A comparative study of Aztec, Miden, and Ola

In this article, we delve into the concept of “Hybrid Rollup,” examining how projects Aztec, Miden, and Ola approach this technology. We investigate their unique smart contract languages, explore state tree designs, and consider the trade-offs in privacy designs. Our objective is to provide a comprehensive overview of Hybrid Rollup technologies, helping you understand their key components and envision their future trajectory.

What is Hybrid Rollup?

We are delighted to see that our recent initiatives have been garnering an increasing amount of attention in the market. “Hybrid Rollup” is the most accurate summary of what we at Ola have been working on:

1. Rollup:

a. It operates at Layer 2, but it also has the flexibility to function at Layer 3, depending on the platform utilized for the verification contract deployment.

b. It’s a scalability solution.c. It has programmability - “Rollup” doesn’t specifically indicate this feature; “Programmable Rollup” is more accurate.

c. It has programmability - “Rollup” doesn’t specifically indicate this feature; “Programmable Rollup” is more accurate.

1. Hybrid:

a. It supports public, private, and hybrid contract types.

b. Developers can freely choose the contract type based on their needs.

c. Users can freely choose the transaction type in hybrid contracts.

The diagram below simply illustrates the positioning and functionalities of Hybrid Rollup. Currently, it’s known that three projects, namely Aztec, Miden, and Ola, are dedicated to building in this direction. Although the technical details of each vary, they share a common vision: to bring more possibilities, higher security performance, more real-world scenarios and more everyday users to the blockchain industry.

[

流程圖

1074×834 70 KB

](https://ethresear.ch/uploads/default/original/2X/2/2601daa3fe695e5d5a9dfba760e4cc36ff3c03b7.jpeg)

Programmable Rollup

As a (Programmable) Rollup, a smart contract (SC) language is needed to support developers in building DApps. However, since Privacy isn’t EVM (Ethereum Virtual Machine) compatible, Solidity cannot be used directly to develop Private DApps. Therefore, a custom SC language is required, one that can support both the writing of public contracts and private contracts. Of course, this also necessitates the adaptation of the VM module, as it needs to maintain different types of state trees.

Currently, the SC languages for the Aztec, Miden, and Ola projects are as shown in the following table:

Projects

Type

Language name

Language type

Turing-complete

Status

Aztec

ZK-ZKDSL

NOIR

DSL

No

In Development

Miden

ZK-ZKVM

Miden IR

GPL

Yes

In Development

Ola

ZK-ZKVM

Ola lang

GPL

Yes

In Development

The diagram below shows the different processing logic of Aztec and Ola (Miden is similar to Ola):

[

流程圖 (1)

1338×1594 163 KB

](https://ethresear.ch/uploads/default/original/2X/a/a2cc0d0a2f6099341eab7aace692a01fb47972d2.jpeg)

The most significant difference lies in the fact that in Aztec (and also in Aleo), each function of a contract is viewed as a fixed computation. At the compilation stage, specific circuits and pk/vk (public keys/verification keys) are generated for these computations, and the vk is used to identify the function being called. This leads to the following implications:

1. Function calls require special handling. The correct passing and returning of parameters are achieved in Aleo by introducing read-only registers, with the number of registers not being fixed.

2. The logic of the function must be deterministic computation; otherwise, it cannot be represented as a deterministic circuit and pk/vk cannot be generated.

3. Dynamic types are not supported, such as variables for the number of loops and dynamic array types. This is because these are unknown at the compilation stage and are only known during execution, thus it is impossible to compile the function into a circuit at the compiler stage.

In the design of Ola, we are not treating each function as a specific computation, for which we generate and store a unique pk/vk (public key/verification key) within the contract. Instead, we have designed an ISA-based VM, OlaVM. It’s universal, Turing-complete, and zk (zero-knowledge) friendly, capable of supporting computations of arbitrary logic. Simultaneously, we’ve defined a universal constraint system for the instruction set to regulate the correct execution of programs. Based on these designs, Ola possesses the following capabilities:

1. Turing Completeness: It supports computations of any logic, including dynamic types.

2. ZK Friendliness: A simplified constraint system is employed to regulate the correct execution of any program.

3. Savings on Computation and Storage: There is no need for additional computation and storage of pk/vk for each function.

