# Decentralised and Confidential Rollups for Ethereum

Project Cinnamon team
cinnamon@capsule-corp.io

*Abstract* - **Rollups technologies are at the forefront of the ongoing effort of scaling the ETHEREUM Layer 1 blockchain via Layer 2 implementations. With several thousands of rollups, hundreds of thousands of assets and millions of users coming on board, there are potential issues arising from centralized rollup services posing significant risks in terms of security, liveness and reputational damage risks to the crypto industry.**

**Our research aims at identifying how a combination of Trusted Execution Environments (TEEs) for integrity of computations, token-based incentivization of node operators for decentralization and economic security can help solving or mitigating centralisation of rollups, to enable their large scale adoption, with well understood and minimized risks.**

**More practically, our goal is to leverage TEEs to attest, secure and decentralise all or a subset of the individual services that are a part of rollups, validiums and optimiums built on top of the Ethereum ecosystem. Thus eliminating the need for web3 projects to trust centralized operators (such as Devops teams or Rollup-as-a-service providers) and to enable the transition from *central, trusted rollups* to a *decentralized, trust-minimized rollups* era.**

**Over the course of this paper, we will delve into concepts such as TEE proofs, proof-of-batch consensus, economic security, and infrastructure-as-a-service detailing how their combined implementation can solve existing rollups centralization issues in efficient and trust-minimized ways.**

*Index Terms* - Rollups, TEE, Blockchain, TEE proofs

*Observations* - This document and research discusses proofs using SGX, Intel's TEE technology. However, there are similar technologies available from other chip manufacturers which can also be used, but the general principles remain the same.

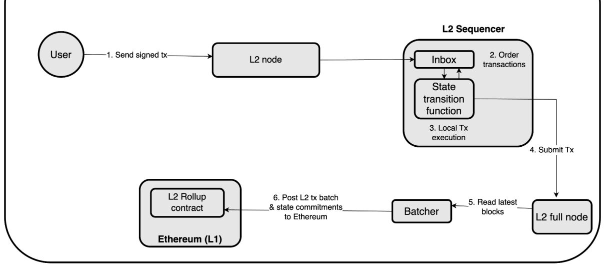
## Rollups

### I. Definitions

Rollup architectures vary widely, so do the terminologies associated with the components of the rollup stack. For the purpose of this document, here is a terminology of the various parts of the rollup stack in TEE proofs, and a diagram showing the main components of a theoretical rollup, which will be the subject of our varied implementations throughout this paper.

| Component | Description |
|---|---|
| **L1 rollup contract (Solidity)** | This is the bridge contract that stores the L2 state commitments on L1. In case of rollup, the transaction call data is also posted to L1. |
| **L1 SGX contract (Solidity)** | This is used to whitelist MRENCLAVE measurements of the L2 component binary (running within a genuine SGX enclave) that is authorised to initiate data posts from L2. It also records the authored signing keys of enclaves that have staked tokens to provide data posting services. |
| **Sequencer** | This is the component that provides transaction ordering functionality for L2 blocks. |
| **Batcher** | This is the component running in an enclave that listens to new L2 blocks and posts batch data to L1. Note that it is possible for the sequencer to double up as a batcher. |
| **L2 Full node** | The node on L2 network that validates new transactions and blocks , executes them and constructs the L2 state. |
| **Data availability** | Data availability is assumed to be provided by Ethereum L1 itself, in this post, in the classic rollup architecture. However it is possible to use external DA services to post transaction batch data or state diffs, and only post the state commitments and proofs to L1. |
| **Security council** | This is likely to be a multi-sig account representing the core social and administrative governance layer of the L2 network. It is expected to contain at least 8 members or more. |

**Figure 1: Theoretical Rollup Framework**



### II. Issues identified

Rollups work by aggregating multiple transactions off-chain and then committing their data to the main chain, significantly improving scalability and transaction throughput. However, this process introduces potential centralization points, as the operation of rollups often relies on a limited number of key services such sequencers, executors, provers and aggregators to process and post transactions to the main chain. This concentration can lead to vulnerabilities, where the compromission or failure of a few centrally-operated services could disrupt the entire rollup integrity. Furthermore, the design and implementation of proof-based rollups necessitate complex cryptographic proofs (e.g., zk-SNARKs or optimistic rollups' fraud proofs), which are not only resource-intensive to generate but also place a significant trust in the entities that generate these proofs. McCorry et al. discuss the risks associated with the centralization of trust and the potential for censorship or manipulation of transactions by a few operators within these systems[2]. Al-Bassam highlights the importance of decentralized and trust-minimized solutions in rollups to mitigate such risks, suggesting the need for protocols that ensure transparency, security, and broad participation to uphold the decentralization ethos of blockchain technology[3].

