X-Rollup: A Layer2 Solution for Ethereum

Part 1. Overview

With the explosive development of Defi, decentralized apps are being proposed in large numbers on Ethereum. Unfortunately, Ethereum, which is the most widely used smart contracts platform, is not prepared to scale to match the demands of end-user applications with mass adoption. Moreover, the user experience of DApps is very poor and in no way conducive for average users. There have been a few cases where one or the other particular application temporarily succeeded in achieving a significant user base, but it led to crippling the entire network during the high network load times. Specifically, Ethereum has the following drawbacks.

(1) Low Transaction Throughput

Ethereum has to maintain a certain amount of time lag between the production of adjacent blocks to ensure ample time for block propagation. Also, the block size needs to be small to ensure quick propagation of the block through the network. This entails that the number of transactions in a particular block needs to be fairly limited

(2) High Transaction Fees

With the rapid growth of the blockchain ecosystem, new crypto assets are increasingly being created, transferred, and sold, often involving multiple crypto tokens. Also, most decentralized apps have their own token and economy. Paying tokens for the services or doing any kind of transaction on blockchains requires on-chain transfers. Ethereum charges gas fees on each transaction, which greatly limits the on-chain operations in Defi.

(3) Poor User Experience

Due to the low transaction throughput and high transaction fees, interactions with DApps have high costs as well as unsatisfactory latency. As a result, users have a very bad experience in participating in Defi.

X-Rollup is a novel layer2 solution for Ethereum aiming to solve the scalability and usability issues, while not compromising on decentralization and leveraging the existing developer community and ecosystem. Compared to previous approaches, X-Rollup has advantages in security, compatibility, scalability, and latency.

Part 2. Technical details of X-Rollup

1. Architecture

The architecture of X-Rollup is shown in Figure 1. Layer1 refers to Ethereum with limited transaction throughput and high transaction fees. Layer 2 refers to Wanchain with high transaction throughput and low transaction fees. Developers can transplant their DApps from Layer1 to Layer2, while users can transfer their assets from Layer1

to Layer2 in order to gain a better user experience. Besides, they can redeem their assets back to Layer1 anytime without any latency. The security and robustness of the whole protocol are ensured by the Layer Connector, which connects Layer1 and Layer2 by building a channel for assets transferring and information verification.

