 2023. Aztec: the Hybrid zkRollup. <https://medium.com/aztec-protocol/aztec-the-hybrid-zkrollup-a90a197bf22e>.
- [36] Uniswap Labs. 2023. The Uniswap Protocol. <https://docs.uniswap.org/concepts/uniswap-protocol>. Accessed: 2023-01-23.
- [37] Sangho Lee, Ming-Wei Shih, Prasun Gera, Taesoo Kim, Hyesoon Kim, and Marcus Peinado. 2017. Inferring Fine-grained Control Flow Inside SGX Enclaves with Branch Shadowing. In *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, Vancouver, BC, 557–574. <https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/lee-sangho>
- [38] Rujia Li, Qin Wang, Qi Wang, David Galindo, and Mark Ryan. 2022. SoK: TEE-Assisted Confidential Smart Contract. *Proceedings on Privacy Enhancing Technologies* 3 (2022), 711–731.
- [39] Joshua Lind, Oded Naor, Ittay Eyal, Florian Kelbert, Emin Gün Sirer, and Peter Pietzuch. 2019. Teechain: A Secure Payment Network with Asynchronous Blockchain Access. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles (Huntsville, Ontario, Canada) (SOSP ’19)*. Association for Computing Machinery,



- New York, NY, USA, 63–79. <https://doi.org/10.1145/3341301.3359627>
- [40] Chang Liu, Xiao Shaun Wang, Kartik Nayak, Yan Huang, and Elaine Shi. 2015. OblivM: A Programming Framework for Secure Computation. In *2015 IEEE Symposium on Security and Privacy*. IEEE, San Jose, CA, USA, 359–376. <https://doi.org/10.1109/SP.2015.29>
- [41] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. 2016. Making smart contracts smarter. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 254–269.
- [42] Sinisa Matetic, Mansoor Ahmed, Kari Kostiaainen, Aritra Dhar, David Sommer, Arthur Gervais, Ari Juels, and Srđjan Capkun. 2017. ROTE: Rollback Protection for Trusted Execution. In *Proceedings of the 26th USENIX Conference on Security Symposium (Vancouver, BC, Canada) (SEC'17)*. USENIX Association, USA, 1289–1306.
- [43] Sinisa Matetic, Karl Wüst, Moritz Schneider, Kari Kostiaainen, Ghassan Karame, and Srđjan Capkun. 2019. BITE: Bitcoin Lightweight Client Privacy Using Trusted Execution. In *Proceedings of the 28th USENIX Conference on Security Symposium (Santa Clara, CA, USA) (SEC'19)*. USENIX Association, USA, 783–800.
- [44] Patrick McCorry, Ethan Heilman, and Andrew Miller. 2017. Atomically Trading with Roger: Gambling on the Success of a Hardfork. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, Joaquín García-Alfaro, Guillermo Navarro-Arribas, Hannes Hartenstein, and Jordi Herrera-Joancomartí (Eds.). Springer International Publishing, Cham, 334–353.
- [45] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V. Rozas, Hisham Shafi, Vedvyas Shanhogue, and Uday R. Savagaonkar. 2013. Innovative Instructions and Software Model for Isolated Execution. In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy (Tel-Aviv, Israel) (HASP'13)*. Association for Computing Machinery, New York, NY, USA, Article 10, 1 pages. <https://doi.org/10.1145/2487726.2488368>
- [46] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M. Voelker, and Stefan Savage. 2016. A Fistful of Bitcoins: Characterizing Payments among Men with No Names. *Commun. ACM* 59, 4 (mar 2016), 86–93. <https://doi.org/10.1145/2896384>
- [47] Pratyush Mishra, Rishabh Poddar, Jerry Chen, Alessandro Chiesa, and Raluca Ada Popa. 2018. Obliv: An Efficient Oblivious Search Index. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, San Francisco, CA, USA, 279–296. <https://doi.org/10.1109/SP.2018.00045>
- [48] Ahmad Moghimi, Gorka Irazoqui, and Thomas Eisenbarth. 2017. CacheZoom: How SGX Amplifies The Power of Cache Attacks. *CoRR* abs/1703.06986 (2017), 69–90. [arXiv:1703.06986](https://arxiv.org/abs/1703.06986) <https://arxiv.org/abs/1703.06986>
- [49] Ahmad Moghimi, Jan Wichelmann, Thomas Eisenbarth, and Berk Sunar. 2017. MemJam: A False Dependency Attack Against Constant-Time Crypto Implementations. *International Journal of Parallel Programming* 47 (2017), 538–570.