A majority of rollups rely upon critical services such as sequencers, executors and provers run in a centralized manner. These centralized components pose the following risks:

- **Censorship**: User transactions can be selectively dropped or censored, either due to bugs in code, malicious central operators or under pressure from governments.
- **MEV/Value extraction**: In centralized block building services (e.g. sequencers), value can be extracted centrally from user transactions e.g. through sandwich orders.
- **Services unavailability**: Centralized services represent a single point of failure and liveness risk.
- **Loss of user assets**: Unavailability of L2 services can result in inability of users to withdraw their assets. Even with forced inclusion, there can be significantly higher costs to withdraw L1 assets, and high risk of loss of L2-native assets.

## Code attestations (TEE proofs)

*I. Definitions*

TEE proofs, or Code Attestations, are intended as a new proof system that can either be used as an alternative to zk proofs and fraud proofs, or can be used in conjunction with them for added security.

TEE proof refers to L2 code execution authenticity guarantees provided using SGX technology that can be verified on L1.

TEE proofs can be generated for any code on the L2 stack that's executed within an SGX enclave. This includes sequencers, provers and L2 full nodes.

TEE proofs can be attached to any data that is posted on L1 contract. These proofs provide strong guarantees of valid L2 execution to the L1 contract.

*II. Architecture*

A TEE proof consists of 4 main components, described below:

**ECDSA signature of enclave**: From an application perspective, an L2 service component running in an enclave will have two identities associated with it. One is the keypair of the operator of the enclave. The second is a signing key that is generated within the enclave, which even the operator of the enclave VM will not have access to. This will ensure that any data posted from the enclave can be trusted to be signed by the enclave only, and not someone else pretending to be the enclave. This signature can be compared by the L1 contract against the list of whitelisted enclave addresses allowed to post data on L1.

**MREnclave measurement of enclave binary**: The measurement of the code and data for the L2 service component running in the enclave is taken (MREnclave)and recorded (whitelisted) in L1 contract. Every state commitment from the L2 service component is then verified against the MREnclave whitelisted, and state commitments from enclaves with invalid MREnclave measurements are rejected.

**Remote attestation report for SGX enclave**: Intel remote attestation is the process that can be used to verify that an L2 service component is running in a genuine Intel SGX enclave. To remotely attest an enclave, a signed claim called SGX quote can be generated for an enclave. The SGX quote may contain many components including a measurement of the code and the data of the enclave (MRENCLAVE), security patch level of the platform, and any user data the at the enclave wants to include in the quote. This quote is then sent to an Intel Attestation service that verifies the enclave quote and generates an attestation report certifying (or otherwise) the enclave.

**Proof expiry** : contains the L1 block number until which the message signed by the enclave is valid . This can be used to prevent replay attacks.

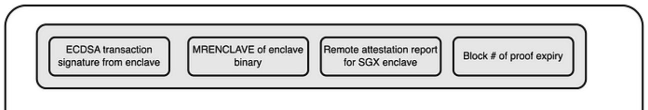

ECDSA transaction signature from enclave
MRENCLAVE of enclave binary
Remote attestation report for SGX enclave
Block # of proof expiry

Figure 2: Structure of SGX proof

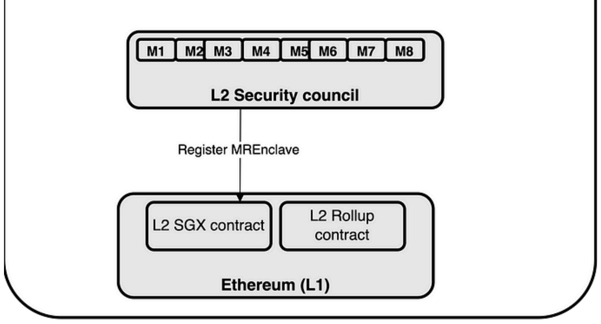
## TEE proof implementation

This section gives an overview of steps needed to setup an L2 network with TEE proofs.

*I. Initialisation*

1. **MREnclave whitelisting**: Whitelisting the enclave binary measurements (MRENCLAVE) from which L2 data is allowed to be posted to L1. This is done by the L2 security council. The code for enclave is expected to be open source. This is shown in figure 3.

