ZKR-TEE: A Zero-Knowledge Rollup Framework in Trusted Execution Environments*

Gianni Canavero, Gaurav Choudhary†, and Nicola Dragoni

DTU Compute, Technical University of Denmark, Lyngby, Denmark
{s243529, gauch, ndra}@dtu.dk

Abstract

The scalability limitations of blockchains such as Bitcoin and Ethereum have led to the development of off-chain solutions known as Rollups where transactions are processed in batches off-chain while relying on the underlying blockchain only for storage or for final verification. This created two families of rollups: Optimistic rollups, which offer low costs but require several days to confirm the transactions, and Zero Knowledge (ZK) rollups, which are confirmed within minutes at the expense of higher costs. In this work, we propose a zero-knowledge prover for rollups that leverages Trusted Execution Environments (TEEs) to implement a dual-proof mechanism that allows for a flexible trade-off between cost and confirmation time. ZK-proofs ensure correctness and transaction validity, while TEE remote attestation proofs provide a lightweight and cost-effective alternative for validation of the batches. By adjusting the frequency at which each type of proof is verified, our system is able to reduce operational costs up to 65% compared to conventional ZK-rollups, while maintaining finality of transactions within a few minutes. We implemented our system using Winterfell, a high-performance STARK prover, and Intel TDX Trusted Execution Environment. The result of this work is a practical rollup prover proof-of-concept that balances efficiency, cost, and security, laying the groundwork for new blockchain solutions.

Keywords: blockchains, zero-knowledge, TEE, ZK-rollups.

1 Introduction

In the last decade, cryptocurrencies have grew in popularity and adoption, showing the criticality that lies in the scalability of this instrument. For the first trimester of 2025, the average processed transactions per second (TPS) by the second-largest cryptocurrency by volume, Ethereum, is 14 transactions [4]. For the same network, the theoretical maximum is 114 TPS [5]. When compared to credit card systems such as Visa, which in August 2017 claimed to process 65000 TPS [28], the scalability issue is clearly notable.

The nature of the blockchain complicates scalability, each block is usually created with a specific time span and its size is limited. For Ethereum, the creation timespan is of 12 seconds and the block size is limited to 30 million gas (below 2MB on average) [22] [2]. Thus, those factors limit solutions implemented into the blockchain. To overcome this issue, an important effort has been put into developing new solutions built on top of the blockchain. These solutions are called Layer 2 solutions (L2), because they run outside the blockchain but still leverage it by submitting information in the blocks in a compressed way.

The most prominent of these solutions are zero-knowledge (ZK) rollups. ZK-rollups networks process transactions off-chain, group them in batches, and produce a single cryptographic proof

*Proceedings of the 9th International Conference on Mobile Internet Security (MobiSec'25), Article No. 39, December 16-18, 2025, Sapporo, Japan.

†Corresponding author

for the whole batch, which is then stored and verified by the blockchain. Rollup integrity is guaranteed thanks to publicly available data published to the blockchain, whereas efficiency and scalability are achieved by compressing the transactions data into a single cryptographic hash.

As of March 2025, there are currently 60 Rollups networks deployed, of which 16 are using zero-knowledge proving [26][6].

Zk-rollups are able to speed up the processing of transactions, but introduce new costs for the verification of the proofs. These costs vary based on the number of transactions in the batch processed and range around 760,000 to 996,000 Ethereum's gas [7][15][1], which correspond to 50 to 66 USD at the 1st of January 2025.

Another approach to processing batches of transactions are the Optimistic Rollups, which do not produce any cryptographic proof but rely on user verification and introduce a dispute period for reporting invalid transactions. Thus, the finality of the transactions is achieved at the end of this period, which is usually of 7 days.

Wen et al. [29] propose Tee-rollup, a system that uses a network of Trusted Execution Environments (TEE) to balance on-chain cost and finality, by submitting the shorter and less expensive to-verify remote attestation proof to the blockchain instead of a ZK-proof.

A TEE is a secure and isolated hardware within a CPU; it has been developed to ensure that sensitive data and computations executed on it remain protected even in a compromised system. Additionally, TEEs can provide cryptographic proof of their initial state by remote attestation.

Unfortunately, TEEs are subjected to continuous improvements to address emerging threats. Vulnerabilities affecting TEEs have been found in the past, which have been subsequently been patched [3][21][27]. To address this issue, Tee-rollup uses a network of different TEEs which have to agree on the blockchain state after each batch of transactions and submit each one its own proof of execution.

In this work, we follow the approach of Tee-rollup by investigating the feasibility of using a single TEE which produces both a ZK-proof and its remote attestation proof. This dual-proof strategy optimizes on-chain verification: the system primarily uses the cheaper TEE proofs for most operations to reduce costs, while the ZK proofs serve as a less frequent but robust check on the TEE's integrity. Transactions reach finality when a ZK-proof is verified. by adjusting the frequency of proof verification, the system can be adapted to suit different applications: prioritizing faster finality at the expense of higher fees, or lower fees with slower finality.

Additionally, working with a single TEE vendor eases the complexity of the maintenance of a network of different TEEs, each one requiring ad-hoc application development, while maintaining a robust and cheap form of integrity verification.

1.1 Scope of this work

This work aims to investigate the integration of ZK-rollups and TEEs to analyze its feasibility and the benefits it can provide for the blockchain's Layer 2 development.

Our main contribution are the following:

- Design and implementation of a ZK-STARK proving system within a TEE
- Comprehensive performances evaluation of the solution and comparison with other systems
- Comparison of costs and analysis of feasibility of this kind of systems

2 Related Works

Trusted Execution Enviroments are already been used as platforms for various blockchain project, such as confidential smart contracts[19], TEE and blockchain-backed data sharing system [31] and Teechain, a payment network based on Bitcoin blockchain [20], within many others.

To the best of our knowledge, TeeRollup by Wen et al. is the only other implementation of a Layer 2 solution using Trusted Execution Environments [29].

