Concilium: More nodes, more scalability, decentralization and security

v0.1.0

Amir Faramarzi
amir@concilium-labs.org

Abstract. This paper presents a new Byzantine Fault Tolerant consensus mechanism, called Proof of Council (POC), which, with support for horizontal scalability, can process millions of transactions per second in parallel. POC assigns the task of processing each transaction to a small, randomly selected group in the network, referred to as the "Validation Council." If the consensus result of these councils approves a transaction, the transaction is not considered final and irreversible from the network's perspective, as it may be rejected in the second consensus stage, which is performed by the entire network. However, from the user's perspective, the consensus result of these councils is considered final and irreversible, since the compensation protocol, in the event of such a situation, obligates the council members to compensate the affected user with an amount of coins equal to the amount transferred in the transaction.

1 Introduction

With the introduction of Bitcoin, distributed ledgers emerged as an innovative solution for establishing trust without the need for intermediary institutions. This technology enables the transparent, secure, and censorship-resistant recording and verification of transactions, paving the way for a wide range of applications in finance, supply chain management, digital ownership, and other domains.

Despite significant advancements in blockchain infrastructure and algorithms, scalability remains one of the primary barriers to the widespread adoption of this technology. One fundamental reason for this challenge is what Vitalik Buterin, the founder of Ethereum, has described as the "Blockchain Trilemma." This concept states that no blockchain system can simultaneously achieve scalability, decentralization, and security in full. In other words, the design of any blockchain system inevitably requires a balance or trade-off among these three core elements.

Moreover, increasing the number of nodes in blockchain networks often leads to reduced performance. This is due to the growing complexity of the consensus process, which requires exchanging a large volume of messages and coordinating among a greater number of participants.

Another challenge, which has received less attention, is that most blockchains are not well-suited for everyday applications. For example, using blockchain for daily purchases or commercial deals faces limitations such as high transaction finality times. In many cases, these times are significantly longer than those of traditional banking payment systems, leading to a poor user experience. Additionally, an increase in transaction volume can reduce network stability or even cause it to fail.

In recent years, solutions such as second layers, sharding, and increasing the level of centralization have been proposed. While these approaches have proven useful in certain contexts, many of them conflict with fundamental principles such as decentralization, transparency, and censorship resistance — principles that form the foundation of blockchain technology — or require renewed trust in new intermediaries.

This whitepaper introduces the architecture of a novel distributed ledger that seeks to overcome existing limitations through a different approach, providing an infrastructure that, in terms of efficiency, security, and decentralization, can serve as a viable option for everyday use at a large scale.
The remainder of this paper is structured as follows:
Section 2 explains the process of recording and organizing transactions, Section 3 provides an overview of the network architecture, Section 4 introduces the concept of epochs, Section 5 examines Proof of Council, Section 6 describes the finalization process, Section 7 analyzes the compensation protocol, and finally, Section 8 is dedicated to computations.