Figure 3: MREnclave Configuration by L2 security council



2. **Batcher registration**: Registration of batchers by operators. As part of registration request, the Intel remote

attestation report (containing MRENCLAVE measurement of the enclave, and in-enclave generated signing key in user_data of attestation report) is sent and signed by the batcher. This is shown in figure 4.

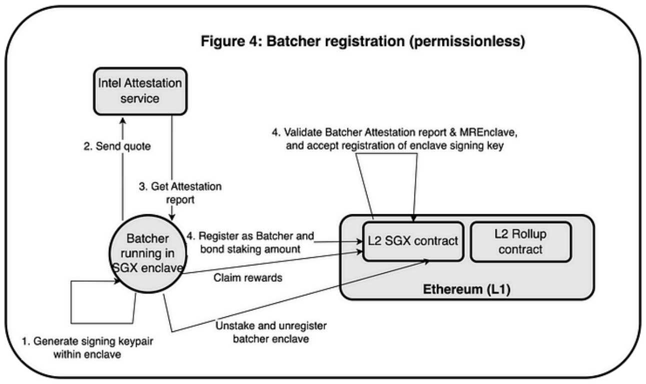


Figure 4: Batcher registration (permissionless)
Intel Attestation service
2. Send quote
3. Get Attestation report
4. Validate Batcher Attestation report & MREnclave, and accept registration of enclave signing key
Batcher running in SGX enclave
4. Register as Batcher and bond staking amount
L2 SGX contract
L2 Rollup contract
Claim rewards
Ethereum (L1)
Unstake and unregister batcher enclave
1. Generate signing keypair within enclave

*II. For each batch/block of data submitted from L2*

3. **Transaction batch and state root submission to L1**: Data submission from batcher with the ECDSA signature of in-enclave-generated batcher signing keys, is verified by L1 contract against the registration list. This is described in figure 5.

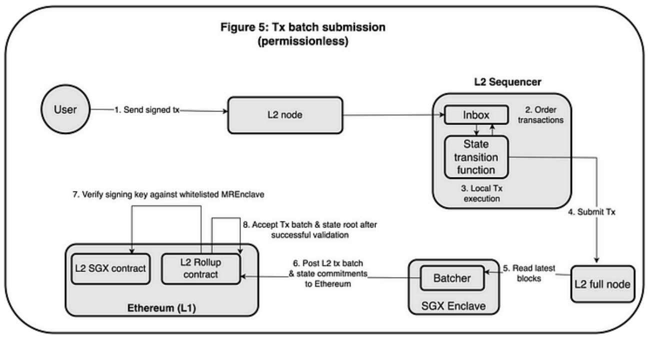


Figure 5: Tx batch submission (permissionless)
User
1. Send signed tx
L2 node
L2 Sequencer
Inbox
2. Order transactions
State transition function
3. Local Tx execution
4. Submit Tx
7. Verify signing key against whitelisted MREnclave
8. Accept Tx batch & state root after successful validation
L2 SGX contract
L2 Rollup contract
Ethereum (L1)
6. Post L2 tx batch & state commitments to Ethereum
Batcher
SGX Enclave
5. Read latest blocks
L2 full node

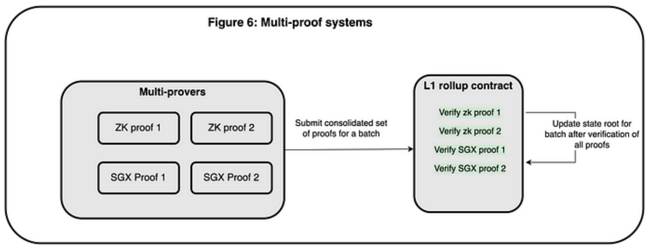
*III Finalisation*

4. **Unregistration**: The batcher operators can unregister from L1 contract, and choose to shut down the service. They will have to wait for a unlocking period before their staked ETH becomes withdrawable.
5. **Remove whitelisted MREnclave**: The L2 security council can decide to remove a given MRENCLAVE from the whitelist, and replace it with another measurement, or move to another proof system.

## Multi-proof architectures

Given that zk-proofs are still bleeding-edge tech, and optimistic proofs are based on hope-and-pray logic, TEE proofs can be used to supplement them in a multi-proof system.
Multi-proof generation can adopt a variety of technologies such as zk-snark, zk-stark, fraud proof and TEE proofs. However all of them ultimately are designed to prove the same L2 state commitments for a given batch or block posted to L1 rollup contract.