- [50] Malte Möser and Rainer Böhme. 2017. The price of anonymity: empirical evidence from a market for Bitcoin anonymization. *Journal of Cybersecurity* 3, 2 (2017), 127–135.
- [51] Bernhard Mueller. 2018. Smashing ethereum smart contracts for fun and real profit. *HITB SECONF Amsterdam 9* (2018), 54.
- [52] Malte Möser, Kyle Soska, Ethan Heilman, Kevin Lee, Henry Heffan, Shashvat Srivastava, Kyle Hogan, Jason Hennessey, Andrew Miller, Arvind Narayanan, and Nicolas Christin. 2017. An Empirical Analysis of Traceability in the Monero Blockchain. <https://doi.org/10.48550/ARXIV.1704.04299>
- [53] Jianyu Niu, Wei Peng, Xiaokuan Zhang, and Yinqian Zhang. 2022. NARRATOR: Secure and Practical State Continuity for Trusted Execution in the Cloud. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (Los Angeles, CA, USA) (CCS '22)*. Association for Computing Machinery, New York, NY, USA, 2385–2399. <https://doi.org/10.1145/3548606.3560620>
- [54] Shen Noether, Adam Mackenzie, et al. 2016. Ring confidential transactions. *Ledger* 1 (2016), 1–18.
- [55] Oasis Labs. 2020. The Oasis Blockchain Platform. Whitepaper. [https://assets.website-files.com/5f59478e350b91447863f593/628ba74a9aee37587419cf65\\_20200623TheOasisBlockchainPlatform.pdf](https://assets.website-files.com/5f59478e350b91447863f593/628ba74a9aee37587419cf65_20200623TheOasisBlockchainPlatform.pdf). Accessed: 2023-01-22.
- [56] Rafail Ostrovsky. 1990. Efficient computation on oblivious RAMs. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*. 514–523.
- [57] João Otávio Chervinski, Diego Kreutz, and Jiangshan Yu. 2021. Analysis of transaction flooding attacks against Monero. In *2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. 1–8. <https://doi.org/10.1109/ICBC51069.2021.9461084>
- [58] Daniel Perez and Benjamin Livshits. 2021. Smart Contract Vulnerabilities: Vulnerable Does Not Imply Exploited. In *USENIX Security Symposium*. 1325–1341.
- [59] Phala. 2020. Phala Wiki. Wiki. <https://wiki.phala.network/en-us/general/phala-network/intro/>. Accessed: 2023-01-22.
- [60] Ashay Rane, Calvin Lin, and Mohit Tiwari. 2015. Raccoon: Closing Digital {Side-Channels} through Obfuscated Execution. In *24th USENIX Security Symposium (USENIX Security 15)*. Usenix Association, Santa Clara, CA, 431–446.
- [61] Fergal Reid and Martin Harrigan. 2013. *An Analysis of Anonymity in the Bitcoin System*. Springer New York, New York, NY, 197–223. [https://doi.org/10.1007/978-1-4614-4139-7\\_10](https://doi.org/10.1007/978-1-4614-4139-7_10)
- [62] Mark Russinovich, Edward Ashton, Christine Avanesians, Miguel Castro, Amaury Chamayou, Sylvan Clebsch, Manuel Costa, Cédric Fournet, Matthew Kerner, Sid Krishna, Julien Maffre, Thomas Moscibroda, Kartik Nayak, Olya Ohrimenko, Felix Schuster, Roy Schwartz, Alex Shamis, Olga Vrousou, and Christoph M. Wintersteiger. 2019. *CCF: A Framework for Building Confidential Verifiable Replicated Services*. Technical Report MSR-TR-2019-16. Microsoft. <https://www.microsoft.com/en-us/research/publication/ccf-a-framework-for-building-confidential-verifiable-replicated-services/>
- [63] Moritz Schneider, Ramya Jayaram Masti, Shweta Shinde, Srđjan Capkun, and Ronald Perez. 2022. SoK: Hardware-supported Trusted Execution Environments. <https://arxiv.org/abs/2205.12742>. <https://doi.org/10.48550/ARXIV.2205.12742>
- [64] Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, and Stefan Mangard. 2017. Malware Guard Extension: Using SGX to Conceal Cache Attacks. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, Michalis Polychronakis and Michael Meier (Eds.). Springer International Publishing, Cham, 3–24.
- [65] scrtlabs. 2023. Snip-20 Reference Implementation. <https://github.com/scrtlabs/snip20-reference-impl/tree/20b0dfd5b1e594e0a23bbb7cc508f7f650accdc7>. Accessed: 2023-02-15.
