Umi Network: The First Unified Multi-VM Infrastructure Blockchain for Secure and Scalable Move

Umi Network

https://uminetwork.com

Abstract

The Move programming language is widely regarded for its secure, resource-oriented design — offering a powerful foundation for the next generation of smart contracts. Yet, Move-based blockchains have struggled to scale beyond niche adoption, often siloed from the broader liquidity, tooling, and user base of mainstream ecosystems.

Umi Network introduces a new class of blockchain infrastructure: a Unified Multi-VM Infrastructure (UMI) that allows multiple virtual machines — starting with Move and EVM — to run natively, side-by-side, in a single execution environment. This unprecedented architecture enables true cross-VM composability, letting developers build applications that seamlessly integrate logic and assets across VM boundaries.

Umi is not simply a Layer-2, a sidechain, or a fork. It is a network extension architecture — a new execution layer designed to unify the best of multiple blockchain ecosystems while remaining sovereign, secure, and highly performant. Built with the modular OP Stack, Umi leverages Ethereum's mature infrastructure, liquidity, and developer tooling — while providing a modern programming and user experience with Move.

For the first time, both Move-based and EVM-based applications can deploy natively within a single, composable runtime â without sacrificing performance, security, or developer experience. Umi expands what's possible on Ethereum — while simultaneously redefining what's possible in blockchain execution itself.

1 Introduction

The inception of Bitcoin in 2008 [**?**] ushered in a new era of decentralized and trustless computation facilitated by blockchain technology. While Bitcoin's core purpose was enabling peer-to-peer digital cash, it laid the foundational primitives for a more generalized blockchain framework. Ethereum [**?**], introduced in 2015, built upon these primitives by incorporating a Turing-complete virtual machine, paving the way for executable smart contracts and decentralized applications (dapps). The Ethereum Virtual Machine (EVM) is a stack-based VM with a Turing complete instruction set that enables the deployment and execution of user-defined bytecode programs on the Ethereum blockchain. This revolutionary model of distributed computation rapidly catalyzed innovation, giving rise to decentralized finance (DeFi), non-fungible tokens (NFTs), decentralized autonomous organizations (DAOs), among other blockchain-native applications and use cases.

Solidity vs Move. While Solidity became the de facto language for Ethereum smart contract development, it suffers from inherent security risks stemming from semantic ambiguities, lack of formalism, and code complexity [**?**]. These vulnerabilities have led to numerous high-profile exploits resulting in substantial financial losses, most notably the infamous 2016 DAO attack [**?**] which drained around $60 million in ether at the time, as well as the more recent $190 million Nomad bridge hack [**?**] in 2022. The Move language [**?**], originally designed at Meta (Facebook) for the Diem blockchain, emerged as a safer and more robust alternative to Solidity. Move employs a bytecode interpretation execution model and follows a strict design philosophy focused on simplicity, auditability, and prevention of unintended behaviors. With influences from linear logic and secure coding principles, Move mitigates many of Solidity's pitfalls by preventing re-entrancy, data races, and other common vulnerabilities through its design that allows only a single execution context to access resources at any given time. Despite its security advantages, the Move ecosystem has faced challenges with limited liquidity, hindering widespread adoption compared to the more established Ethereum/Solidity landscape.

Umi Network Solution. To unlock Move's full potential while inheriting Ethereum's unparalleled liquidity and network effects, we introduce Umi - a novel optimistic rollup architecture built on the Optimism OP Stack [**?**

]. Umi replaces the traditional EVM execution layer with a Move execution layer, enabling seamless execution of Move smart contracts. The OP Stack's modular architecture allows us to leverage its sequencer, batcher, and proposer modules, while integrating our custom Move execution layer. As part of the execution process, we store contract data in a manner consistent with traditional Move blockchains, ensuring compatibility and ease of use for Move developers. To further improve transaction finality, we will be adding ZK proof computation on top of the optimistic rollup, leveraging the benefits of both technologies. By combining the scalability and usability of optimistic rollup with the security and finality of ZK proofs, Umi achieves a unique balance of performance and security, enabling fast and secure execution of Move smart contracts.

Central to Umi is a modular rollup design that prioritizes scalability, parallelization, and flexibility. We employ parallel execution techniques to maximize throughput, with the ability to run multiple Move virtual execution sessions concurrently. Our architecture remains flexible, enabling integration of the most efficient and available ZK tooling to provide fast finality. This agile approach ensures Umi can adapt to emerging innovations, consistently delivering a high-throughput layer 2 scaling solution as the ecosystem evolves.

Outline. Rest of the paper is organized as follows. In Section **??** we give an overview of the Umi's modular rollup solution; in **??** we describe in detail how the ZK proof computation is designed; in Section **??** we discuss EVM compatibility and the benefits of seamless interaction with the Ethereum ecosystem; then lastly in Section **??** we describe the SDK for developer experience and detail the gas computation on the native ETH token.