2 Transaction Recording and Organization

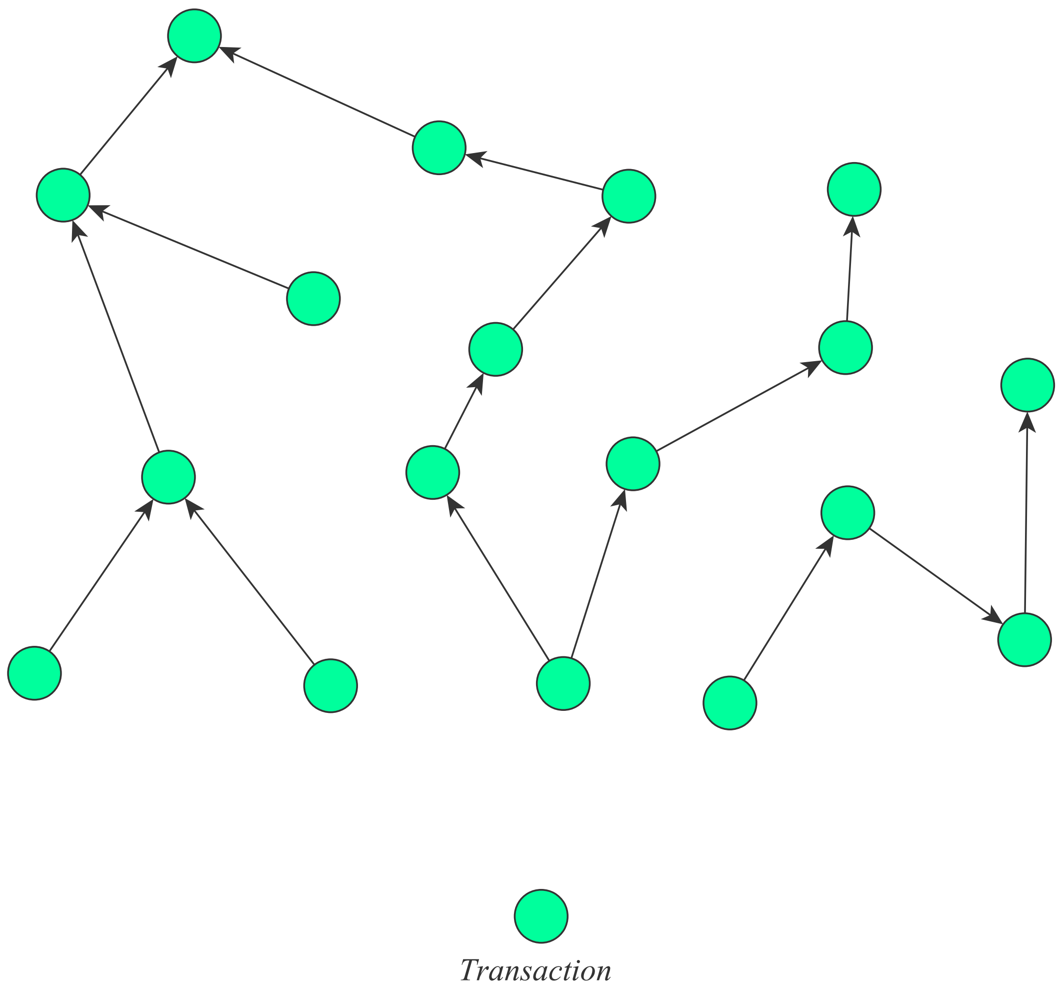

Figure 1: How transactions are stored

As illustrated in Figure 1, in this network—unlike traditional blockchains—it is not blocks that are linked together in a chain; rather, it is a directed acyclic graph (DAG) in which transactions are directly connected to each other without sequential dependencies. In this architecture, each transaction can reference multiple previous transactions and be confirmed without waiting for the creation of a new block [1]. This approach eliminates issues such as forks and parallel processing limitations, while reducing finalization time, thereby creating ideal conditions for networks that require high-throughput, concurrent, and scalable transaction processing.

The network employs the Extended Unspent Transaction Output (EUTXO) model [6] to record and manage transaction states. This model is an advanced version of the UTXO system first introduced in Bitcoin, enhanced with features such as data storage and embedding smart contract logic within each output, enabling the implementation of smart contracts.

In the EUTXO model, each transaction consumes a set of inputs and produces a set of new outputs [5]. Each output can be spent only once and may contain data and scripts that define the conditions under which it can be spent. This characteristic allows transactions to be processed in parallel, significantly improving the network's throughput.

Choosing the EUTXO model not only enhances security against double-spending, but also enables high scalability, reduces transaction conflicts, and provides a robust foundation for smart contracts. These advantages make it an ideal choice for networks that require processing large volumes of transactions.

3 Overview of Network Architecture

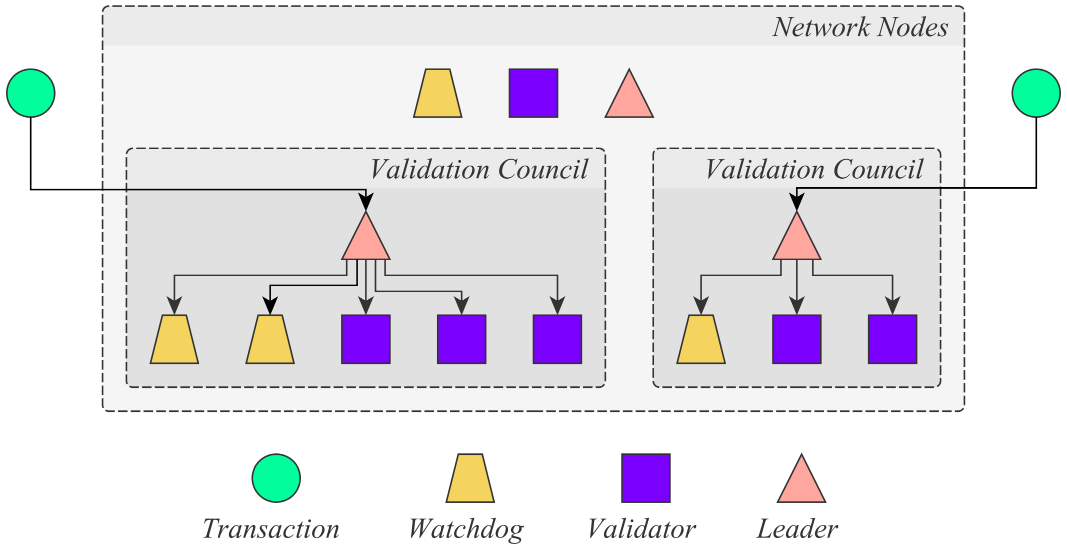

Figure 2: Network architecture

As shown in Figure 2, each transaction is assigned to a validation council consisting of three main roles:
Council Leader: Responsible for coordinating among members.
Validators: Tasked with verifying the validity of the transaction.
Watchdog: Preventing council fraud and misconduct.