- [66] Secret Network. 2023. Resolving Newly Discovered Privacy Vulnerabilities with SNIP-20 Tokens. <https://web.archive.org/web/20230614213000/https://scrt.network/blog/resolving-newly-discovered-privacy-vulnerabilities-with-snip-20-tokens/>. Accessed: 2023-06-14.
- [67] Latanya Sweeney. 2002. k-anonymity: A model for protecting privacy. *International journal of uncertainty, fuzziness and knowledge-based systems* 10, 05 (2002), 557–570.
- [68] Interchain Core Teams. 2023. Cosmos SDK Documentation. <https://docs.cosmos.network/v0.47>. Accessed: 2023-02-23.
- [69] Tendermint. 2023. Tendermint Core. <https://tendermint.com/core/>. Accessed: 2023-02-23.
- [70] Afonso Tinoco, Sixiang Gao, and Elaine Shi. 2022. EnigMap: Signal Should Use Oblivious Algorithms for Private Contact Discovery. *Cryptology ePrint Archive, Paper 2022/1083*. <https://eprint.iacr.org/2022/1083> <https://eprint.iacr.org/2022/1083>
- [71] Christof Ferreira Torres, Ramiro Camino, and Radu State. 2021. Frontrunner Jones and the Raiders of the Dark Forest: An Empirical Study of Frontrunning on the Ethereum Blockchain. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, online, 1343–1359. <https://www.usenix.org/conference/usenixsecurity21/presentation/torres>
- [72] Petar Tsankov, Andrei Dan, Dana Drachler-Cohen, Arthur Gervais, Florian Buenzli, and Martin Vechev. 2018. Security: Practical security analysis of smart contracts. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 67–82.
- [73] Jo Van Bulck, Daniel Moghimi, Michael Schwarz, Moritz Lippi, Marina Minkin, Daniel Genkin, Yuval Yarom, Berk Sunar, Daniel Gruss, and Frank Piessens. 2020. LVI: Hijacking Transient Execution through Microarchitectural Load Value Injection. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, San Francisco, CA, USA, 54–72. <https://doi.org/10.1109/SP40000.2020.00089>
- [74] Jo Van Bulck, Frank Piessens, and Raoul Straeckx. 2017. SGX-Step: A Practical Attack Framework for Precise Enclave Execution Control. In *Proceedings of the 2nd Workshop on System Software for Trusted Execution (Shanghai, China) (SysTEX'17)*. Association for Computing Machinery, New York, NY, USA, Article 4, 6 pages. <https://doi.org/10.1145/3152701.3152706>
- [75] Stephan van Schaik, Andrew Kwong, Daniel Genkin, and Yuval Yarom. 2020. SGAXe: How SGX Fails in Practice. <https://sgaxeattack.com/>.
- [76] Stephan van Schaik, Marina Minkin, Andrew Kwong, Daniel Genkin, and Yuval Yarom. 2021. CacheOut: Leaking Data on Intel CPUs via Cache Evictions. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, San Francisco, CA, USA, 339–354. <https://doi.org/10.1109/SP40001.2021.00064>
- [77] Stephan van Schaik, Alex Seto, Thomas Yurek, Adam Batori, Bader AlBassam, Christina Garman, Daniel Genkin, Andrew Miller, Eyal Ronen, and Yuval Yarom. 2022. SoK: SGX.Fail: How Stuff Get eXposed. <https://sgx.fail>.
- [78] Xiao Wang, Hubert Chan, and Elaine Shi. 2015. Circuit oram: On tightness of the goldreich-ostrovsky lower bound. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 850–861.
- [79] Ying Yan, Changzheng Wei, Xuepeng Guo, Xuming Lu, Xiaofu Zheng, Qi Liu, Chenhui Zhou, Xuyang Song, Boran Zhao, Hui Zhang, and Guofei Jiang. 2020. Confidentiality Support over Financial Grade Consortium Blockchain. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (Portland, OR, USA) (SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 2227–2240. <https://doi.org/10.1145/3318464.3386127>
- [80] Hang Yin, Shunfan Zhou, and Jun Jiang. 2022. Phala Network: A Secure Decentralized Cloud Computing Network Based on Polkadot. <https://files.phala.network/phala-paper.pdf>
- [81] Yi Zhou, Deepak Kumar, Surya Bakshi, Joshua Mason, Andrew Miller, and Michael Bailey. 2018. Erays: Reverse Engineering Ethereum’s Opaque Smart Contracts. In