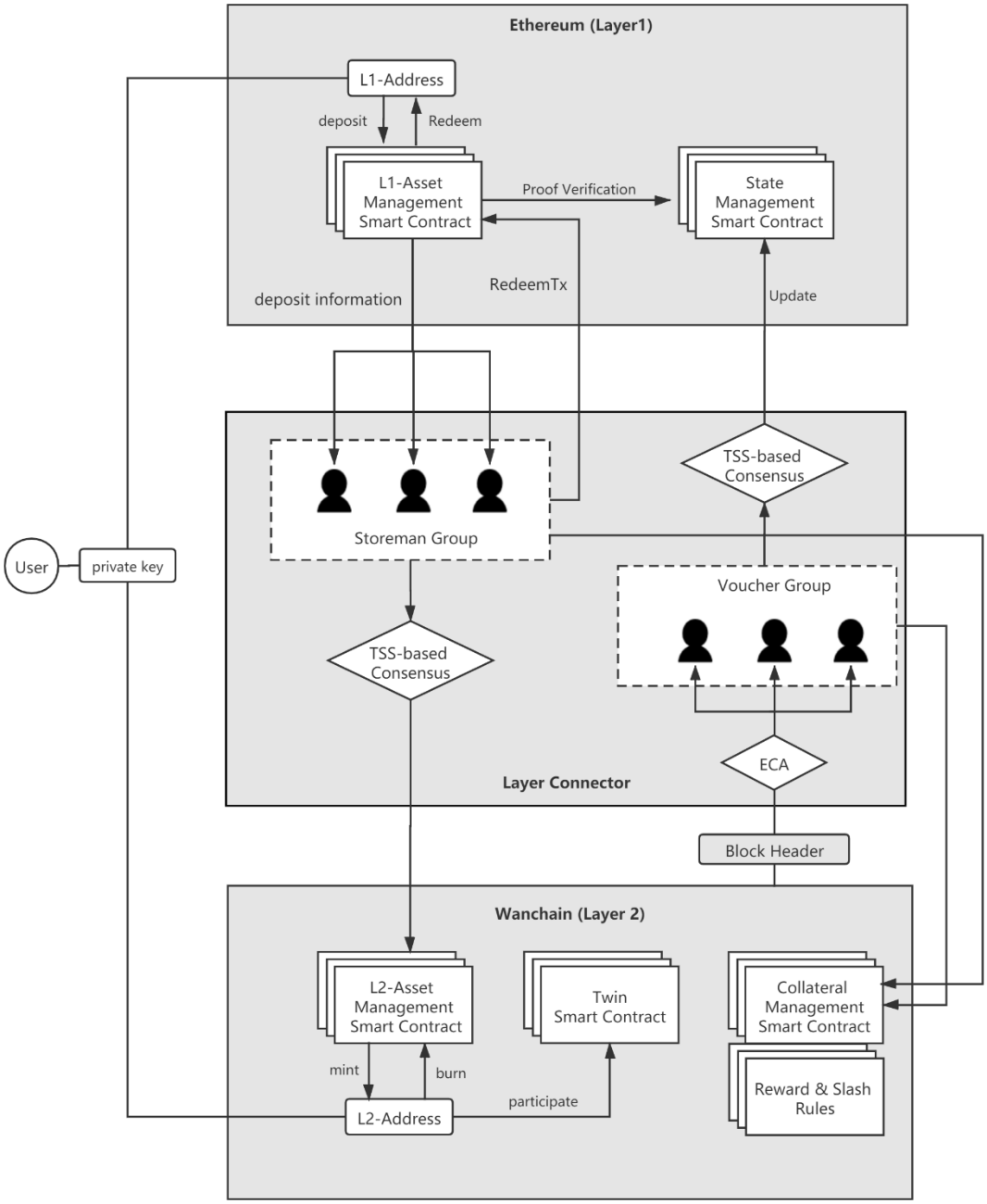

Figure 1. Architecture of X-Rollup

2. Roles

The Layer Connector is run by two roles, namely, Storeman and Voucher, which are described as follows.

Storeman

Storeman is in charge of transferring assets between Layer1 and Layer2. Specifically, the Storeman Group is listening to the events related to L1-Asset Management Contract in Layer1 and L2-Asset Management Contract in Layer2 24 hours a day. When users deposit assets into L1-Asset Management Contract, Storeman Group will mint the same amount of assets for users in L2-Asset Management Contract, which transfers assets from Layer1 to Layer2. Similarly, when users burn their assets in L2-Asset Management Contract, Storeman Group will send a redeem request to L1-Asset Management Contract and redeem users' assets to their accounts, which transfers assets from Layer2 back to Layer1.

Voucher

Voucher is in charge of committing block information of Layer2 to Layer1, which ensures any transaction in Layer2 can be verified in Layer1. Specifically, the Voucher Group maintains the latest blockchain data of Layer2 and commits the compressed block headers to Layer1 at regular intervals. With all the committed information, Layer2 is pegged to Layer1 automatically.

We remark that the members of Storeman Group and Voucher Group are elected from the community and required to lock a certain amount of collateral in the Collateral Management Contract. If they behave maliciously (for example, sending a fake redeem request or committing a wrong block header), their collateral will be slashed to compensate users' possible loss.

3. Smart Contracts

In our solution, there are five smart contracts to be deployed, of which two are in Layer1 and three are in Layer2.

L1-Asset Management Contract (L1-AMC)

L1-AMC is deployed in Layer1 and used to manage the original assets. It has two basic functions, i.e., deposit function and redeem function. The deposit function, which can be invoked by any user through deposit-transactions, receives original assets and locks them in L1-AMC. The redeem function, which can only be invoked by Storeman Group through redeem-transactions, verifies 2Stage-Proof, unlocks, and returns the original assets in L1-AMC to the user.

State Management Contract (SMC)

SMC is deployed in Layer1 and used to store the Merkel root of the compressed block headers of Layer2. It can only be accessed by Voucher Group. The Merkel root stored in SMC updates each time Voucher Group commits new block headers (Figure 2).