After the consensus process is completed, the council leader broadcasts the final result to the entire network for finalization.

The number of validation council members for each transaction may differ from that of other transactions; the reason for this will be explained in later sections. Additionally, some network nodes may, at certain times, not belong to any council and remain in a standby state until a transaction deems them eligible to participate in validation.

4 Identifiers

In this network, each node has two types of identifiers: a primary identifier and a temporary identifier.
Upon joining the network, a new node is assigned a unique primary identifier, which remains constant until the node leaves the network.
In addition, each node also has a temporary identifier. This identifier plays a decisive role in the process of selecting the nodes that must participate in the consensus operation on a specific transaction.
To maintain network security, the temporary identifiers of all nodes are changed in each epoch. The details regarding the structure and scheduling of these epochs will be explained in the next section.

4.1 Epochs

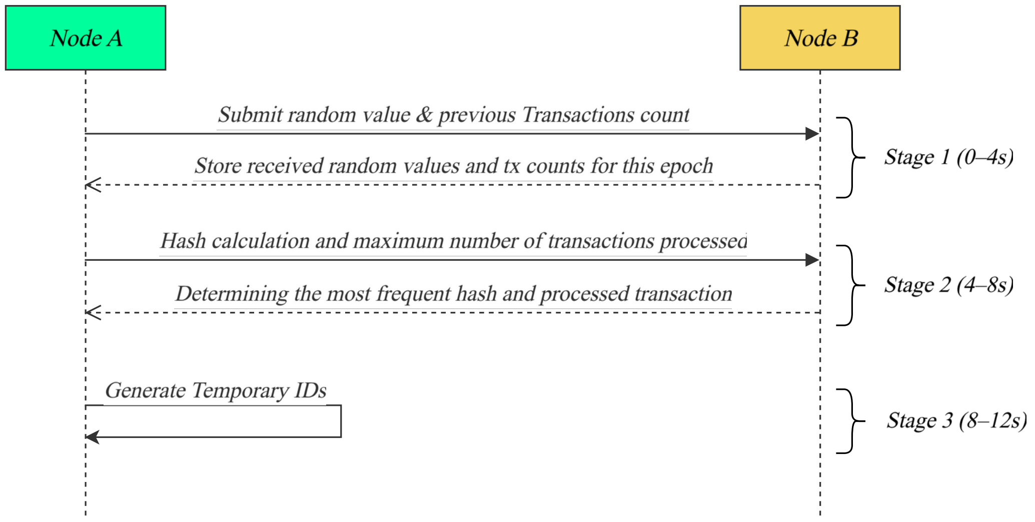
Node A
Node B
Submit random value & previous Transactions count
Store received random values and tx counts for this epoch
Stage 1 (0–4s)
Hash calculation and maximum number of transactions processed
Determining the most frequent hash and processed transaction
Stage 2 (4–8s)
Generate Temporary IDs
Stage 3 (8–12s)

Figure 3: Epochs

As explained in the section "Identifiers", the temporary identifier of all nodes is updated at the beginning of each epoch. The duration of each epoch is 12 seconds, and as shown in Figure 3, every epoch is divided into three consecutive phases, each executed within a 4-second interval. These phases are responsible for determining the new identifiers for the next epoch.
The beginning of each new epoch serves as a reference point for synchronizing all nodes in the network. To ensure full coordination, all nodes are required to participate in the recurring and predefined processes of each epoch.

Each epoch includes three key parameters:
- Epoch number
- Epoch identifier
- Number of transactions processed in the previous epoch

The synchronization phases in each epoch are executed as follows:
Recording random values and the number of transactions from the previous epoch
In the first four seconds, each node sends a random number along with the number of transactions it processed in the previous epoch to the other nodes in the network. At the same time, each node receives similar data from other nodes and records them for the same epoch.

Recording the most frequent hash and the maximum processed transactions
In the middle four seconds, each node sums the random values received from other nodes in the previous phase with its own generated random value and then hashes the result using the SHA256 algorithm. Afterwards, the node sends this hash together with the largest number of processed transactions received from other nodes in the previous phase to the entire network.
In turn, upon receiving similar data from other nodes, each node maintains a counter for each hash value and for each maximum processed transaction value it receives. Each time the same value is received again from a different node, the corresponding counter is incremented.
The goal of this process is to identify the hash and transaction count that appear most frequently across all nodes in the network. Due to potential delays in data transmission, some information from the previous phase may not have reached all nodes. Therefore, with each newly received value (hash or maximum transaction count), the corresponding counter is updated so that, by the end of this phase, full synchronization is achieved and the entire network converges on a single hash and a single maximum processed transaction count.