27th USENIX Security Symposium (USENIX Security 18). USENIX Association, Baltimore, MD, 1371–1385. <https://www.usenix.org/conference/usenixsecurity18/presentation/zhou>

- [82] Guy Zyskind, Oz Nathan, and Alex Pentland. 2015. Enigma: Decentralized Computation Platform with Guaranteed Privacy. <https://doi.org/10.48550/ARXIV.1506.03471>

## A COST OF CONDUCTING SANDWICH ATTACKS

We conducted benchmarking to assess the cost of running the sandwich attack (other attacks should have similar costs) on a local private network. The attack involves only offline simulation, which would not be influenced by the network environment. We run a local Secret node in v1.7.0-rc.2, on a machine with an Intel Xeon E-2276G 6-core CPU (3.80GHz) and 64 GB RAM. However, only 1 core is used for the attack. The primary bottleneck for launching the sandwich attack lies in the bisection search process, which progressively reduces the range. Therefore, our focus was on benchmarking the time required for a single iteration. It's important to note that the exact time to conduct each attack depends on the range being searched and the specific application being targeted.

In our experiments, we found that one iteration of the bisection search takes approximately 0.40s. Within each iteration, we begin by restoring the database states, which takes around 0.06s. Additionally, we need to generate and sign a new adversary transaction. Since we made minimal changes to Secret Network's codebase, we first generate a transaction and then modify the transaction nonce by resigning it. This process involves two interactions with the network and takes a total of 0.11s. Subsequently, we simulate the victim and front-running transactions offline, with each simulation costing approximately 0.06s. Furthermore, we need to query the liquidity pool both before and after simulating the victim transaction, with each query taking about 0.05s.

It is important to highlight that our focus was not on optimizing performance, and we made modifications solely to the untrusted code. During our benchmarking we found that 5% of the time to simulate a transaction was spent in the trusted code base (i.e. executing `ecall_handle`, `ecall_init`, and `ecall_query`) and represents code that cannot be optimized. Considering the block time of approximately 6-7 seconds in the Secret network, our unoptimized implementation allows a malicious miner to iterate approximately 17 times before proposing a block. The miner can search within a range of  $x \cdot 2^{17}$ , where  $x$  represents the precision they set for the attack. Notably,  $x$  can exceed the minimal unit of SNIP-20 tokens, enabling the miner to enlarge the search range within this limited time.

## B ANALYSIS OF STRATEGIES FOR CONDUCTING SANDWICH ATTACKS

To assess the accuracy of the MEV sandwich attacks described in Section 6.2, we simulated various potential attacker strategies. These strategies also assume that the attacker can paralyze their execution of bisection sort. In a real scenario paralyzing the search on one transaction would not affect the accuracy of the attack but executing one transaction would affect the price of any transaction after it, therefore attacking two transactions in parallel would affect the accuracy of the attack on the second transaction. Typically, we

can assume users set their transaction's slippage limit based on a fixed percentage of the received token's price.