The process of generating proofs also is similar regardless of the type of proof. The prover should run a node for the L2 chain from which the necessary data needed to generate the proof is extracted. This can include transaction data, block parameters, merkle proofs, and in the case of zk proofs, execution traces of all the transactions in the block.



To verify proofs, the rollup contracts adopt different verification mechanisms based on the technology. In case of zk-proofs, zk-verifiers are used. For TEE proofs, verification of ECDSA signatures and proof expiry checks are performed.

Finally it is important that all proofs compute the same final state root hash, in order for the L1 rollup contract to update the L2 state root.

## Proof of batch

*I, Definition*

To improve the security model of our rollup, it is important to decentralize the batcher layer, and allow multiple operators to run batcher nodes. But how will they all agree on what date should be posted to L1, in case of disagreements?
This section introduces Proof-of-Batch, a consensus mechanism for multiple TEE-based batcher nodes to agree on the payload of a batch to be posted to L1.

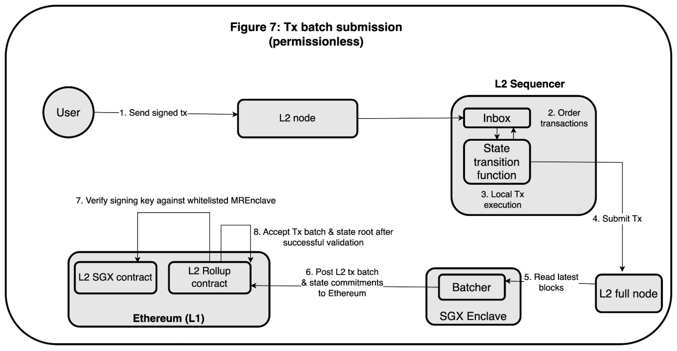


Figure 7: Tx batch submission (permissionless)
User
1. Send signed tx
L2 node
L2 Sequencer
Inbox
2. Order transactions
State transition function
3. Local Tx execution
4. Submit Tx
7. Verify signing key against whitelisted MREnclave
8. Accept Tx batch & state root after successful validation
L2 SGX contract
L2 Rollup contract
Ethereum (L1)
6. Post L2 tx batch & state commitments to Ethereum
Batcher
SGX Enclave
5. Read latest blocks
L2 full node

*II, Architecture*

*Consensus protocols* help to achieve agreements among a distributed set of nodes. There are three key aspects to consider in a consensus protocol design — network synchrony, crash fault tolerance and Byzantine fault tolerance.

*Network synchrony* refers to the degree of coordination of components in a distributed system. They are of three types -synchronous, asynchronous and partially synchronous.

*Crash fault tolerance* (CFT)is the ability of other components to detect faults in a participation component, and adjust their actions accordingly to continue providing the service as a whole unit.

*Byzantine fault tolerance* (BFT) is the ability to deal with and recover from a situation where a malicious node fails to act in accordance with the rules of the protocol.

The strength of a consensus protocol is measured by the number and type of faulty components it can tolerate. Crash-faults are easier to detect and address through SRE practices. But BFT is harder to detect and address. For any distributed system with N components, with f being byzantine, $N \geq 3f + 1$ is required to ensure consensus. So, to tolerate one byzantine component, network size should at least be 4. While proof-of-Batch does not adopt the classical BFT consensus protocol with multiple rounds of messages among participants, it still needs to detect and address byzantine behavior.

With this in mind, let's look into the design of the proof-of-Batch (PoB) consensus. Figure 8 illustrates the design of PoB.

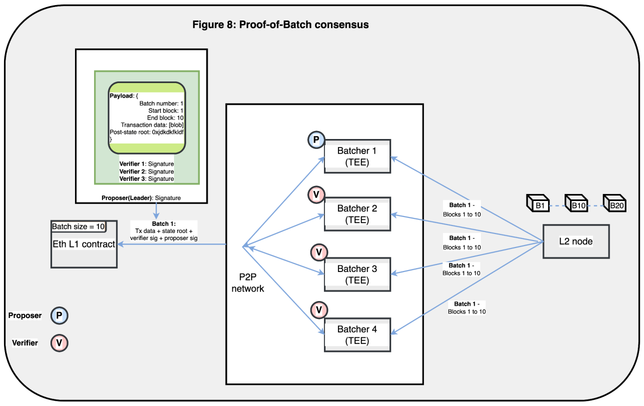


**Figure 8: Proof-of-Batch consensus**

*III, Consensus algorithm (informal)*

A descriptive version of the consensus algorithm can be found here:

1. **Batch Size**

A batch consists of a set of blocks, whose data is posted to L1 rollup contract (validating bridge). This is configured on the L1 contract.