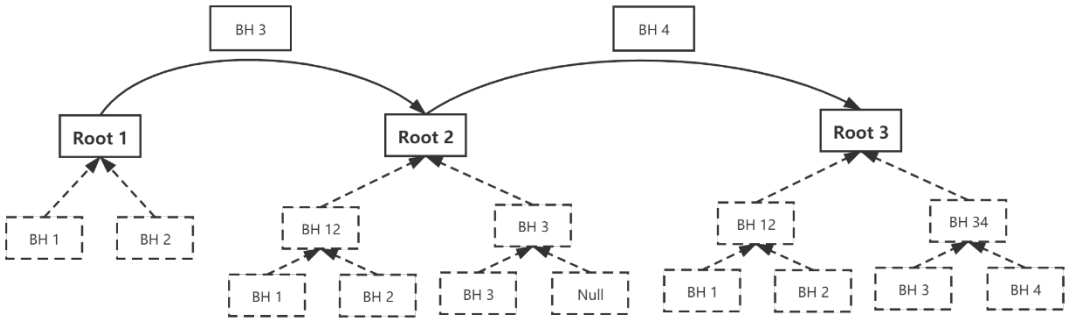

Figure 2. The evolution process of Merkel root stored in SMC

We remark that only the Merkel root is stored in SMC, which is necessary but also sufficient to verify any transaction in Layer2. This design has a much higher compression ratio compared to previous approaches and greatly saves the storage space of SMC.

L2-Asset Management Contract (L2-AMC)

L2-AMC is deployed in Layer2 and used to manage the wrapped assets. It has two basic functions, i.e., mint function and burn function. The mint function, which can only be invoked by Storeman Group through mint-transactions, mints wrapped assets for users in Layer2. The burn function, which can be invoked by any user through burn-transactions, burns wrapped assets.

Twin Contract (TC)

TC is deployed in Layer2 and has the same functionality and usage as the original smart contract in Layer1. With the same input (transaction), TC and the original smart contract share the same output. So users can interact with TC freely in Layer2 and return the result to Layer1 afterward.

Collateral Management Contract (CMC)

CMC is deployed in Layer2 and used to manage the collaterals of Storeman/Voucher Group. It realizes the reward and slash rules designed to ensure Storeman/Voucher's honest behavior. Specifically, on inputting the work records of Storeman/Voucher, CMC will evaluate its behavior automatically. Honest behavior will be rewarded, while malicious behavior may lead to the collateral being slashed. The whole process is fair as well as fully decentralized.

4. Core Algorithms

TSS-based Consensus (TBC)

TBC is used by Storeman/Voucher Group to reach an agreement on data in Layer1 and Layer2. Threshold Signature (TSS) is a cryptographic technique that is widely used in the blockchain area, such as crosschain solutions and digital wallets. Basically, a (n, t)-TSS separates the signing right among n parties. No less than t parties can work

together to generate a valid signature. The signing process is just like reaching a consensus, for a valid signature can only be generated with no less t parties' participation. TBC is the first consensus algorithm based on TSS. Compared with BFT, TBC has high flexibility as well as low computation complexity and storage consumption.

TBC proceeds as follows (Figure 3).

(1) DKG Stage: In this stage, all the parties perform Joint-Feldman secret sharing with a shared public key as output. Besides, each party also receives a secret key share. We remark that this procedure only needs to be done once.

(2) Consensus Stage: In this stage, all the parties try to reach a consensus on a piece of common data. A party approves the data by signing it with his secret key share and broadcasting the generated signature share to other parties. Each party also collects signature shares from the network. As long as t signature shares are collected, the party reconstructs the complete signature through Lagrange interpolation. If the signature is valid against the public key generated in the DKG stage, then all the parties have reached a consensus on the data.

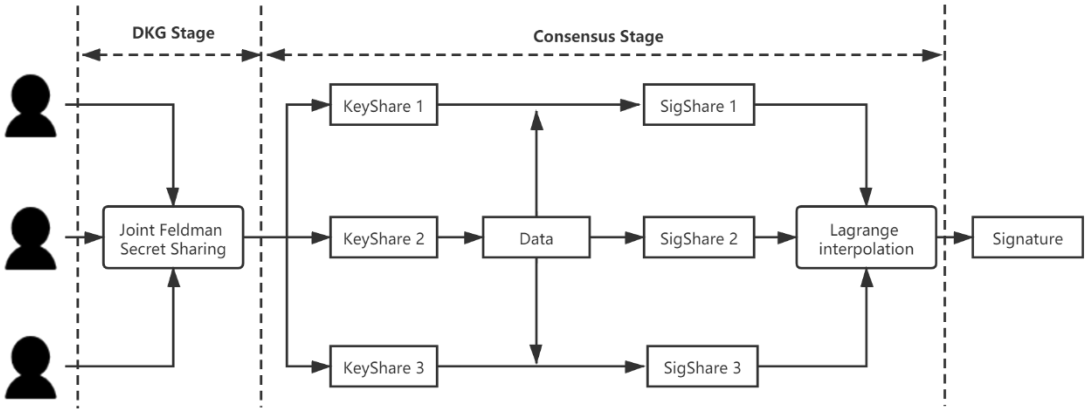

Figure 3. Working flow of TBC

Enhanced Compression Algorithm (ECA)