In TeeRollup, transactions are processed off-chain by a group of TEEs from different vendors. In specific, Intel SGX, Intel TDX and Hygon CSV are used. One of the TEEs initiate the processing by sending the list of transactions to the rest of the group which reports back with their responses and TEE quotes. If the majority of TEEs agrees on the correctness of the transactions, those are sent to the on-chain contract which verifies the provided quotes.

TeeRollup is able to cut state update on-chain verification costs from 760 000 gas of ZK-SNARK rollup systems or 996 000 gas of ZK-STARK systems to 156 000 gas while still maintaining a finality of transaction within minutes.

3 Proposed Solution

The primary objective of this work is to build a proof-of-concept ZK-rollup prover, which guarantees integrity of the system, even in the event of a compromised TEE, and generally a fast finality of transactions.

System integrity: Correctness and consistency of state transitions is guaranteed even under adversarial conditions.

Fast finality: Transactions are processed and finalized on the blockchain within a time frame of a few hours, aligning with the latency typically observed in other ZK-rollup systems.

3.1 System model

We consider a network consisting of n sequencers, each responsible for ingesting transactions, aggregating them into batches, and forwarding them to a pool of m provers. These sequencers operate outside a TEE and may show either honest or malicious behavior.

Provers operate within Intel TDX environments and are responsible for generating both a remote attestation quote of the integrity of their initial state, and a zero-knowledge proof (ZK-proof) verifying the correctness of their transaction analysis. Both proofs are submitted to the blockchain and verified according to a smart contract.

Intel TDX has been chosen due to its high CPU and memory performances [8].

Additional smart contracts are implemented to force redeemability of tokens in case of malicious or unresponsive sequencers and provers.

3.2 Threat model

Following Teerollup work [29], we also adopt a threat model which considers the TEE as temporary compromisable. On the contrary, previous works, such as the already cited Teechain

[20] or others like L2chain [32] and POSE [13] consider TEE-specific vulnerabilities outside the scope of their work.

In this work, we acknowledge that TEEs may be vulnerable to specific attacks or exploits at certain points in time, due to undiscovered flaws or threats. However, such vulnerabilities are eventually identified and patched, restoring the TEE's integrity. This approach follows the recommendation from Georgia Institute of Technology's *sgx.fail* team: "We recommend that developers have a TCB recovery plan in place, and should implement mitigations against SGX compromise as a development priority" [27].

The following is the list of cases considered for the threat model. Since the sequencer and the prover operate independently, the scenarios can appear in any combination.

- **Honest sequencer:** No deviation from the protocol.
- **Malicious sequencer:** Transactions assigned to these sequencers can be censored or delayed. Communication with provers is compromised. Availability of the service is not guaranteed.
- **Uncompromised prover's TEE:** No deviation from the protocol.
- **Compromised prover's TEE:** Again, transactions can be censored or delayed. Therefore, availability of the service cannot be guaranteed. In addition, the prover can fake its remote attestation quote as it has control of the TEE's private key. Whereas we expect the ZK-proof to be truthful. Thus, a compromised prover can submit fraudulent transactions only if the TEE's quote is verified.

Main chain assumptions. The smart contracts executed on the main chain enforce the finality of the transactions. The smart contracts collects a group of batches of transactions and verifies the remote attestation proof for the majority of the batch. For a randomly selected minority of the batches, the ZK-proof is verified. Once all proofs are correctly evaluated, the batches are submitted to the main chain. Once on the main chain, finality is provided as every participant can verify the validity of the publicly available data.

3.3 Design challenges

A compromise of either the prover or the sequencer, is the reason for the main challenges. Specifically to our system's goals, a compromise of the former can harm both correctness and redeemability, whereas a compromise of the latter can harm only redeemability.

- **Compromised correctness:** Recalling Section 3.2 a compromised TEE can forge remote attestation proofs. To mitigate this, we leverage our double-proofing system. In a group of batches of transactions, the smart contract will approve the majority of the batches by verifying its remote attestation proof to lower on-chain costs. Whereas the ZK-proof is verified for the remaining batches. Both proofs are submitted to L1, and the smart contract chooses with a random weighted binary function which proof it should verify. This makes it harder for a malicious actor to know when it should generate a forged proof or a real one.

  The smart contract stores the batches into the blockchain only after all the proofs in the group have been verified.

  The probability of the random function to choose which proof to verify is directly correlated to the success of an attack, this will be explored in depth in Section 4.

Both proofs are stored with transaction data, this enables the network to restore integrity in the case of a successful attack.

- **Compromised redeemability:** Redeemability can be compromised by an attack to both the prover and the sequencer. Following various implementations [25], we introduce a smart contract based challenge mechanism to enforce the redeem of tokens. In case an user's transaction is not processed, the user can submit it to the smart contract, which triggers the challenge. If the funds are not released in the allotted time, the smart contract unlocks all the funds for withdrawal on the main chain. This deterrent ensures the redeemability of the funds, even if the sequencer or the prover go offline.

3.4 System architecture

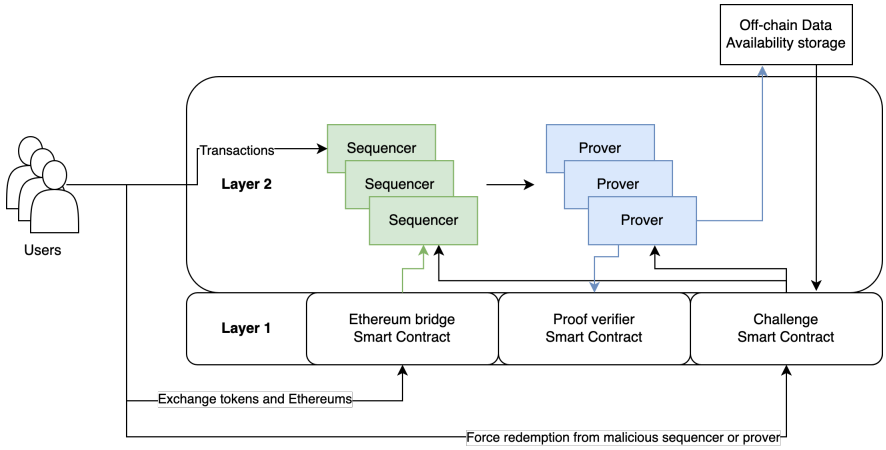

Figure 1: High level architecture of the system.