2. **Batchers**

There are two parts to batcher nodes:

**Batcher operators**: The wallets that stake tokens to register as a batcher service provider.

**Batcher enclaves:** The software server that runs within TEE enclaves, which participate in the Proof-of-Batch consensus in order to post transaction data and state commitments to L1. The keypair used by the batchers to sign batches is generated within the enclave ( during the start up of each enclave, if the keys don't already exist), and even the batcher operator will not have access to the private key of the keypair. This private key generated is sealed and stored in the enclave, and is retrieved during the next startup of the node.

The registration process for Batcher nodes is described in Section "TEE proofs Implementation"

3. **Proof-of-Batch consensus**

For each batch, a leader (Proposer) is elected by the L1 contract. The proposer retrieves the set of blocks from L2 node for the next batch, creates the transaction batch , computes the state commitments, and constructs the Batch payload. The batch payload is broadcast to other TEE nodes in the network.

All the other active (non-faulty) nodes in the network perform the role of Verifier for the batch. They receive the batch payload constructed by the Proposer, and verify it. Once verified, they sign the payload and submit the signature to the Proposer.

Once the Proposer has the required number of verifier signatures (quorum), it attaches them to the Payload, and then signs the overall Batch packet. (Note in diagram, the transaction blob is shown as being part of the batch payload for ease of understanding. In reality, the transaction batch will be posted separately as a blob on L1, while the rest of the batch data with post-state roots and verifier signatures will be posted separately to the L1 contract.)

4. **Batch payload**

The batch payload posted to L1 consists of the following data elements:
*Batch number*: A sequentially- incremented integer
*Starting block*: The starting L2 block number for the batch
*Ending block*: The ending block number for the batch
*Transaction root*: The merkle root computed for the set of transactions in the batch
*Transaction batch*: In addition to the transaction root, the transaction batch with the call data is itself posted to L1 as a blob.
*State root:* The final state root (also called post-state root) arrived at by taking the starting state root, and applying the state transition function with the set of input transactions in the batch.
*Verifier signatures*: The set of verifier signatures obtained by the proposer for the batch is included in the batch payload.

*IV, Algorithm (semi-formal)*

A semi-formal description of the steps in the proof-of-batch consensus algorithm is defined below.

1. **Variables and Functions**:

*L2_Blocks*: Set of L2 blocks in a batch.
*Batch_Size*: Number of L2 blocks in a batch (configured in L1 contract).
*T_i,j:* Transaction data in the i-th L2 block and j-th transaction in that block.
*H*: Cryptographic hash function (e.g., SHA-256).
*Pre_State_Root:* State root before the application of state transition function for a set of input L2 transactions.
*Post_State_Root*: State root after the application of state transition function for a set of input L2 transactions.

*Batch_Number*: Identifier of the batch being posted to L1
*Starting_Block*: Block number of the first L2 block in the batch.
*Ending_Block*: Block number of the last L2 block in the batch.
*Transaction_Root*: Transaction Merkle root for the transactions included in the batch.
*Verifier_Signatures*: Set of verifier signatures for a batch.
*Threshold*: Threshold number of verifier signatures required for consensus.
*Leader*: Leader elected for a batch. also known as Proposer.
*Timeout_Duration*: Specified time limit for the proposer to send the proposed state and transaction roots to other nodes for a batch for which it is appointed as the leader.

## 2. Algorithm steps:

i. *Configuration*

Set Batch_Size as the number of L2 blocks in a batch. This is configured in the L1 contract.