Computing temporary node identifiers for the next epoch
In the final four seconds of each epoch, the temporary identifiers of nodes for the next epoch are computed. This process uses the final hash (the most frequent hash) and a pseudorandom number generator algorithm called ChaCha20.
First, the final hash is used as input to the ChaCha20 algorithm. The output of this algorithm is a sequence of pseudorandom numbers generated in the same quantity as the number of active nodes in the network. Each output value falls between 1 and the maximum primary identifier of the network's nodes. This output is stored as an array, and its ordering serves as the basis for determining the temporary identifiers.
In the next epoch, each node's temporary identifier is determined from this array based on the index of its primary identifier. Put simply, the position (index) of the node's primary identifier (minus one) in the array specifies its temporary identifier.
For example, suppose the final hash equals a5w9...87h8 and the network contains 500 active nodes. This hash is fed into the ChaCha20 algorithm, which then produces 500 random numbers ranging from 1 to 500. Suppose Node A has the primary identifier 10. To determine the temporary identifier of Node A in the next epoch, one only needs to read the value at index 9 of the output array. That value will be the temporary identifier of Node A in the next epoch. This process is applied in the same way for all nodes in the network.

ChaCha20 has been chosen because it maintains a balance between speed and security compared to many similar algorithms; some algorithms focus only on security and some only on speed, but ChaCha20 provides both features simultaneously.
Pseudorandom number generator algorithms such as ChaCha20 take a seed as input and produce an unpredictable sequence of numbers as output. An important property of these algorithms is that given a specific seed, the same output is always repeated [8].

In this paper, a rewritten version of ChaCha20 is used as follows:

```
chacha20(hash, to, how_many)
```

hash: the first input, always a SHA256 hash, serving as the seed.
to: the second input, defining the upper bound of the output numbers (numbers between 1 and to).
how_many: the third input, specifying the number of values to be generated in the output.

Throughout this whitepaper, whenever ChaCha20 is mentioned, it refers to this rewritten version.

5 Proof of Council(POC)

POC is a stake-based consensus mechanism that shares structural similarities with POS (Proof of Stake), but also incorporates fundamental differences.
In this method, all nodes in the network must stake a fixed amount of coins in order to participate in the consensus process. Unlike POS, where the voting power of each node is proportional to the amount staked [4], in POC all nodes stake the same amount of coins and therefore have equal voting rights.
The transaction validation process in POC works such that each transaction is assigned to a small council consisting of a number of nodes selected from the entire network.
The selection of council members for each transaction is based on the transaction's own data and is performed using ChaCha20, a cryptographically secure pseudo-random number generator (CSPRNG). This approach ensures that no node or group can predict in advance which transactions will be assigned to them, thereby minimizing the risk of collusion or targeted attacks.
One of the key advantages of POC is its horizontal scalability. Since transaction processing is carried out in parallel among different councils, the transaction throughput increases linearly as the number of network nodes grows. Furthermore, an increase in the number of nodes directly enhances both the security and the decentralization of the network.

5.1 Council Member Selection

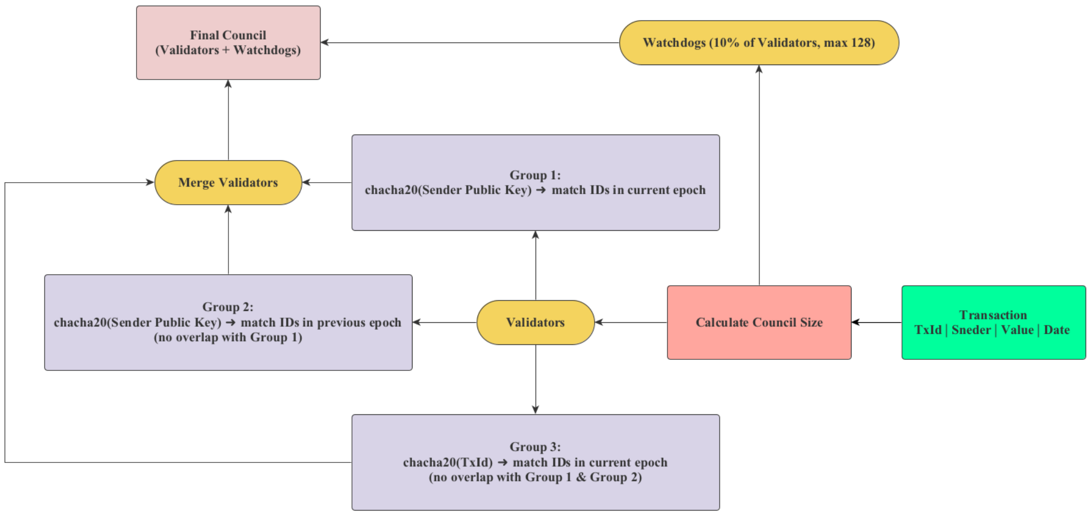
Final Council
(Validators + Watchdogs)
Watchdogs (10% of Validators, max 128)
Merge Validators
Group 1:
chacha20(Sender Public Key) ➔ match IDs in current epoch
Group 2:
chacha20(Sender Public Key) ➔ match IDs in previous epoch
(no overlap with Group 1)
Validators
Calculate Council Size
Transaction
TxId | Sneder | Value | Date
Group 3:
chacha20(TxId) ➔ match IDs in current epoch
(no overlap with Group 1 & Group 2)

Figure 4: Council Member Selection

As shown in Figure 4, the process of selecting council members is based on the information contained within the transaction itself. The number of council members depends on the amount of coins being transferred by the transaction. For example, one transaction may require a council with 250 members, another with 600 members, or in special cases, it may even require the participation of all network members in the consensus.
However, each council consists of at least 140 members, including 128 validators and 12 watchdogs.