The system architecture is divided into 4 components: the sequencers, the provers, the off-chain storage for data availability, and the smart contracts.

Here we will briefly give an high-level explanation of the components:

The sequencers:

Sequencers collect, order, and group in batches, user submitted transactions. Each sequencer has a publicly available endpoint where the rollup users can request funds transfers. Sequencers are connected also to the Ethereum bridge contract, which handles deposits and withdrawals between the Layer 1 blockchain and the rollup.

The provers:

Provers verify the correctness of the list of transactions and generate both a ZK-proof and a remote attestation quote. Both proofs are sent to the proof verifier smart contract and to the off-chain data availability storage, but only the latter stores them long-term.

The off-chain storage for data availability:

To ensure full access to the whole transaction data history, we use off-chain storage, following an approach similar to that of ZK-Porter and Polygon Avail [14][17]. The prover stores the

transactions list and both ZK-proof and remote attestation quote in the off-chain storage, whereas only a hash of these elements is submitted to the blockchain.

Smart contracts: The smart contracts are publicly accessible similarly to the sequencers, the following is a list of their main functionality:

- **Ethereum bridge contract**: this contract permits to the user to trade Ethereums for rollup tokens or vice versa.

  Deposit: When called, the client deposits Ethereums into the contract. A sequencer is invoked to schedule a transaction of an equivalent amount of token with an unspecified sender, similar to what is done with coin minting, and as recipient the user account in the rollup. In case of the first deposit, the account is created and then the smart contracts saves a key-value pair mapping Ethereum account address with rollup account address.

  Redeem: a transaction burns the amount of redeemed tokens by setting an unspecified recipient. The smart contract sends the requested amount of Ethereums to the mapped Layer 1 account.

- **Proof verifier Smart contract**: this contracts verifies one of the submitted proofs and stores the new state into the blockchain.

  A random weighted function chooses which proof to be verified. In the case of a remote attestation proof, the contract verifies the signature using a preloaded Intel public key. Otherwise, for the ZK-proof, the verify script defined in the Winterfell's AIR is used. In both cases, an hash of the proof is submitted to the blockchain, whereas the proof is stored off-chain.

- **Challenge Smart contract**: this contract allows the user to force transactions in case of malicious sequencer or prover.

  The contract starts a challenge with the responsible sequencer or prover. A time limit τ is imposed to process the transactions otherwise the contract will ban the sequencer/prover from the rollup and settle the transactions by itself. The contract can reconstruct the state and force the transaction thanks to access to off-chain data availability storage.

3.5 Rollup prover

The prover uses the Winterfell STARK library to generate the new state and prove the correctness of its computation over a Merkle tree of 2^{15} accounts.

3.5.1 ZK-proofs with Winterfell

We chose to use a STARK proof system because it does not require a trusted setup and offers post-quantum security guarantees. For implementation, we selected Winterfell, a STARK prover developed by Facebook, due to its performance, developer-friendly design, and the fact that it is an actively maintained project [10].

To add Zero-Knowledge capabilities, we use a modified Winterfell version that adds randomization of the execution trace and salted Merkle trees for the commitment scheme following the work of Ulrich Habock and Al Kindi [16].

3.5.2 Proof options

The proof are selected based on Winterfell's recommended defaults, aiming to achieve a balance between proof size and generation time [11]. We increased the number of queries from the recommended 28 to 40 to reach better security without increasing proof generation time. The generated proof has a conjectured security level of 127 bits.

3.5.3 Prover steps

To optimize FFT-based computations during proof generation, we adopt a power-of-two number of steps for the proof computation. Hashing is the base operation of the state update. It is done to generate the updated leaf of the Merkle tree based on the account's new balance and nonce and then to update all the ancestor of the leaf in the tree. Therefore, hash computation should be done in a power-of-two number of steps.

We model our hash implementation after Winterfell's use of the Rescue hash function. Each hash is computed over 7 steps, with an additional step required for register setup, resulting in a total of 8 steps per hash operation. Consequently, the full base hashing operation spans 2^3 steps in our trace. The hash computation is explained more in-depth in Section 3.5.4.

Furthermore, we restrict each batch to a power-of-two number of transactions. In our experimentation, we use batch sizes of $2^7 = 128$, $2^{10} = 1024$, and $2^{14} = 16384$ transactions.

Finally, the height of the state Merkle tree must follow the form $2^x - 1$, allowing for a total of 2^x hash computations for each transaction, counting the updated leaf hash (1) plus the tree traversal ($2^x - 1$ hashes). In our tests, we use a tree height of 15, which supports up to $2^{15} = 32{,}768$ user accounts. For production deployments, however, we recommend using larger tree heights, such as 31 or 63, making it highly improbable for typical applications to exceed it.

In our benchmarked computations, the total amount of steps for 1024 transactions corresponds to:

$$\text{Total steps} = (\text{hash setup} + \text{hash compute}) \cdot (\text{leaf} + \text{tree traversal}) \cdot \text{n transactions} \tag{1a}$$

$$= (1 + 7) \cdot (1 + 15) \cdot 2^{10} = 2^{17} = 131072 \text{ steps} \tag{1b}$$

3.5.4 Rescue hash

The Rescue Prime hash function is implemented with a state width of 6 field elements ($6 \cdot 128 = 768$ bits, 96 bytes), of which four elements are allocated for the rate and two for the capacity. The output digest consists of the first two field elements (32 bytes). Each permutation of the hash function uses seven rounds, providing a 128-bit security level with a 40% security margin. This choice differs from the Rescue Prime specification, which recommends 8 rounds to achieve a 50% margin, but following Winterfell's approach, we greatly simplify the Algebraic Intermediate Representations (AIR) by using one less step [24] [12].