Set threshold as the minimum number of verifiers that must agree on the batch payload for a given batch. This is configured in the L1 contract.

ii. *Batch Initialization:*

Set L2_Blocks as the set of L2 blocks for the current batch.

iii. *Leader Election:*

Elect a leader for the batch, who will be responsible for proposing the state and transaction roots.

iv. *Proposal Phase (Leader):*

The Leader computes the state and transaction roots for the batch and the initial batch payload.

a) **Transaction merkle root computation**: Compute the transaction Merkle root Transaction_Root from the list of transactions in L2_Blocks using the appropriate algorithm.

$$\text{Transaction\_Root}=\text{MerkleRoot}(T_{1,1},T_{1,2},\ldots,\text{Tn,m})$$

b) **Post-state root computation**: Compute the post-state root Post_State_Root by applying the transactions in L2_Blocks to the pre-state root using the appropriate algorithm.

$$\text{Post\_State\_Root}=\text{ApplyTransactions}(\text{Pre\_State\_Root},\text{L2\_Blocks})$$

c) **Batch payload construction**: Construct the batch payload including Batch_Number, Starting_Block, Ending_Block, Transaction_Root, and Post_State_Root.

$$\text{Batch\_Payload}=\{\text{Batch\_Number},\text{Starting\_Block},\text{Ending\_Block},\text{Transaction\_Root},\text{Post\_State\_Root}\}$$

Include the signatures of the verifiers Verifier_Signatures in the payload.

d) **Batch payload propagation**: The proposer propagates the computed batch payload to the other nodes in the network (verifiers)

v. *Verification Phase (Verifiers):*

The rest of the nodes play the role of verifiers for the batch. Each verifier independently computes the transaction and state Merkle roots using the same transactions and pre-state roots.

$$\text{Verifier\_Transaction\_Root}=\text{MerkleRoot}(T_{1,1},T_{1,2},\ldots,\text{Tn,m})$$

$$\text{Verifier\_Post\_State\_Root}=\text{ApplyTransactions}(\text{Pre\_State\_Root},\text{L2\_Blocks})$$

If the computed Transaction_Root matches the one proposed by the leader and the computed Post_State_Root matches the one proposed by the leader, the verifier signs the payload, and sends it to the proposer. $S_i = 1$

If not, the verifier does not sign. $S_i = 0$

*vi. Constructing the Final Batch Payload (Leader):*

The leader Leader constructs the final batch payload, including Batch_Number, Starting_Block, Ending_Block, Transaction_Root, Post_State_Root, and the signatures of verifiers Verifier_Signatures.

*vii. Sending to L1 Contract :*

The leader Leader signs the entire packet and sends it to the L1 contract if a threshold of verifier signatures is reached.

$$\text{Threshold\_Reached} = \sum_{n-1}^{i=1} S_i \geq \text{Threshold}$$

*viii. Leader Timeout:*

If the proposer Proposer does not send the proposed state and transaction roots to other nodes within the specified time limit Timeout_Duration, another node is assigned the role of the leader Leader.

*ix. Faults handling*

Different types of faults can occur in the network. Let's look at the different scenarios outlined in figure 9.

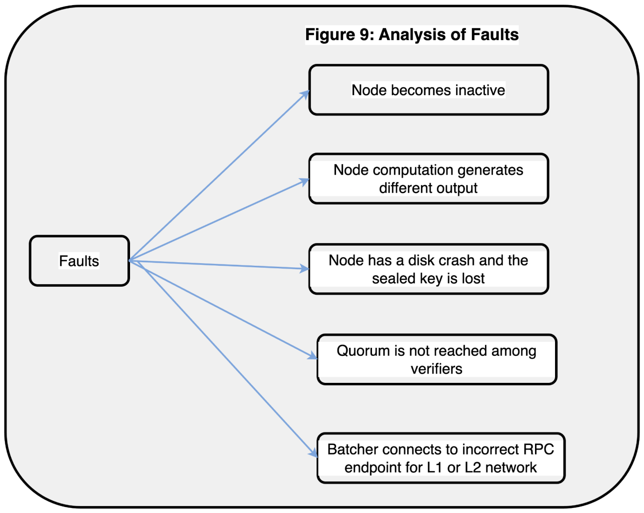


Figure 9: Analysis of Faults
Faults
Node becomes inactive
Node computation generates different output
Node has a disk crash and the sealed key is lost
Quorum is not reached among verifiers
Batcher connects to incorrect RPC endpoint for L1 or L2 network

i. One of the batcher enclaves (nodes) becomes inactive or goes offline: There can be two sub-cases here:

- If the inactive batcher node is the designated leader (proposer)for the next batch: After a timeout (configured in the L1 contract), the next leader is elected by the network.