To determine the council members, three components of each transaction are required:

- Transaction timestamp
- Sender's public key
- Transaction ID

The process of determining council members is carried out in the following order:
Determining the Number of Members
In the first step, it must be established how many members the transaction's validation council will have. For this purpose, the following formula is used:

$$\frac{\text{Transactions processed in the previous epoch} \times \text{Number of coins being transferred} \times 24}{\text{Number of coins staked by nodes}}$$

The output of this formula specifies the number of validators. Then, to determine the number of watchdogs, 10% of the validators are added as watchdogs, with the restriction that the number of watchdogs cannot exceed 128 members.

For example, suppose a user intends to transfer 20 coins, the number of transactions processed in the previous epoch equals 100, and each node has staked 100 coins. By applying these values to the formula, it is determined that validating this transaction requires 480 validators and 48 watchdogs. Consequently, the total number of council members for this transaction will be 528.

Retrieving Epoch Data
At this stage, the epoch corresponding to the transaction's timestamp, as well as the epoch immediately preceding it, are retrieved. From this point on, they will be referred to as the "current epoch" and the "previous epoch," respectively.

Partitioning and Group Formation
The output of the first step (the total number of council members) is divided into three parts to form three separate groups. Each validator is included in only one of these groups. After forming the three groups, they are merged to create the final council.

To select the members of each group, the ChaCha20 function is used. The only difference between the groups lies in the input hash provided to the function. The parameters to and how_many are set equal to the largest main node identifier and the number of active nodes in the network, respectively.

Group One – Sender's Public Key – Current Epoch
To determine the members of the first group, the sender's public key of the transaction is used as input to the ChaCha20 function. The output of this function generates an array that is compared with the list of temporary identifiers from the current epoch. Any node whose temporary identifier (in the current epoch) appears in this array is selected as a member of the first group.
The first element (index zero) of this array is designated as the leader of the validation council.

Group Two – Sender's Public Key – Previous Epoch
The selection process for the second group is similar to the first, except that the output of the ChaCha20 function is compared against the temporary identifiers of the previous epoch. Any node whose temporary identifier from the previous epoch appears in the output array is added to this group.
If there is an overlap between the members of the second group and the first group, the how_many parameter is increased by one, and the selection process is repeated until no duplicate members remain.

Group Three – Transaction ID – Current Epoch
The selection of the third group follows the same procedure as the first, except that the transaction ID is used as input to the ChaCha20 function instead of the sender's public key. The output array is then compared with the temporary identifiers of the current epoch, and nodes whose identifiers appear in this array are selected.
Members of this group must not overlap with the members of the first and second groups.

Watchdogs are selected after the validators reach consensus and based on their aggregated signature. The details of this process are explained in the section "Consensus in the Validation Council."

5.2 Consensus in the Validation Council

The consensus process in the Validation Council operates based on a simple principle: all validators within the council must approve the transaction, and all watchdogs of the council must be informed of the consensus result for the transaction to be recognized as valid by the network. If even a single validator rejects the transaction, or if any watchdog remains uninformed of the consensus result, the transaction will be considered invalid by the network, and no responsibility will be assumed for compensating the user's loss.

The consensus process in the Validation Council is executed in the following steps:
Sending the transaction to validators
The leader of the Validation Council first reviews the transaction and, if approved, identifies the validator members of the transaction using the council selection unit (described in the Council Member Selection section) and sends the transaction to them.

Review and signing by validators
Each validator evaluates the transaction and, if valid, signs it using BLS [3] as proof of approval and returns the signature to the council leader.

Aggregation of validator signatures
After receiving responses from all validators, the leader verifies the validity of the signatures. If all signatures are present and valid, the leader also signs the transaction and aggregates all the signatures into a single unified signature [2].

Watchdog selection
To notify the watchdogs, the leader must first identify them. This is done through the following steps:

1. Placing the aggregated validator signature alongside the current epoch hash and computing its SHA256 hash.
2. Using the resulting hash output as the input for the ChaCha20 algorithm to generate an array of identifiers (equal to 10% of the number of validator members).
3. Selecting the nodes whose identifiers match the ChaCha20 output in the current epoch as watchdogs.

Notification to watchdogs
The council leader sends the transaction and the aggregated validator signature to the watchdogs. The responsibility of the watchdogs is not to verify the validity of the transaction, but simply to sign the received message (transaction + aggregated validator signature) using BLS and return their signature to the leader.