3.5.5 Prover function and trace registers table

The shown code is an high-level pseudocode outlining the steps performed by the prover.

accounts is the array containing all the accounts information, such as address, nonce, and balance. *stateTree* is the Merkle state tree, where each leaf node represents the hash of an individual account's data. Finally, *transactions* is the list containing the transactions' sum, recipient, and sender.

The flags are used to indicate if the currently considered A' and B' are left or right child and to flag each first step of a new transaction computation.

Setup steps can do multiple operations in a single step because they modify the same trace registry only once.

Prover

```
1: procedure BUILDTRACE(self, accounts, stateTree, transactions)
2:     for each tx ∈ transactions do
           // Setup new A,B hash in 1 step
3:         Set init_transaction_flag
4:         Load A_balance, A_nonce into registers [0, 1]
5:         Load B_balance, B_nonce into registers [2, 3]
6:         Update A_balance, A_nonce and B_balance, B_nonce
7:         Store A_balance, A_nonce in registers [5, 6]
8:         Store B_balance, B_nonce in registers [11, 12]
9:
10:        Unset init_transaction_flag
11:        compute 7 Rescue Hash rounds                          ▷ 7 steps
12:        Copy computed A'_hash in stateTree.leaves[A_index]
13:        Copy computed B'_hash in stateTree.leaves[B_index]
14:
15:        for i = 0; i <stateTree.Height; i++ do
               // Setup interior nodes hashes in 1 step
16:            if A' is left child then
17:                Load A'_hash in registers [5, 6]
18:                Load sibling hash into registers [7, 8]
19:            else
20:                Load A'_hash in registers [7, 8]
21:                Load sibling hash into registers [5, 6]
22:            end if
23:            if B' is left child then
24:                Load B'_hash in registers [11, 12]
25:                Load sibling hash into registers [13, 14]
26:            else
27:                Load B'_hash in registers [13, 14]
28:                Load sibling hash into registers [11, 12]
29:            end if
30:
31:            compute 7 Rescue Hash rounds                      ▷ 7 steps
               // Part of next iteration's setup interior nodes step
32:            Store parent(A'_index) into A'_index
33:            Store parent(B'_index) into B'_index
34:            Copy computed A'_hash in stateTree.leaves[A'_index]
35:            Copy computed B'_hash in stateTree.leaves[B'_index]
36:        end for
37:    end for
38: end procedure
```

3.6 AIR's constraints

Here we list the set of transition constraints implemented in the AIR to ensure the correctness of the state transitions throughout the trace. There is a total of 17 constraints.

Rescue permutation and rescue initialization. We defined 12 constraints of degree 5, one for each Rescue hash register ([5..11] and [11..17]). These are enforced in the 7 out of 8 rounds of the Rescue permutation. A mask identifies the current step. In the following formula, x_i is each hash permutation register before the Rescue hash step computation, whereas x_{i+1} is the same register after the step. The degree of the constraint is determined by the S-box, which maps each input x to its fifth power.

The initial round of the 8 hash computation rounds in the Merkle tree traversal is used to initialize the hash during Merkle tree reconstruction. To ensure that the previously computed hash is correctly placed for the computation of the new parent hash and that the 2 capacity registers are reset to 0. We implemented 12 constraints based on the binary flag indicating whether the node is a left ($flag = 0$) or right ($flag = 1$) child in the tree structure. These are not separated constraints from the previous, as they are applied in different steps of the computation and are of a lower degree.

$$mask = [0, 1, 1, 1, 1, 1, 1, 1] \tag{2a}$$

$$mask[i] * (mds(sbox(x_i)) - sbox(mds^{-1}(x_{i+1}))) = 0 \tag{2b}$$

Balance updates. To guarantee the integrity of balance updates, two additional constraints of degree 1 are enforced, verifying that the sender's and receiver's balances are updated in accordance with the transaction value.

Binary flags. Finally, 3 more constraints of degree 2 are enforced to guarantee that the flags are binary.

3.7 Adversarial scenarios and mitigations

3.7.1 Malicious sequencer or prover operation

In the case that one or more transactions get dropped or delayed, the user can force their execution through the *Challenge Smart Contract* by providing the same information sent to the sequencer.

The contract verifies the correctness of the report by querying the sequencer proxy's log and the off-chain data availability storage for the submitted transactions. If the proxy's logs file contains the transaction but the data availability storage does not, the transaction has not been processed yet. The user has to provide collateral for the cost of this operation, which is released if the request is honest.

On a valid claim, the contract initiates a challenge with the sequencer or prover, which has to process the transaction within a time span τ. τ is a multiple of the block time of Ethereum, and it is set to several hours.

In the case the transaction is settled within τ the rollup proceed as normal. Otherwise, the contract bans the malicious sequencer or prover from the system and proceeds with forwarding the transaction again to the sequencers pool. In case of a withdrawal operation, the contract directly executes it. The final outcome is recorded on-chain for transparency.

To prevent malicious operation, a collateral for sequencers and provers can be introduced, which is slashed in case of misbehavior.

3.7.2 Incorrect proof submission

If a prover submits a proof that fails verification, the corresponding batch is immediately rejected and returned to the sequencer pool for reprocessing, ensuring the soundness of the system is preserved.

The prover responsible for the invalid submission is permanently removed from the rollup, and this action is publicly recorded on-chain.

Again, to discourage such behavior, the system may require provers to post collateral when registering to the rollup system. This collateral would be returned when the prover de-registers from the rollup only if it has not manifested malicious behavior.

In case of incorrect ZK-proof submission, the contract will temporarily enter into a safety mode, which until external intervention will only verify ZK-proofs.

4 Results and Discussion

This section demonstrates how our system meets the security, integrity and performance requirements defined in Section 3.