- If the inactive batcher node is a verifier for the next batch: As long as the quorum of verifier signatures is reached for a batch, there would be no impact to the operations of the batcher network.

ii. One of the batcher enclaves (verifier) computes a different batch payload, as part of the verification: The verifier signs on the payload it computed, and submits it to the proposer. The proposer includes this verifier signature (with a different payload) in the batch payload to L1. The L1 contract verifies each signature, and will detect this signature on invalid payload, and ignores this verifier signature as long as the other verifier signatures comprise the quorum. Otherwise, the entire batch is rejected.

iii. The in-enclave generated key is lost (e.g., due to disk crash on the TEE VM): In such a case, the operator starts the batcher enclave in recovery mode. In this mode, the enclave generates a new keypair , and makes a request to the Ternoa Key management system to delegate access to the secret NFT containing the private key to itself. After verification, the enclave is provided with delegate access, and the enclave retrieves the key from the Ternoa enclave cluster. Note that all these operations occur within the enclave, and even the enclave operator will not have access to the recovered private key.

iv. Quorum is not reached among the verifiers on the payload for a batch: In this case, the current batch (n) is put on hold. When it’s time for batch n+1 , a new batch is computed including the blocks of both batch n and batch n+1, and again an attempt is made to obtain a quorum of signatures from the verifiers. In an extreme case, if quorum is not reached even after multiple attempts, the security council can intervene.

v. DNS record of the RPC node that the enclave is configured to connect to (for either L1 or L2 network) goes under control of an unauthorized person: To prevent this, the genesis hash of both the L1 and L2 networks are baked into the enclave code. While starting the batcher enclaves, the operator would be given the option to specify the RPC node endpoint. the batcher enclave, on startup , compares the built-in genesis hash with that obtained from the RPC endpoint provided by the operator during startup. If these two don’t match, the enclave will not start.

### *V, Observations*

i. **Cost optimisation**
If a batch n does not contain any transactions from L2 nodes, the proposer can choose to not submit the batch, but instead create a consolidated batch that contains blocks from batch n and n+1. This is to optimize the cost of posting data to L1.

ii. **Batcher operator rewards**
As part of each batch submission, the proposer includes the signatures of all verifiers. This also serves to record a list of verifiers that are not active for any given batch. This data is used for calculation of rewards and penalties, to incentivise good behavior for the consensus protocol.

iii. **Inter-Batcher communications**
Since batcher nodes run in TEE enclaves, the messaging between them involves mutual authentication involving both their SGX credentials and their registration status with the L1 contract. Further, the enclave communications are over a secure TLS channel.

## Economical Concepts

### *I, Economic Security*

Token staking on a decentralized network represents a fundamental shift towards aligning the incentives of network participants with the overall health and security of the system. In this model, participants, or validators, lock up a certain amount of the network's native tokens as a form of collateral to participate in the network's consensus mechanism. This process is pivotal in maintaining the network's integrity, as it ensures that validators have a vested interest in acting honestly. The staked tokens act as a security deposit that can be forfeited in cases of malicious actions or failure to comply with the network's protocols, making attacks financially impractical. This system of economic staking mitigates the risk of centralization and enhances network security by distributing the power among a wider group of validators, thus preventing any single entity from gaining disproportionate control. Additionally, it provides a mechanism for rewarding participants who contribute positively to the network, further incentivizing proper behavior. The concept of economic security via staking is supported by Buterin's (2014) discussion on proof of stake in decentralized networks[10], which highlights the method's efficiency and security benefits over traditional proof of work systems ("Ethereum Whitepaper", Buterin, 2014). Furthermore, research by Kiayias et al. delves into the mathematical underpinnings that ensure the security and viability of staking mechanisms, offering a rigorous framework for understanding their contribution to decentralized network security[11].

Token staking can be applied to individual operators of a decentralized network of TEEs, collaborating to provide TEE proofs. A first implementation of this concept was developed and implemented to secure the decentralized key management system of Ternoa, a Layer 1 blockchain[12].

### *II, Staking delegation*

Delegated token staking on a decentralized network introduces an innovative layer of flexibility and efficiency in achieving consensus and maintaining network security. In this model, token holders can delegate their staking power to operators of their choice, who are then responsible for participating in the consensus mechanism on their behalf. This delegation process allows for a more inclusive participation in the network's governance, as it lowers the barriers for token holders who may not have the resources or expertise to run a validator node themselves. Moreover, it enhances network security by aggregating staking power to competent and reliable validators, thereby making it more difficult for malicious actors to compromise the network. Delegated staking also fosters a competitive environment among validators to offer better services and security, as they must earn the trust of token holders to receive delegations. The economic incentives are aligned in such a way that validators are rewarded for their integrity and performance, further underpinning the network's resilience and reliability. Research by Bentov extends theoretical support for the efficiency of delegated staking models, highlighting their potential to reduce energy consumption compared to traditional proof of work systems, while still maintaining robust security mechanisms[7]. Additionally, the practical implementation and success of delegated staking in networks like Tezos and Cosmos demonstrate its viability and effectiveness in enhancing decentralized network security and governance[8].