Aggregation of watchdog signatures
After receiving the signatures from all watchdogs, the leader aggregates them into a single signature as well.

Finally, the council leader sends the transaction ID along with the two aggregated signatures to the user who submitted the transaction.
These signatures allow the user to claim compensation from the Validation Council if the transaction is rejected during the finalization stage.

5.3 Watchdogs

As explained in the section “Consensus in the Validation Council”, watchdogs do not play a direct role in approving or rejecting transactions. Their primary responsibility is to be present in the validation process as monitors, ensuring that members of the Validation Council do not engage in malicious behavior.

In some cases, council members may intentionally attempt to undermine the security of the network by approving a transaction that should be rejected, or conversely, rejecting a transaction that should be approved. They may also withhold a transaction from being broadcast to the network in order to avoid penalties or compensation to a user, effectively acting as if such a transaction never existed.

In these situations, watchdogs who were present during the validation process step in and report the full details of the transaction and its outcome to the entire network. This action enables the network to detect malicious behavior, penalize malicious validators, and, if necessary, compensate affected users.

5.4 Self-Validation of Signatures

As explained in the section “Consensus in the Validation Council,” the two returned signatures after a transaction is confirmed are critical data, since users can rely on them to claim compensation in case of an issue.

Because transactions are processed in a council-based manner, although the probability of malicious behavior by Validation Council members is very low, it is not absolutely zero. In rare cases, some members may act unlawfully or attempt to disrupt the network.

To mitigate this risk, after the transaction has been confirmed by the Validation Council and the two signatures are received, the user can personally verify the validity of the signatures or delegate this task to a trusted node in the network. The purpose of this verification is to ensure the authenticity of the signatures and to guarantee the ability to claim compensation if the transaction is rejected during finalization or never recorded in the ledger.

The steps of self-validation by the user are executed as follows:

Identification of Validators

Using the method described in the section “Council Member Selection,” the user first identifies the validators responsible for their transaction.

Aggregation of Validators’ Public Keys

The public keys of all identified validators must be aggregated into a single aggregated public key.

Validation of Validators’ Signatures

It is checked whether the signatures provided by the council’s validators were indeed generated by the aggregated public key (i.e., the collective signature of all members).

Identification of Watchdogs

Next, the user identifies the watchdogs associated with their transaction using the method described in the section “Consensus in the Validation Council,” and aggregates their public keys as well.

Validation of Watchdogs’ Signatures

The aggregated signature of the watchdogs is compared against their aggregated public key to confirm the presence of all watchdogs in the consensus related to the transaction.

If both signatures are valid, the user can be confident that even if the transaction is rejected during finalization or removed from the network, the ability to claim compensation remains intact.

If the user chooses to delegate the verification process to a trusted node, that node performs exactly the same procedure and reports the validation result back to the user.

6 Finalization

In POC, similar to many distributed ledgers, the finalization of transactions requires the consensus of the entire network.
Every 2 seconds, network members are obligated to send to the entire network a package containing the set of transactions that have been validated under their leadership.
Upon receiving the package, network members review the transactions within it, and if at least $\frac{2}{3}$ of the network members agree, the transactions are definitively recorded and become non-removable from the ledger.
If the leader fails to deliver the transaction packages, the watchdogs intervene and broadcast to the network the transactions they have received from users but which were not published by the leader, ensuring that the finalization process for those transactions is also carried out.

7 Compensation Protocol

The Compensation Protocol is a mechanism that allows users, in cases where their transaction has been approved by the Validation Council but has not been recorded in the ledger, or is rejected at the finalization stage, to claim the equivalent amount of assets stated in that transaction as compensation from the balance of the members of that same Validation Council.
In most cases, when such a situation occurs, the network watchdogs report the misconduct or fraud to the entire network. However, if the watchdogs do not provide such a report, the user will have up to 7 days to personally submit their transaction, along with the two aggregated signatures received from the Validation Council, to the network and request compensation.
For example, suppose "Bob" sends 100 coins to "Alice" in a transaction. This transaction is approved by the Validation Council, but after one epoch, Bob or Alice realizes that the transaction does not exist in the network or was rejected during the finalization stage, and this misconduct was not reported by the watchdogs. In this case, Bob, Alice, or any other person who holds a copy of the transaction along with its two valid aggregated signatures can report this misconduct to the network.
Once the report is received, the network members will verify the validity of the transaction and signatures. If confirmed, the equivalent of 100 coins will be deducted from the balances of the Validation Council members who approved that transaction and will be credited to the destination address of the transaction (in this example, Alice).