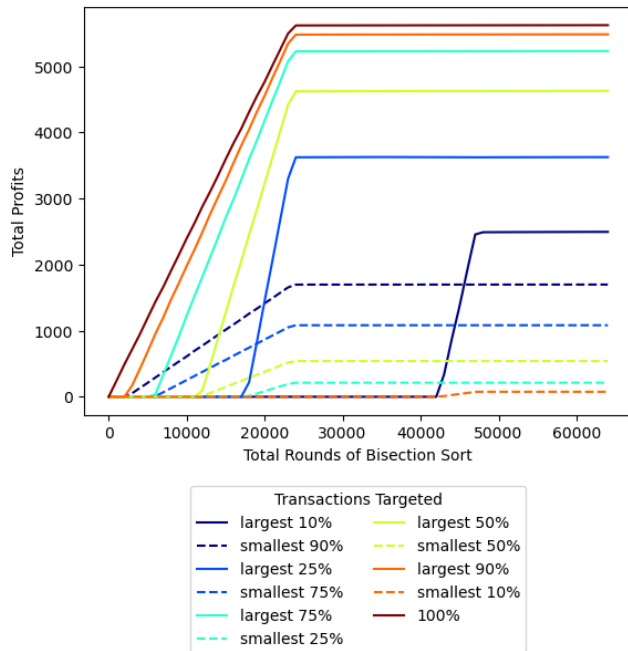
The attacker can use one round of bisection sort to determine if a transaction is in a given range of values. To determine if a transaction is trading above or below  $X$  of a given token (assuming a fixed slippage percentage) the attacker would calculate the amount of the token they would have to buy before the victim's transaction to result in the victim's transaction failing. They would set `guessM` from Figure 9 to this value. If that round of bisection sort fails then that victim's transaction is below the threshold  $X$ . If it succeeded the victim's transaction is above the threshold.

Using this technique the attacker can filter out transactions above or below certain values. One strategy includes spending more rounds of bisection sort being more precise in attacking large "whale" transactions as they would result in larger profit after the attack. But large transactions are less common so another strategy is to target many smaller transactions.

Our simulation code executes by looking at the transaction data and using one round of bisection sort to filter out the top or bottom percentage of transactions. We assume the attacker will aim to spend the same amount of rounds of bisection sort on all transactions in a the desired range. If the optimal frontrun amount for a transaction was determine in less than the allowed rounds the extra rounds are used on a different transactions. In our scenario the attacker had the resources to execute the given number of rounds of bisection sort, and the transaction mempool (i.e. list of pending transactions) always contained transactions to full use these resources.

We analyzed these strategies using a dataset from the Trader Joe decentralized exchange on Avalanche blockchain. This dataset includes transaction data for users trading UCSD for e-WAVAX token. This it is the most prolific pool among the top 40 trading pairs on Sushiswap and Uniswap on the Ethereum Network, and Trader Joe on the Avalanche Network. The data contains 47,069 transactions collected over 5 days. The average token transfer amount was 0.59 with a standard deviation of 1.15. The maximum transfer amount was 48.94. We assume that the users would accept a slippage limit of 80% of the original price of the token. We also assumed the same starting pool amounts for each transaction. While in a real scenario the effect of one transaction would change the pool values we use this as an estimate. The total profits the attacker would receive for strategies targeting the smallest and largest percentage of transactions are shown in Figure 11. Figure 11 shows that the best strategy is to consider as many transactions as possible. This dataset did not have enough difference between the small and large transactions where it would have been advantageous to filter out transactions.

When comparing the strategies that use the largest 10% and smallest 90% and the largest 25% and smallest 75% of transactions targeting the smaller half of transactions results in more profit than the larger half because resources are wasted filtering out the smaller transactions. But targeting the smaller half of transactions results in a lower cap of profit as the largest transactions yielding the most profit are excluded.