4.1 Security analysis

We analyze security under the assumption that TEE vulnerabilities capable of forging or replaying quotes are publicly disclosed before an exploit based on them is found. Once such a disclosure appears, we expect the maintainers of the protocol to rapidly update the verifier's weighted functions to temporarily exclude all the TEE quotes verification, using only the ZK-proofs.

This assumption makes the security of the system comparable to the one of other ZK-rollup networks already deployed on Ethereum.

To ensure comprehensiveness, Section 4.1.1 covers the case of an attack using an undisclosed vulnerability for Intel TDX.

Following L2beat's risk analysis, we focus on state validation, data availability, sequencer failure, and proposer failure [25].

State validation. State validation is achieved by ZK STARK proofs, which are verified by the on-chain verifier smart contract for every group of batches. This would be rated secure in L2beat's ranking.

Data availability. State diffs containing a list of transactions, new state, ZK-proof, and TEE quotes are stored off-chain, but their commitment hash is submitted to Layer 1. Additionally, the new state root is stored on-chain. This follows the approach used by Starknet's Volition mode [23], which is deemed secured following the classification.

Sequencer failure or malice. In case of an unavailable or malicious sequencer, which censors transactions, users can enqueue via a Layer 1 smart contract their transactions. This fallback, used by several other rollups systems, is rated as Medium security by L2Beat. It could be improved to High by executing the entire fallback transaction on L1, with higher processing costs.

Prover failure or malice. In case of an unavailable or malicious prover, users can enqueue via a Layer 1 smart contract their transactions. In our system there is no prioritized prover, therefore anyone can self-propose as a prover and process the transactions that were dropped from the failed proposer. This is deemed secure following L2beat classification.

Alignment with design goals. Relative to the objectives stated in Section 3, system integrity is guaranteed by the state validation mechanism based on ZK-proof, whereas fast availability is discussed in Section 4.2.

For availability in general, when considering the threat model cases of a compromised sequencer and/or prover the implemented smart contracts ensure the availability of the system. An additional tool that we introduced to prevent malicious behavior is requiring sequencers and provers to provide collateral and slash it in case of unavailability or censorship.

4.1.1 Undisclosed TEE vulnerability case

The worst-case scenario for our system occurs when a vulnerability in the TEE is known only to the attacker and has not yet been publicly disclosed. In this scenario, the attacker may be able to forge or replay TEE quotes, but cannot generate invalid ZK-proofs that pass verification due to their cryptographic soundness.

Let T_T be a truthful TEE quote, whereas T_F is a forged one. For ZK-proof we define Z_V for a valid proof and Z_I for an invalid one. The possible pairs of proofs are the following:

Table 1: Proof pair cases

Proof pair	**Explanation**
T_T, Z_V	Normal operation, both proofs are valid. Always accepted.
T_T, Z_I	Computational error/bug. Only TEE quote is accepted.
T_F, Z_V	Forged quote generation but correct processing. Only ZK-proof passes.
T_F, Z_I	Malicious quote generation and execution. Only TEE quote passes.

The pair (T_F, Z_V) represents an attack targeting system availability and is automatically mitigated: the verifier contract bans the prover and resubmits the batch of transactions to the pool of sequencers.

In contrast, an attacker targeting state integrity is interested in submitting only the (T_T, Z_V) and (T_F, Z_I) pairs.

Thanks to the weighted randomized selection mechanism for proof verification, the attacker cannot predict whether the secure ZK-proof or the forgeable TEE quote will be verified. This introduces a probability of discovering the attack, which is linearly correlated to the weight assigned to the ZK-proof verification.

Let $P(M)$ be the probability that the attacker submits the malicious pair (T_F, Z_I) and $P(Z)$ the weight assigned to the ZK-proof verification, we have:

$$\text{Probability of discovering the attack} = P(M \cap Z) = P(M) \cdot P(Z) \tag{3a}$$

$$\text{Probability of success of the attack} = \tag{3b}$$

$$= (P(M) \cdot (1 - P(Z))) \cdot ((1 - P(M)) \cdot P(Z)) \cdot \prod_{i}^{i \geq 0} (1 - P(M)) \tag{3c}$$

The probability of a successful attack depends on the attacker being able to submit at least one forged TEE quote that is selected for verification, at least one valid ZK-proof that passes verification, and an indefinite number of otherwise valid proof submissions.

Furthermore, we can use a geometric distribution to estimate the expected number of batches processed before the malicious batch is caught via ZK-proof verification.

$$\text{Geo} \sim P(M \cap Z) \qquad f_x(x) = P(M \cap Z) \cdot (1 - P(M \cap Z))^{x-1} \tag{4a}$$

$$E[X] = \frac{1}{P(M \cap Z)} \tag{4b}$$

This provides a probabilistic estimate of the number of malicious batches that may be submitted before the attack is detected. It is important to note that once a malicious batch is identified, the verifier's smart contract switches to verifying only ZK-proofs, preventing submission of other compromised batches until manual intervention occurs.

Furthermore, through external intervention, the rollup state can be restored to a correct version by retrieving the full transaction history from the data availability storage, identifying the malicious batches, and replaying only the valid transactions.

4.2 Performance evaluation

This section presents a performance evaluation of the proposed Prover based on benchmarking.

4.2.1 Test hardware

The prover has been tested on two different hardware configurations. The listed costs correspond to instances in europe-west4 region, as of June 2025.

Untrusted hardware: Google Cloud N2D instance with 4vCPU AMD Epyc Milan and 16GB of RAM. OS: Debian 12.
Cost: 0.18604 USD per hour

Trusted hardware: Google Cloud C3 instance with 4vCPU Intel Xeon Sapphire Rapids with **Intel TDX** enabled and 16 GB of RAM. OS: Ubuntu 24.04 LTS Confidential VM.
Cost: 0.21169 USD per hour

4.2.2 Measured performances

Table 2: Average execution time in ms.