7.1 Compensation in Smart Contracts

Smart contracts are also covered under the compensation protocol, and such compensations are paid through the network's coin.
The main challenge in this area is that the network cannot determine the exact value of the functions executed on a smart contract. Therefore, in the event of a loss, it is unclear how much compensation should be paid to the user.
To address this issue, the user is required to specify the value of the intended operation in terms of the network's coin when invoking any function of a smart contract. This specified value serves as the basis for calculating potential compensation, and if needed, the network can reimburse the user accordingly.
Naturally, the number of validation council members responsible for executing the transaction related to a smart contract function is determined based on this user-declared value. This approach aligns the cost of executing an operation with the level of security and compensation guarantee associated with it.

8 Attacks

Due to the nature of the POC architecture, two types of attacks are considered more likely than others:

- Forged transaction generation
- Double spending

The following sections describe how these attacks can be carried out and the mechanisms POC employs to counter them.

Attack via Forged Transaction Generation

This attack has been the primary security challenge for POC since its design. In this scenario, a group of malicious nodes attempts, through brute force, to generate and submit transactions to the network where the Validation Council consists solely of themselves. If successful, they would be able to:

- Create and approve forged transactions,
- Withhold the result from the network or submit a false result,
- Then request compensation as the "victim," even if the compensation exceeds their own staked coins.

This scenario can place the network in an uncertain state, as it becomes unclear whose account should bear the compensation.

To counter this attack, POC incorporates several security layers:

Layer One – High Computational Difficulty

Even if $\frac{1}{3}$ of the network members are malicious and act in full coordination, they would need to examine approximately 10^{69} different transactions to find one with at least 128 Validation Council members (all malicious). This must be accomplished in less than 16 seconds, since after this interval, the identifiers of all network members are refreshed.

Layer Two – Unpredictable Watchdogs

POC uses the aggregated signature of the Validation Council to determine watchdogs. This feature ensures that attackers, even if they successfully assemble their desired council, cannot know the identity of the watchdogs until the transaction has been signed.

Layer Three – Compensation Capacity Formula

The formula for determining the number of Validation Council members is designed to guarantee that even if a council cheats in validating transactions across two consecutive periods, it will still be capable of compensating all affected transactions. Meanwhile, if a node commits multiple violations within a single period, it will be temporarily frozen and barred from participating in transaction validation councils until the network reaches a decision.

Furthermore, the structure of this formula is such that the greater the value attackers attempt to move in a forged transaction, the more computational power they must expend. As a result, attempts to execute such attacks grow exponentially more costly.

Double Spending Attack

As explained in the section "Council Member Selection," even the smallest modification to a transaction alters the composition of its Validation Council. This property, along with the 2-second interval for submitting transactions to the network for finalization, can make POC vulnerable to two types of double-spending attacks.

Case One – Simultaneous Transaction Submission

An attacker places a single EUTXO into two separate transactions before broadcasting them and submits both simultaneously to the network. Because the TxId differs between the two, each transaction receives a different Validation Council, and the councils remain unaware that the same EUTXO is being spent again.

Countermeasure: $\frac{1}{3}$ of the Validation Council members for each transaction are selected based on the sender's public key. This ensures that if two transactions are submitted simultaneously, this fixed $\frac{1}{3}$ can easily detect the double spend of the EUTXO.

Case Two – Submission Across Period Boundaries

An attacker places an EUTXO into two transactions, sending one in the final second of the current period and the other in the first second of the next. Due to the change in member identifiers between periods and the 2-second interval for transaction propagation and finalization, both transactions may be validated by their respective councils.

Countermeasure: $\frac{1}{3}$ of the Validation Council members for each transaction are selected based on the sender's public key from the previous period. This ensures that part of the council remains constant across two consecutive periods, allowing them to recognize and block the double spend.

9 Calculations

In this section, we examine a scenario in which an attacker or a group of attackers controls $\frac{1}{3}$ of all network nodes and intends to create a fraudulent transaction. The attacker's goal is to construct a transaction in which all members of the Validation Council (with a minimum size of 128 nodes) are under their control. For this purpose, it must be determined how many transactions the attacker, on average, has to check in order for such a condition to be fulfilled.

In the previous sections, it was explained that the ChaCha20 function receives a hash value generated with SHA256 as input and returns an array of random numbers. With the slightest change in the input, the output can change completely and does not necessarily include the previous values. For this reason, we use the hypergeometric distribution model to simulate this scenario.

For better understanding, suppose we have a bag containing 1000 white and black balls representing all nodes of the network. $\frac{1}{3}$ of the balls are black (malicious nodes) and $\frac{2}{3}$ are white (honest nodes). In each experiment, 128 balls are randomly selected. If not all balls are black, the balls are returned to the bag and the experiment is repeated.