*III, Infrastructure-as-a-Service*

Token-based Infrastructure as a Service (IaaS) models represent a paradigm shift in how computational resources are provisioned and managed, leveraging blockchain technology to decentralize infrastructure services. In this innovative model, blockchain tokens are utilized both as a means of exchange and an access control mechanism, allowing users to purchase, sell, or rent computational resources within a decentralized network. This approach not only democratizes access to infrastructure services but also introduces a novel economic layer that incentivizes the provision of high-quality, reliable services by participants. One of the key benefits of a token-based IaaS model is its ability to facilitate a marketplace without centralized intermediaries, where prices for services are determined by supply and demand dynamics within the network. Furthermore, the use of blockchain technology ensures transparency, security, and auditability of transactions. Projects like Golem and iExec have pioneered this approach, demonstrating the feasibility and efficiency of decentralized cloud computing services[18]. These platforms enable users to monetize their idle computing power or access additional resources on-demand, effectively creating a distributed supercomputer. The token-based IaaS model not only challenges traditional centralized cloud services but also paves the way for a more resilient, efficient, and user-centric digital infrastructure.

Rollup framework stacks users (such as Polygon, zk sync, Starknet) could opt-in to decentralized services as described in this document by registering their service images along with their cryptographic measurements.

External service providers would opt-in to run rollup services (from one of the rollup framework stacks) by registering their TEE machines along with their proof-of-authenticity (aka remote attestation reports), and agree to participate in a system of on-chain incentives and penalties with economic bonding.

Web3 projects could launch their own rollup by choosing one of the compatible rollup framework stacks and deploying them on one of the available external service providers, based on a pay-per-rollup approach. The benefit for web3 projects is that they do not have to trust their own DevOps team or the rollup-as-a-service operators for integrity and security of their rollups, and can instead outsource the management of their rollup infrastructure to untrusted external service providers, while remaining assured that the transactions, state and code associated with their rollup are not modified, censored or tampered by the operators.

## Summary

In this paper we have discussed the technical design of TEE proofs as a way to verify integrity of computations performed within a Trusted Execution Environment. In addition, we also looked at how to construct a protocol for consensus of computation results among a network of TEEs, along with a few considerations for implementation.

This paper is intended as a first step towards the development and implementation of a prototype decentralized TEE prover. This application will be merging technical and economical concepts explored herein. Possible business models associated are out of scope of this document.

## References

[1] Victor Costan and Srinivas Devadas. “Intel SGX explained.” , MIT.
[2] McCorry, P., et al. (2021). "SoK: Rollups and their discontents." https://arxiv.org/abs/2101.00000
[3] Al-Bassam, M. (2018). "Fraud and data availability proofs: Maximising light client security and scaling blockchains with dishonest majorities." https://arxiv.org/abs/1809.09044
[4] Buterin, V. (2014). Ethereum Whitepaper. https://ethereum.org/en/whitepaper/
[5] Kiayias, A., Russell, A., David, B., & Oliynykov, R. (2017). Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol. In Annual International Cryptology Conference (pp. 357-388). Springer, Cham
[6] Ternoa Fortress developed by Capsule Corp. teams, docs.ternoa.network
[7] Bentov, I., Lee, C., Mizrahi, A., & Rosenfeld, M. (2016). Cryptocurrencies Without Proof of Work. In International Conference on Financial Cryptography and Data Security (pp. 142-157). Springer, Berlin, Heidelberg.
[8] Goodman, L. (2014). Tezos: A Self-Amending Crypto-Ledger White Paper. https://tezos.com/tezos-whitepaper.pdf
Kwon, J. (2016). Cosmos: A Network of Distributed Ledgers. https://cosmos.network/cosmos-whitepaper.pdf
[9] (Zamfir et al., 2017; Sillaber & Waltl, 2018)
[10] Sillaber, C., & Waltl, B. (2018). "Life cycle of smart contracts in blockchain ecosystems." Datenschutz und Datensicherheit - DuD, 42(8), 497-500.