Transactions per batch	**Untrusted 99bit (ms)**	**Trusted 99bit(ms)**	**Proof Size (KB)**	**Untrusted 127bit (ms)**	**Trusted 127bit (ms)**	**Proof Size (KB)**
128	1195.9	1258.8	55.4	1248.0	1322.1	76.4
1024	10,268.5	10,834.7	76.4	10,750.1	11,297.6	103.3
16384	182,102.3	192,530.8	107.2	187,360.8	198,867.1	145.2
Time increase:		5.5%			5.7%	

To evaluate the efficiency of proof generation under different configurations, we measured execution time and proof size across varying batch sizes and security levels. Specifically, we compared the performance of producing a ZK-proof of 99 bit conjectured security and 127 bit in both untrusted and trusted execution environments. The results are summarized in Table 2.

The 99-bit conjectured security proofs are constructed using 28 queries, producing a list decoding regime of 55 bits and a unique decoding regime of 39 bits. In contrast, the 127-bit security proofs require 40 queries, resulting in a list decoding regime of 69 bits and a unique decoding regime of 49 bits.

As expected, increasing the security level from 99 bits to 127 bits introduces some computational overhead. However, this overhead is modest: the average execution time increases by at most 5.03% which occurs for 128 transactions, whereas it slightly decreases with larger batches (4.27% for 1024 and 3.29% for 16384).

While proof size increases substantially at 127-bit security, this has no impact on on-chain costs due to the use of off-chain storage. Note that because we use a 256-bit hash in the Fiat–Shamir transform, the protocol cannot offer more than 128 bits of conjectured security as this is the ceiling set by hash collision probability.

Regarding hardware performance, the comparison between untrusted and trusted environments shows that for both security levels, the proving time increases by less than 6%, which remains well within the bounds required to meet our fast finality objectives.

4.2.3 On-chain costs

We collected on-chain costs from different sources to estimate the total cost to operate our system. These costs are shown in Table 3.

Gas is the unit used by Ethereum for computational cost. For better comprehension, we converted the gas to Ethereum based on the average gas cost of 2024 ($19.73 \cdot 10^{-9}$) [9]. USD conversion is based on Ethereum price at the 1st of January 2025 opening (3335.91 USD).

Table 3: List of prices in Gas, Ethereum and US dollars for different operations.

Operation	**Gas**	**Ethereum**	**USD at 01/01/2025**
SNARK proof verification (Plonk) [1]	760,000	0.014,99	50.02
STARK proof verification (Boojum) [1]	996,000	0.019,65	65.55
TEE Quote verification [29]	156,263	0.003,08	10.28
Hash storage on-chain [30]	20,000	0.000,39	1.32

SNARK proof verification is cheaper than STARK due to its linear complexity, whereas verifying a TEE quote is almost 5 times cheaper than verifying a SNARK proof. Finally, storing a hash on the blockchain is a minor cost, which is important for our use case because we want to store two hashes for the two committed proofs instead of a single one.

Table 4 presents the average costs of proving a batch of transaction within a group of 10 batches. Each row corresponds to a different weight configuration for the proof verification function. Notably, our approach achieves a 64 % cost reduction while maintaining an average finality time slightly above one minute.

To explain how this is calculated, we focus on the 20 % ZK-proof configuration. In a group of 10 batches:

- 2 will be proven with ZK-proof, costing 65.55 USD each.
- The remaining 8 cost 10.28 USD each due to the verification of the TEE Quote.

Table 4: List of prices for 10 batches of 1,024 transactions, varying the verifier function weights. Costs are in USD at 01/01/2025 for on-chain fees and June 2025 for hardware fees.

ZK-Proofs	TEE quotes	Cost per batch	Cost per transaction	Cost reduction	Finality
100 %	0 %	68.18$	0.067$	0 %	13.56s
90 %	10 %	62.66$	0.061$	8.11%	15.06s
80 %	20 %	57.13$	0.056$	16.21%	16.95s
60 %	40 %	46.08$	0.045$	32.42%	22.60s
40 %	60 %	35.02$	0.034$	48.63%	33.89s
20 %	80 %	23.97$	0.023$	64.84%	67.79s
10 %	90 %	18.44$	0.018$	72.95%	135.57s

- Additionally, each batch incurs a storage cost of 2.64 USD to store the two ZK-proofs.

The finality is reached every time we verify a ZK-proof. On average, one ZK-proof is verified every 5 batches. Therefore, we compute the average finality time by multiplying the ZK-proof generation time by 5. In addition, we consider also a 20 % overhead for communication between components.

4.3 Comparison with counterparts

Table 5 compares our solution with different weight configurations with ZKsync Era and Polygon zkEVM. We chose these two rollup systems because they are fully open-source, making them well-documented and easier to analyze based on available research. Additionally, they represent opposite ends of the rollup design spectrum:

ZKsync Era employs a GPU-based prover, enabling massive parallelization and high throughput. While it is efficient in terms of CPU and RAM usage (only a third, and a tenth, respectively, compared to our configuration), it relies on batching 3,000 to 5,000 transactions to amortize costs. This leads to longer finality times while maintaining a low hardware cost. In our comparison, we use the suggested hardware configuration based on 32 vCPUs, 128 GB RAM, and one NVIDIA L4 GPU.

Polygon zkEVM By contrast, Polygon zkEVM is designed for low-latency finality. It submits very small batches (typically less than 50 transactions), achieving quicker confirmations. However, this approach introduces higher fees per transaction unless ZK-proofs are aggregated and compressed efficiently. Considered hardware: 96 vCPUs and 768 GBs of RAM.

To compare our solution with other rollup systems, we have to account for hardware costs in the evaluation and estimate its performance at the scale of a real system.

Hardware costs: To simulate a real-world scenario, we adopt the hardware configuration recommended for Polygon's zkEVM: 96 vCPUs and 768 GBs of RAM. This decision is supported by our experimental results using Intel TDX, which showed no significant increase in CPU or memory usage when running in a trusted environment, as confirmed by tools like *top* and Valgrind's *Massif*.