At first, we use the probability formula in the hypergeometric distribution to calculate the probability that all 128 balls are black.

$$P(X=k)=\frac{\binom{K}{k}\times\binom{N-K}{n-k}}{\binom{N}{n}}$$

Where:

- $N = 1000$ (total number of balls or total network nodes)
- $K = 333$ (number of black balls or malicious nodes)
- $n = 128$ (number of balls drawn)
- $k = 128$ (number of black balls desired)

In the next step we substitute the numbers.

$$P(X=128)=\frac{\binom{333}{128}\times\binom{1000-333}{128-128}}{\binom{1000}{128}}=\frac{\binom{333}{128}\times\binom{667}{0}}{\binom{1000}{128}}$$

Since $1=\binom{667}{0}$, we can simplify this probability to:

$$P(X=128)=\frac{\binom{333}{128}}{\binom{1000}{128}}$$

In the next step we must do the calculations, but because the resulting numbers are very large we use factorials.

$$\binom{333}{128}=\frac{333!}{128!\times(333-128)!}=\frac{333!}{128!\times205!}$$

$$\binom{1000}{128}=\frac{1000!}{128!\times(1000-128)!}=\frac{1000!}{128!\times872!}$$

Therefore:

$$P(X=128)=\frac{\frac{333!}{128!\times205!}}{\frac{1000!}{128!\times872!}}=\frac{333!\times872!}{1000!\times205!}$$

Calculating this value precisely in numeric form is difficult, because very large factorials are involved. For this reason, we use Stirling's approximation to estimate the logarithm of factorials.
Stirling's approximation formula for factorials:

$$\ln(n!)\approx n\ln(n)-n+\frac{1}{2}\ln(2\pi n)$$

We substitute the factorials into the formula.

$$\ln(333!)\approx 333\ln(333)-333+\frac{1}{2}\ln(2\pi\times333)\approx 1604.8865$$

$$\ln(872!)\approx 872\ln(872)-872+\frac{1}{2}\ln(2\pi\times872)\approx 5036.6165$$

$$\ln(1000!)\approx 1000\ln(1000)-1000+\frac{1}{2}\ln(2\pi\times1000)\approx 5911.3725$$

$$\ln(205!)\approx 205\ln(205)-205+\frac{1}{2}\ln(2\pi\times205)\approx 889.796$$

In the next step, after obtaining the approximate value of each, we calculate log p.

$$\ln(P)=\ln(333!)+\ln(872!)-\ln(1000!)-\ln(205!)$$

$$\ln(P)\approx 1604.8865+5036.6165-5911.3725-889.796\approx -159.6655$$

Therefore it can be said:

$$p \approx e^{-159.6655}$$

In the final step we simplify the probability of success in each attempt.

$$\ln(p) \approx -159.6655 \approx -159.6655 . 2.3026 . \log_{10}(e) \approx -69.6 . \log_{10}(e)$$

$$p \approx 10^{-69.3}$$

Up to this step we have calculated the probability of success in each attempt. In the next step we examine how many attempts are required on average for the first success.
Formula:

$$E[\text{Number of attempts}] = \frac{1}{p}$$

Finally, we substitute $p \approx 10^{-69.3}$ to calculate the average attempts required for the first success.

$$E[\text{Number of attempts}] \approx \frac{1}{10^{-69.3}} = 10^{69.3}$$

The results show that even with control of $\frac{1}{3}$ of all network nodes, the attacker must, on average, create and check 10^{69} different transactions in order to reach a transaction in which all members of the Validation Council are under their control. This is while the attacker, in the best case, has only 16 seconds to perform this process.
From a computational perspective, even with the most powerful supercomputers in the world, checking such a volume of transactions in this time frame is practically impossible.

it should be noted that the number of transactions required to be checked does not depend on the total number of nodes in the network, but rather on the percentage of nodes controlled by the attacker. Our prediction shows that even if the attacker controls $\frac{2}{3}$ of the network nodes, still, to construct a fraudulent transaction, they must check about 10^{24} different transactions. This amount too is computationally and temporally unattainable in practice.

References

[1] Dr. Leemon Baird, Mance Harmon, and Paul Madsen. Hedera: A Public Hashgraph Network & Governing Council. https://files.hedera.com/hh_whitepaper_v2.2-20230918.pdf
[2] Dan Boneh, Craig Gentry, Ben Lynn, Hovav Shacham. Aggregate and Verifiably Encrypted Signatures from Bilinear Maps. https://crypto.stanford.edu/~dabo/pubs/papers/aggreg.pdf
[3] Dan Boneh, Manu Drijvers, Gregory Neven. BLS Multi-Signatures With Public-Key Aggregation. https://crypto.stanford.edu/~dabo/pubs/papers/BLSmultisig.html
[4] Vitalik Buterin. The Ethereum protocol. https://ethereum.org/en/whitepaper
[5] Manuel M.T. Chakravarty, James Chapman, Kenneth MacKenzie, Orestis Melkonian, Michael Peyton Jones, Philip Wadler. The Extended UTXO Model.
https://omelkonian.github.io/data/publications/eutxo.pdf
[6] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf
[7] Leslie Lamport, Robert Shostak, Marshall Pease. The Byzantine Generals Problem.
https://lamport.azurewebsites.net/pubs/byz.pdf
[8] Daniel J. Bernstein. ChaCha, a variant of Salsa20. https://cr.yp.to/chacha/chacha-20080120.pdf