Compression tricks are key to the scalability of rollup solutions. ECA can compress the Layer2 blockchain into a 32 Bytes length value that can be used to verify any transaction or account status in Layer2. Compared with previous approaches, ECA achieves a better compression ratio by compressing the block headers rather than the transactions and constructing a Merkel tree of the compressed block headers. Specifically, ECA proceeds as follows:

Step 1：Reducing the number of block header's parameters.

There are 16 parameters in a Layer2 block header, most of which are irrelevant to data verification. So ECA block header only keeps three key parameters, i.e., ParentHash, Root, and TxHash. ParentHash can be used to identify block height, while Root and

TxHash can be used to verify account status and transactions, respectively. The Layer2 block header is compressed from over 516 Bytes to 96 Bytes in this step. The details are shown in Table 1.

Table 1. The data structure of ECA Block Header

Parameter	Layer2 Block Header	ECA Block Header
ParentHash	32 Bytes	32 Bytes
UncleHash	32 Bytes	——
Coinbase	20 Bytes	——
Root	32 Bytes	32 Bytes
TxHash	32 Bytes	32 Bytes
ReceiptHash	32 Bytes	——
Bloom	256 Bytes	——
Difficulty	8 Bytes	——
Number	8 Bytes	——
GasLimit	8 Bytes	——
GasUsed	8 Bytes	——
Time	8 Bytes	——
Extra	Not Fixed	——
Extra2	Not Fixed	——
MixDigest	32 Bytes	——
Nonce	8 Bytes	——

Step 2: Hashing ECA block header to 32 Bytes.

The final block header is the hash value of ParentHash, Root, and TxHash:

$$BH = SHA256(ParentHash, Root, TxHash)$$

This operation results in further compression of 64 Bytes. Even though there will be extra computing resource consumption in data verification, it is still worthy in storage-limited scenarios.

Step 3: Constructing a Merkel tree of ECA block headers

With the above two steps, the Layer2 block header has been compressed from over 516 Bytes to 32 Bytes. But it is still impractical to store each block header of Layer2 in Layer1, for the storage grows linearly as the Layer2 blockchain extends. To solve this problem, we construct a Merkel tree of ECA block headers, of which only the root is

stored in Layer1. It is updated each time new block headers are committed by the Voucher Group.

From the three steps, the Layer2 blockchain is compressed into a 32 Bytes length value (i.e. the Merkel root).

2-Stage Proof

2-Stage Proof is used to prove the correctness of a Layer2 transaction using the ECA compressed data stored in Layer1. A 2-Stage Proof has two components, namely, the First Stage Proof (FSP) and the Second Stage Proof (SSP). FSP is used to prove the fact that tx is stored in some Layer2 block with bh as its block header. SSP is used to prove the fact that bh is a valid block header of some block in Layer2. With a combination of FSP and SSP, we know that tx is a valid transaction in Layer2. We show this procedure with an example in Figure 4.

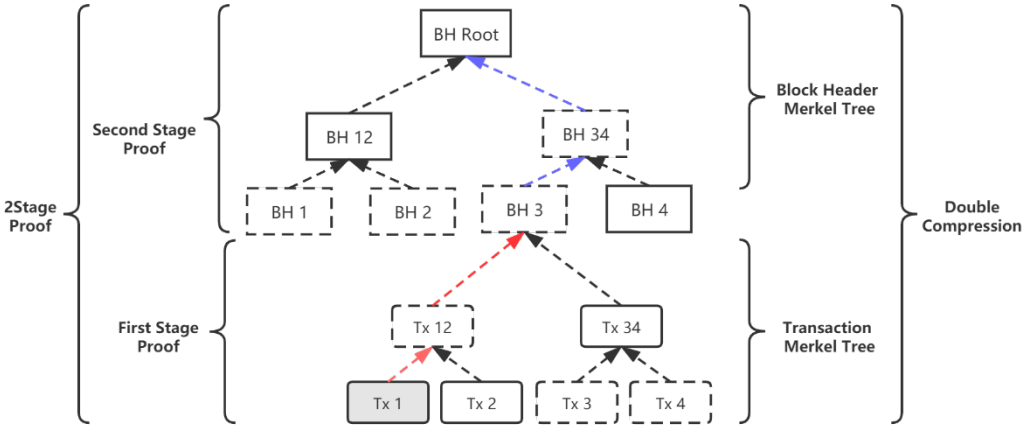

Figure 4. Verification path of 2-Stage Proof

To prove the correctness of $Tx1$, 2-Stage Proof is constructed as follows.