Table 5: Our solution compared to ZKsync Era and Polygon zkEVM rollups.

Rollup	Gas per batch	Gas per transaction	Hardware cost per hour	Cost per transaction	Finality
ZKsync Era [7] [26]	816,275	209	1.87 $	0.014$	1 h
Polygon zkEVM [7] [26]	59,434	2201	6.023$	0.149$	21 m
80-20 solution	868,053	847.71	6.668$	0.112$	119 s
60-40 solution	700,105	683.69	6.668$	0.090$	158 s
40-60 solution	532,158	519.69	6.668$	0.069$	4 m
20-80 solution	364,210	355.67	6.668$	0.048$	8 m

Scale: When compared to the two reference systems, our batch size of 1024 transactions appears well balanced. However, our proof of concept is tested against a state size holding up to 32 768 accounts, significantly below the scale expected in realistic deployments, which may involve billions of accounts. Scaling the state introduces increased memory demands, which are manageable given our high-memory configuration.

Fortunately, when considering the proving time, the growth is logarithmic when varying the account number thanks to the usage of Merkle trees. The finality time and the costs per transaction hypothesized in Table 5 are based on a conservative state size of 2^{63} accounts, resulting in an expected proving time over 4 times longer than in our experiments. Additionally, we include a 100 % overhead assumption to account for slowdowns associated with managing this substantially larger state.

The main reason Polygon zkEVM takes longer to reach finality is its use of proof aggregation [18]. Rather than submitting a ZK-proof for each individual batch, it waits to accumulate multiple ZK-proofs and then aggregates them into a single compressed proof, which lowers the storage prices and the verification cost per transaction. This process improves cost efficiency but introduces latency, as the system must wait for enough batches to justify aggregation. Proof aggregation can be done after the prover processes each batch and usually involves recursively combining multiple proofs into a single one. A similar system can be integrated into our rollup between the STARK proofs to further reduce costs in deployments where a longer finality time is acceptable.

4.4 Findings and limitations

Our work introduces a novel approach to rollups which leverages a double-proof system based on ZK-rollup and TEE quotes. Unlike purely optimistic or ZK-based rollups, our system offers configurable finality, allowing developers to adapt security and cost parameters to match different application profiles. This positions our design as a flexible alternative in the field of L2 scaling solutions.

The measured performances are in line with already deployed rollups and we believe that this model should be furthermore researched due to its peculiarities.

On the other hand, our experimentation is focused exclusively on the prover, which is the core part of the rollup but this limitation obliges us to estimate the performances of a full

production-ready setup. In addition, the prototype was tested with a limited number of users due to hardware constraints. This might hide performance bottlenecks caused by scaling up the system. Finally, the expected finality time is based on a continuous stream of transactions, whereas for ZKsync and Polygon systems, the value is derived from real-time data.

4.5 Security comparison with TEERollup

To the best of our knowledge, the work by Wen et al. is the only other implementation of rollups using Trusted Execution Environments [29]. Their system, TEERollup, considers the risk of TEE vulnerabilities and mitigates it by using a group of TEEs from different vendors. Consensus among this group is required to approve a batch of transactions, and each TEE quote is submitted to a Data Availability storage. Whereas the on-chain contract verifies only a single quote.

This design allows TEERollup to continue operating correctly even if a vulnerability is exploited on a single TEE, as consensus enables detection of forged quotes and exclusion of the compromised prover.

In contrast, our proposed system would, in this scenario, be temporarily vulnerable to a malicious prover, potentially allowing a limited amount of fraudulent batches before detection. However, once the vulnerability is identified, our architecture can fully switch to relying on ZK-proofs, providing continued security at the cost of higher fees. TEERollup, on the other hand, may become non-functional if multiple TEEs across vendors are compromised simultaneously.

The core difference between the two systems lies in their underlying security assumptions:

- **Our work** assumes that vulnerabilities are disclosed to the public before an attacker can exploit them.
- **TEERollup** assumes that TEEs from different vendors are not compromised at the same time.

In line with recommendations from the sgx.fail research group, our design incorporates a more comprehensive fallback mechanism in the event of a TEE compromise when compared to TEERollup [27]. However, their model relies on a simpler and more realistic security assumption.

Regarding performances, Teerollup costs for state update are composed only of the cheaper TEE quote verification and are comparable to a 0 % ZK-Proofs configuration of our system. Unfortunately, Wen et al. do not mention precisely the finality time but claim it is within few minutes.

4.6 Further research

Our work is based on a proof of concept, which, as previously noted, brings various limitations for its evaluation. A primary direction in research based on this architecture is clearly integrating it into a complete test environment. This involves developing sequencers, smart contracts for the Ethereum VM, and a deployment hardware capable of supporting real-world scale. A thorough benchmark in this context can measure communication overhead between system components and bottlenecks introduced by high memory consumption, which in this work we could only estimate. Measuring these factors is crucial to assess the rollup's scalability and its suitability for use in production ready environments.

A second direction is the investigation of proof compression techniques adapted to our double-layer security model. This approach aims to reduce ZK-proof verification costs at the

expense of longer finality, but preserving the same security guarantees. Such flexibility could make the system adaptable to a even wider range of deployment contexts, from applications requiring low latency to those prioritizing cost efficiency.

5 Conclusion

This work explored the combination of Trusted Execution Environments and Zero Knowledge proofs to build a rollup prover for Ethereum blockchain with configurable cost and performance balance. We designed, implemented, and benchmarked a dual-proof system based on the STARK prover Winterfell and a single Intel TDX enclave.

Our implementation demonstrates that the performance of this hybrid system is comparable to established rollups. In particular, using TEE quotes for most batches and ZK proofs selectively enables cost reductions up to 72 % without significantly compromising finality. In our experimentation, the Intel TDX enclave incurs only a 6 % performance overhead, confirming its practicality for this kind of workload.