**Figure 11: Average profits that an attacker can receive for a number the rounds of bisection sort executed across all transactions from the Trader Joe USCD to e-WAVAX pool 5 day historical dataset**

## C USAGE STATISTICS FOR TEE-BACKED SMART CONTRACT PLATFORMS

In Table 2 we provide some additional statistics about the TEE-backed smart contract platforms we evaluate in this work, so to provide additional context about their real-world usage and interest. While most of these platforms are either prelaunch or have launched recently, we note that there is considerable investment and community interest in developing platforms which use this technology. In order gauge to a rough measure of community interest, we provide both the number of unique addresses that have interacted with the ecosystem and the number of members in each community’s official Discord (collected June 2023).

Table 3 includes more detailed statistics about specific SNIP-20 tokens on Secret Network.

<sup>20</sup><https://web.archive.org/web/20230614000618/https://www.mintscan.io/secret>, <https://web.archive.org/web/20230614000028/https://coinmarketcap.com/currencies/secret/>, <https://www.businesswire.com/news/home/20220119005731/en/Secret-Network-Announces-400-Million-in-Ecosystem-Funding-Alongside-Substantial-New-Investment-from-Leading-Firms>

<sup>21</sup><https://web.archive.org/web/20230613233731/https://explorer.sapphire.oasis.io/>, <https://web.archive.org/web/20230613234509/https://coinmarketcap.com/currencies/oasis-network/>, <https://oasisprotocol.org/blog/the-oasis-ecosystem-fund-reaches-235-million>, <https://archive.is/NphXH>

<sup>22</sup><https://archive.is/yUJ8w>, <https://web.archive.org/web/20230613235845/https://coinmarketcap.com/currencies/phala-network/>, <https://medium.com/phala-network/phala-network-value-tops-10-million-on-new-fundraising-31a8c25f9bb9>

<sup>23</sup><https://www.crunchbase.com/organization/obscuro-labs>

<sup>24</sup>The data has not been updated since Nov 2022. <https://archive.is/OxfnG>

<sup>25</sup><https://secretanalytics.xyz/bridge>, <https://www.coingecko.com>

**Table 2: Chain Data of TEE-based Frameworks as of June 2023. Obscuro is still in development and has not launched a mainnet yet. We list the overall count of utilized addresses across all paratimes within Oasis, where 427 of them are from the Sapphire paratime.**

	Secret Network <sup>20</sup>	Oasis Sapphire <sup>21</sup>	Phala <sup>22</sup>	Obscuro <sup>23</sup>
Online since	2020-02-13	2022-12-16	2022-04-01	N/A
Number of Transactions	8,317,479	176,951	3,363,334	N/A
Market cap	\$63M	\$257M	\$55M	N/A
Funding	\$400M	\$235M	\$10+M	\$5.85M
Cumulative Unique Addresses	248.1K <sup>24</sup>	258,255	3,447	N/A
Discord Members	37,142	63,029	6,660	8,171

**Table 3: Snip-20 Token Statistics as of February 2023**

Token Name	Market Cap (USD) <sup>25</sup>	Successful Transactions	Unique Senders
Secret Secret (sSCRT)	\$5.94m	477.6k	36.2k
Shade (SHD)	\$4.88m	35.7k	10.3k
Sienna	\$2.87m	94.2k	6.7k
Secret Eth (SETH)	\$1.93m	39.5k	4.3K
Eth (SETH BSC)		5.2k	0.7k
Tether (SUSDT)	\$1.55m	53.8k	2.8k
Secret Wrapped BTC (SWBTC)	\$950k	22.7k	1.2k
Monero (SXMR)	\$890k	26.5k	1.5k
Secret USDC (SUSDC)	\$475k	8.3k	0.5K
USDC (SUSDC BSC)		9.6k	1.7k
Secret Finance (SEFI)	\$339k	111.5k	8.8k
Binance (SBNB BSC)	\$124k	27.0k	3.8k
Buttcoin (BUTT)	\$108k	13.4k	1.7k
ALTER	\$123k	21.8k	2.7k
Dai (SDAI)	N/A	105.0k	0.5k
StkdSecret (stkd-SCRT)	N/A	32.0k	5.7k
(Combined)	\$20.2m	1.08m	44.9k