(1) Construct FSP:

$ECA\_BH3 = (ParentHash, Root, TxHash)$

$FSP = \{Tx1, Tx2, Tx34, ECA\_BH3\}$

(2) Construct SSP:

$BH3 = SHA256(ECA\_BH3)$

$SSP = \{BH3, BH4, BH12\}$

(3) Compose the two proofs:

$2StageProof = \{FSP, SSP\}$

The 2-Stage Proof is valid if the following equations hold:

(1) $TxHash = SHA256(SHA256(Tx1, Tx2), Tx34)$;

(2) $BH3 = SHA256(ECA\_BH3)$

(3) $BHRoot = SHA256(BH12, SHA256(BH3, BH4))$

5. Working Process

With all the notions and algorithms above, we show the working process of our solution as below.

Preparation Stage

(1) Deploy L1-Asset Management Contract and State Management Contract in Layer1;
(2) Deploy L2-Asset Management Contract, Twin Contract and Collateral Management Contract in Layer2;
(3) Complete the election of Storeman/Voucher Group;
(4) Members of Storeman/Voucher execute the DKG process of TSS-based consensus to generate the public key and corresponding secret key shares.

Working Stage

(1) The user sends a deposit-transaction from his original address (i.e. L1-Address in figure 1) to deposit ETH or any kind of ERC-20 tokens to L1-Asset Management Contract.
(2) Members of Storeman Group notice the deposit transaction and reach an agreement on the depositing information, including depositor's address, asset type, and asset amount through TSS-based consensus.
(3) Based on the agreed information, Storeman Group sends a mint-transaction to L2-Asset Management Contract to mint the same amount of wrapped-asset in Layer2 to L2-Address. We remark that L2-Address is the user's address in Layer2 and shares the same private key as the user's original address in Layer1.
(4) With L2-Address and the wrapped-assets in it, the user can send transactions to Twin Contract to participate in Defi in Layer2 at a low cost.

Withdraw Stage

(1) The user sends a burn-transaction from L2-Address to L2-Asset Management Contract to burn his wrapped-tokens.
(2) Members of Storeman Group notice the burn-transaction and reach an agreement on the information of the burned assets, including the sender's address, asset type, and asset amount through TSS-based consensus.
(3) Based on the agreed information, Storeman Group constructs and sends a redeem-transaction to L1-Asset Management Contract.
(4) L1-Asset Management Contract verifies the 2-Stage Proof in the redeem transaction according to the Merkel tree root of the block headers stored in State Management Contract and redeems the assets to L1-Address if the proof is valid.

We remark that there are two ways for Voucher Group to commit the compressed form of block headers to Layer1. The first way is committing at regular intervals once the Preparation Stage finishes. The second way is commiting related block headers once

there is a burn transaction in Layer2.

Part 3. Security Analysis

The security model of X-Rollup is based on two assumptions:

(1) SHA256 is collision-free;

(2) Honest majority assumption for Storeman/Voucher Group.

SHA256 is collision-free

It is a common consensus that SHA256 is a collision-free hash function, which is used in X-Rollup for data compression. Then we have the following Theorem:

Theorem: If the BH root stored in SMC is valid, then any PPT adversary cannot forge a 2-Stage Proof for an invalid transaction in Layer2.