Hybrid Global State

Tree Design

Within the Hybrid Rollup, two types of states will be maintained separately. One is the public state tree, corresponding to the account type, and the other is the private state tree, corresponding to the UTXO (Unspent Transaction Output) type, which we will refer to as the Note type going forward. The table below indicates the types of trees adopted by different projects:

Projects	Public state tree	Private note tree	Private nullifier tree
Aztec	SMT	Append-only Merkle tree	Indexed Merkle tree
Miden	SMT	Append-only Merkle tree	SMT
Ola - scheme-1	SMT	Append-only Merkle tree	Indexed Merkle tree
Ola -scheme-2	SMT	SMT	-

This article will not delve into the detailed design principles of different tree states. The design principles of different trees have already been provided in the table above. We express our respect for the work of the Aztec and Miden teams. Below, we will introduce the characteristics of different state trees:

1. Public state tree

The leaf nodes of the public state tree are the plaintext information of the accounts. It needs to support four functions: addition, deletion, modification, and search. Sparse Merkle Tree (SMT) is the best choice for this purpose, as in SMT:

a. The hash of an empty node is fixed, which can be cached, and the nodes don't need to store the full data of the tree.

b. When calculating the root, if an empty value is encountered, it can be directly utilized, leading to a reduction in the number of hash operations.

c. Even though the tree size is fixed, the upper limit is infinitely large, with a total of 2^256 leaf nodes.

d. It supports non-membership proof.

1. Private note tree

The leaf nodes of the private state tree are the commitment information of the note, and do not contain plaintext information. Each privacy transaction will consume old notes and generate new ones. If the note tree is updatable, nodes or listeners can infer the note information involved in the current transaction based on the leaf status of the state tree, such as which note commitments have been spent, which new note commitments have been generated, but the plaintext information of the note will still not be revealed. In other words, the user information and transaction information of this privacy transaction will still not be exposed.

Therefore, in order to implement the untraceability of privacy transactions, the private state tree can only be append-only, and leaf nodes can not be deleted or updated.

1. Private nullifier tree

The private nullifier tree is a tree maintained to assist the untraceability of privacy transactions

. Its main purposes are to:

a. Prevent the same note from being double-spent;

b. Disconnect the link between the inputs of a privacy transaction and the outputs of previous transactions, achieving untraceability;

c. As shown in the diagram below:

[

流程圖 (2)

1276×1195 103 KB

](https://ethresear.ch/uploads/default/original/2X/5/5eafbe324e7185addaa6d9f8021e591f98b605be.jpeg)

Therefore, for the private nullifier tree, it is necessary to support leaf addition and non-membership proof. Hence, the Sparse Merkle Tree (SMT) is a relatively good choice. From an efficiency perspective, Aztec introduced the concept of an index merkle tree, which is an enhanced version of the SMT.

Privacy Design

Ola has been contemplating privacy for a long time. What kind of needs and desires do users have regarding privacy?

a. Sometimes, users do not want their on-chain transactions to be monitored.

b. Sometimes, users do not want their on-chain data to be used illegally or without compensation.

c. The cost of private actions and public actions should be similar.

The table below shows the privacy features that the three projects - Aztec, Miden, and Ola - can provide:

Projects

Traceability

Data privacy

User Privacy

Compliance

Aztec

no

yes

yes

nomal

Miden

no

yes

yes

nomal

Ola - scheme -1

no

yes

yes

nomal

Ola - scheme -2

yes

yes

yes

better

All the solutions in the table above can achieve the two points a and b mentioned earlier. If we adopt the design of three different state trees in the Tree design section, this would provide the highest level of security, namely, untraceable privacy transactions

. However, this comes at the cost of a more complex implementation, transaction structure, circuit design, and higher transaction costs.

Therefore, when Ola designed privacy, it faced a trade-off between the untraceability of privacy transactions and the cost of privacy transactions:

a. Given that user transactions on the blockchain are already private, such as the implementation of Data privacy and User privacy, is there still a need to implement the untraceability feature, even if it leads to more complex designs and higher costs?

b. If privacy transactions can be traced, will it completely destroy users' privacy?

We believe that traceability might be a desirable privacy protection scheme:

a. It can still protect the privacy of user transactions, and achieve user data ownership.

b. It has a simpler privacy architecture design.

c. It requires less computational resources for proof, hence the transaction cost would be lower.

d. It is regulator-friendly, making it easy to trace the flow of privacy objects (without revealing other information).

Based on the considerations above, Ola will support two levels of privacy solutions, leaving the choice of privacy needs for developers. Over the next few months, Ola will gradually implement and verify the aforementioned solutions.

Summary

Ola is a ZK-ZKVM platform, whose primary purpose is to build a Layer2 infrastructure that combines optional privacy with high performance.

It can easily extend corresponding functionalities to platforms that do not have privacy and high-performance features while inheriting their network security, such as Ethereum, BSC, Aptos, zkSync

, etc. All that is needed is to deploy the corresponding verification contracts and bridge contracts on their chains.

We will continue to monitor and report on the latest developments in this field. Should you have any queries or suggestions, we would be delighted to hear from you. Please feel free to reach out to us at contact@sin7y.org.

Stay Tuned

Website | Whitepaper |GitHub | Twitter | Discord | LinkedIn | YouTube | HackMD | Medium | HackerNoon