System security remains sound under the assumption of a temporarily publicly disclosed TEE vulnerability. Additionally, we investigated the criticalities that lie behind this assumption, showing how the attack surface is limited even in the case of an undisclosed vulnerability.

A novel feature we introduced with this design, is the weighted verification mechanism that can offer a range of cost-finality ratios. This makes the system adaptable to varying application needs, balancing economic efficiency with fast transaction acknowledgment.

We compared our architecture to an already existing solution leveraging TEEs for Layer 2 systems, Teerollup. While Teerollup relies on a group of heterogeneous TEEs to mitigate single-vendor risks, our single TEE approach is simpler to deploy and easier to maintain, but it inherits the risk of vulnerabilities targeted at Intel TDX. The fallback to ZK proofs, however, avoids a total shutdown of the system.

In the discussion, we stated the limitations of this work, which are mostly related to scale limited to 32 768 accounts and to the lack of a complete system for testing. For this reason, we outlined as future research a more extensive experimentation, which involves the development of a full rollup system with smart contracts and sequencers, to better assess its performances in a real-world scenario.

To conclude, this work offers a practical and cost-efficient pathway to deploy rollups by combining the strengths of ZK proofs and TEEs. It extends the design space of Layer 2 solutions and we believe it lays a strong foundation for future works in secure blockchain scalability.

References

[1] Eli Barbieri. Starknet and zkSync: A comparative analysis, 2024.

[2] Blockchain.com. Average block size (MB), 2025.

[3] David Cerdeira, Nuno Santos, Pedro Fonseca, and Sandro Pinto. Sok: Understanding the prevailing security vulnerabilities in trustzone-assisted TEE systems. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1416–1432, 2020.

[4] Chaininspect. Ethereum real-time TPS, 2025.

[5] Chaininspect. What does TPS stand for in blockchain performance?, 2025.

[6] Stefanos Chaliasos, Nicolas Mohnblatt, Assimakis Kattis, and Benjamin Livshits. Pricing factors and tfms for scalability-focused zk-rollups, 2024.

[7] Stefanos Chaliasos, Itamar Reif, Adrià Torralba-Agell, Jens Ernstberger, Assimakis Kattis, and Benjamin Livshits. Analyzing and benchmarking ZK-rollups. Cryptology ePrint Archive, Paper 2024/889, 2024.

[8] Luigi Coppolino, Salvatore D'Antonio, Giovanni Mazzeo, and Luigi Romano. An experimental evaluation of TEE technology: Benchmarking transparent approaches based on sgx, sev, and tdx. *Computers and Security*, 154:104457, July 2025.

[9] Etherscan. Ethereum average gas price chart, 2025.

[10] Facebook. Winterfell, a STARK prover and verifier for arbitrary computations., 2025.

[11] Facebook. Winterfell, proof options, 2025.

[12] Facebook. Winterfell, rescue prime implementation, 2025.

[13] Tommaso Frassetto, Patrick Jauernig, David Koisser, David Kretzler, Benjamin Schlosser, Sebastian Faust, and Ahmad-Reza Sadeghi. POSE: Practical off-chain smart contract execution. In *Proceedings 2023 Network and Distributed System Security Symposium*, NDSS 2023. Internet Society, 2023.

[14] Matter Labs. zkPorter: a breakthrough in L2 scaling, 2021.

[15] Matter Labs. Awesome zero-knowledge proofs (ZKP), 2024.

[16] Polygon labs. A note on adding zero-knowledge to STARK, 2025.

[17] Polygon Labs. Polygon Avail DA, 2025.

[18] Polygon Knowledge Layer. Aggregator, 2025.

[19] Rujia Li, Qin Wang, Qi Wang, David Galindo, and Mark Ryan. Sok: TEE-assisted confidential smart contract, 2022.

[20] Joshua Lind, Oded Naor, Ittay Eyal, Florian Kelbert, Peter Pietzuch, and Emin Gun Sirer. Teechain: A secure payment network with asynchronous blockchain access, 2019.

[21] Kit Murdock, David Oswald, Flavio Garcia, Jo Van Bulck, Daniel Gruss, and Frank Piessens. Plundervolt: software-based fault injection attacks against intel sgx. In *2020 IEEE Symposium on Security and Privacy (SP)*, IEEE Symposium on Security and Privacy, pages 1466–1482. IEEE Computer Society Press, July 2020. 41st IEEE Symposium on Security and Privacy ; Conference date: 17-05-2020 Through 21-05-2020.

[22] Jacob Sharples. Ethereum docs - blocks, 2024.

[23] Starknet. Volition on Starknet: Your Data, Your Choice, 2023.

[24] Alan Szepieniec, Tomer Ashur, and Siemen Dhooghe. Rescue-prime: a standard specification (SoK). Cryptology ePrint Archive, Paper 2020/1143, 2020.

[25] L2BEAT team. L2BEAT.

[26] L2Beat Team. The state of layer 2 ecosystem, 2025.

[27] Stephan van Schaik, Adam Batori, Alex Seto, Bader AlBassam, Christina Garman, Thomas Yurek, Andrew Miller, Daniel Genkin, and Yuval Yarom. SGX fail, 2023.

[28] VISA. VISA fact sheet, 2018.

[29] Xiaoqing Wen, Quanbi Feng, Hanzheng Lyu, Jianyu Niu, Yinqian Zhang, and Chen Feng. Teerollup: Efficient rollup design using heterogeneous TEE, 2025.

[30] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger, 2025.

[31] Hui Xie, Jun Zheng, Teng He, Shengjun Wei, and Changzhen Hu. TEBDS: A trusted execution environment-and-blockchain-supported iot data sharing system. *Future Generation Computer Systems*, 140:321–330, 2023.

[32] Zihuan Xu and Lei Chen. L2chain: Towards high-performance, confidential and secure layer-2 blockchain solution for decentralized applications. *Proc. VLDB Endow.*, 16(4):986–999, December 2022.