Proof: We show that if there exists an adversary $\mathcal{A}$ can forge a 2-Stage Proof for an invalid transaction in Layer2, then he can also break the collision-free property of SHA256. As shown in Figure 5, $\mathcal{A}$ succeeds in forging a 2-Stage Proof for invalid transaction $Tx'4$:

$$FSP' = \{Tx'3, Tx'4, Tx'12, ECA\_BH4\}$$

$$SSP' = \{BH'3, BH'4, BH'12\}$$

$$2StageProof' = \{FSP', SSP'\}$$

According to the definition of 2-Stage Proof, we have the following equation:

$$BHRoot = SHA256(BH'12, SHA256(BH'3, BH'4))$$

$Tx1$ is a valid transaction of Layer2, which can be proved by a 2-Stage Proof. So we have the following equation:

$$BHRoot = SHA256(BH12, SHA256(BH3, BH4))$$

Combining the two equations, we have

$$SHA256(BH'12, SHA256(BH'3, BH'4)) = SHA256(BH12, SHA256(BH3, BH4))$$

Without loss of generality, we denote $(BH'12, SHA256(BH'3, BH'4))$ as x_1 and denote $(BH12, SHA256(BH3, BH4))$ as x_2. Then we have

$$x_1 \neq x_2$$

$$SHA256(x_1) = SHA256(x_2)$$

From the above we know, $\mathcal{A}$ constructs a collision instance for SHA256. As the probability of constructing a collision instance for SHA256 is negligible, $\mathcal{A}$ cannot forge a 2-Stage Proof for an invalid transaction in Layer2.

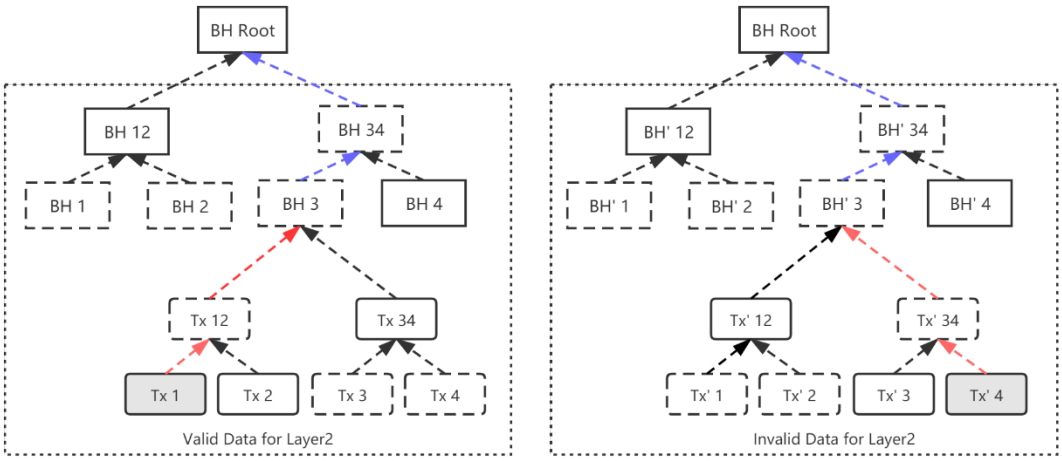

Figure 5. $\mathcal{A}$ is forging a 2-Stage Proof for invalid transaction $Tx'4$

Honest majority assumption for Storeman/Voucher Group

X-Rollup uses (t,n)-TSS in TSS-based consensus, where $t \geq n/2+1$. So under honest majority assumption, we have the following 2 facts:

(1) All mint and redeem transactions are signed by at least one honest Storeman;

(2) All committed block headers are signed by at least one honest Voucher.

Fact (1) ensures the security of users' assets, while Fact (2) ensures the validity of the Merkel root stored in Layer1.

Even though honest majority assumption has been widely used as a basic security assumption in the blockchain area, many blockchain protocols still face the risk of collusion attack. To address this problem, we provide two efficient solutions.

Solution 1：Anonymity

As shown in Figure 6, the election of Storeman/Voucher is divided into two stages. In the first stage, a number of nodes are randomly selected from the community; In the second stage, each of the nodes is assigned with the role of Storeman or Voucher anonymously through ring signature and stealth address. So a single node can neither know the other nodes' real word identity nor roles in X-Rollup. The anonymity greatly improves the difficulty of collusion attacks.

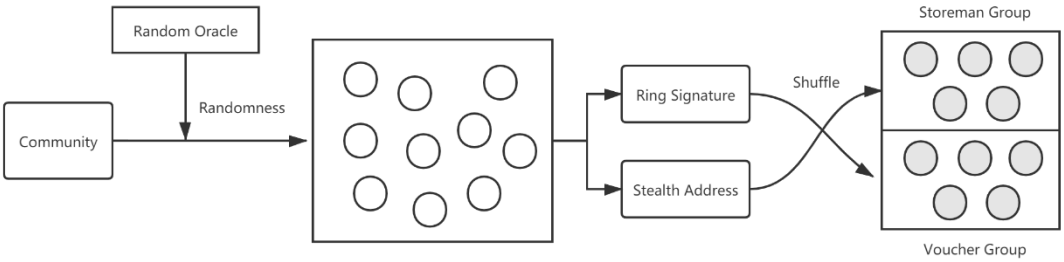

Figure 6. Election of Storeman/Voucher Group

Solution 2：Subgroup

In this solution, the Storeman/Voucher Group is further divided into several subgroups.

Each subgroup executes the DKG process separately, which makes the collusion attack even harder. Assuming there are 10 members in Storeman Group. If any 6 members can collude to control the whole group. But if we divide the Storeman Group into 2 subgroups with 5 members, it is required that 3 members in the first subgroup and 3 members in the second group collude to launch a collusion attack. Actually, the probability decreases exponentially as the number of subgroups increases. Theoretically, we can prevent collusion attacks completely.

Part 4. Comparison

In this section, we compare X-Rollup with Optimistic Rollup and ZK Rollup from aspects of data availability, transaction validity, compatibility, scalability, and latency.

Data availability

Both Optimistic Rollup and ZK Rollup publish compressed transactions in Layer1 to achieve data availability. In X-Rollup, the Voucher Group commits the compressed block headers to Layer1 at regular intervals, which also achieves data availability. Moreover, we employ a public blockchain (i.e. Wanchain) as Layer2, of which the liveness and persistence further improve data availability.

Transaction validity

In Optimistic Rollup, Layer2's transaction validity is ensured by fraud-proof. So At least 1-of-N honest participants who execute all Optimistic Rollup transactions and will submit fraud-proof in case an invalid state transition is published, which makes Optimistic Rollup vulnerable for DDoS attacks. In a ZK Rollup, every state transition is verified by the Rollup smart contract through zero-knowledge (PLONK) before it becomes effective. Unfortunately, the zero-knowledge requires a trusted setup and has high computation complexity. In X-Rollup, transaction validity is ensured by 2-Stage Proof, which is checked against the Merkel root of block header stored in Layer1. Due to the honest majority assumption, Voucher Group always maintains a correct Merkel root. Thus, no invalid transaction can pass the verification in Layer1.

Compatibility

Optimistic Rollup's Optimistic Virtual Machine (OVM) enables the implementation of arbitrary smart contract logic. Almost anything that is possible in Ethereum is also possible in the OVM, including the composability of smart contracts. It can be based on EVM, EWASM, or any other virtual machine. ZK Rollup has low compatibility for existed zero-knowledge schemes are designed for specialized operations such as token transfers or atomic swaps. X-Rollup has high compatibility due to the fact that Wanchain is a fork of Ethereum. So any Ethereum smart contract can transplant to Wanchain without code changes.

Scalability

Scalability is decided by the compression ratio. A higher compression ratio means higher Scalability. Both Optimistic Rollup and ZK Rollup choose transaction as the basic unit for compression. Optimistic Rollup's realistic throughput cap is about 450 TPS, while ZK Rollup's realistic throughput cap is about 680 TPS. As a comparison, X-Rollup employs ECA for data compression, of which the basic unit is block header rather than transaction. This characteristic makes our solution has a much high compression ratio. Theoretically, our solution's throughput cap can be more than 10,000 TPS.

Latency

Optimistic Rollup can only be safe with a 1-2 week fraud-proof challenge window. No transaction can be considered final until this time passes —neither an internal Rollup transaction nor an exit. So Optimistic Rollup has a latency of 1-2 weeks. Currently, ZKPs are quite computationally intense. At present, for a block of 1000 transactions, it has 20 minutes of proof generation time on ordinary server hardware. So ZK Rollup has a latency of 20 minutes. In X-Rollup, the latency comes from the regular intervals of Voucher Group committing the block header, which is no more than 5 minutes. So our solution has a latency of 5 minutes.

Part 5. Conclusion

X-Rollup is a novel layer2 solution for Ethereum with characteristics of high efficiency and low latency. It can be used to build a scalable and user-friendly ecosystem for third-party Decentralized applications to thrive on. The potential use cases include payments, atomic swaps, liquidity providers, decentralized exchange, lending, and any other applications. We believe X-Rollup is a competitive rollup scheme in the era of Defi.