2 Umi Rollup

The Umi rollup is composed of several modular components, each serving a specialized role in the optimistic rollup architecture as shown in Figure **??**.

- **Sequencer**: The sequencer is responsible for ordering and batching incoming transactions, ensuring a consistent and tamper-proof transaction log.
- **Move Execution Layer**: The Move execution layer is a customized component that enables seamless execution of Move smart contracts.

This layer interacts with the sequencer to execute transactions and store the resulting changes in the database.

- **Batcher**: The batcher aggregates transactions into batches, which are then processed by the Move execution layer.

- **Proposer**: The proposer is responsible for proposing new states to the Ethereum mainnet, ensuring that the Umi rollup remains in sync with the Ethereum blockchain.

The Optimism OP Stack, which underlies the Umi rollup, uses a combination of on-chain and off-chain components to achieve scalability and security. On-chain, the OP Stack uses Bedrock contracts to store the ordered sequence of transactions, while off-chain, the sequencer and batcher work together to execute transactions and produce new state roots. The Bedrock contracts serve as the source of truth for the rollup's state, allowing the OP Stack to ensure that the Umi rollup remains in sync with the Ethereum mainnet.

When a new batch of transactions is processed, the OP Stack generates a new state root, which is then proposed to the Ethereum mainnet via the proposer. If the proposal is accepted, the new state root is written to the Bedrock contracts, effectively updating the rollup's state. This process allows the Umi rollup to achieve high throughput and low latency, while still maintaining the security guarantees of the Ethereum mainnet.

The Umi rollup architecture is designed to be modular, allowing for continuous improvement and optimization of individual components. By leveraging the Optimism OP Stack's modular architecture, we can seamlessly swap out and upgrade components without disrupting the overall system. This modularity enables us to support the Move VM by simply replacing the execution layer, rather than building an entirely new rollup.

To ensure the integrity of the rollup, we will utilize validity proofs using ZK Move implementation. This will enable fast and secure execution of Move smart contracts, while maintaining the security guarantees of the Ethereum mainnet. Additionally, a dedicated Bridge Contract enables seamless liquidity flows between the Umi rollup and the Ethereum mainnet. This bridge facilitates transfers of native ETH as well as ERC-20 standard tokens, allowing users to deposit assets from Ethereum into the Umi Network and withdraw them from Umi back to the Ethereum mainnet.

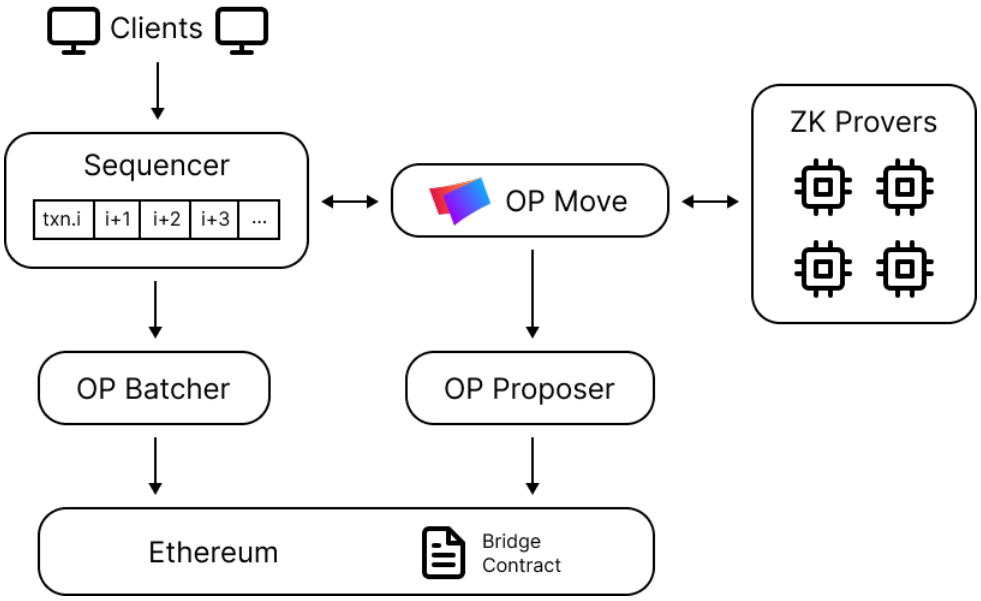


Figure 1: High-level architecture of the Umi optimistic rollup, depicting the interaction between user applications, the sequencer, Move execution layer, batcher, proposer, and the Ethereum L1 verifier contract, with seamless liquidity flows enabled by the bridge contract.

3 Zero Knowledge Proofs

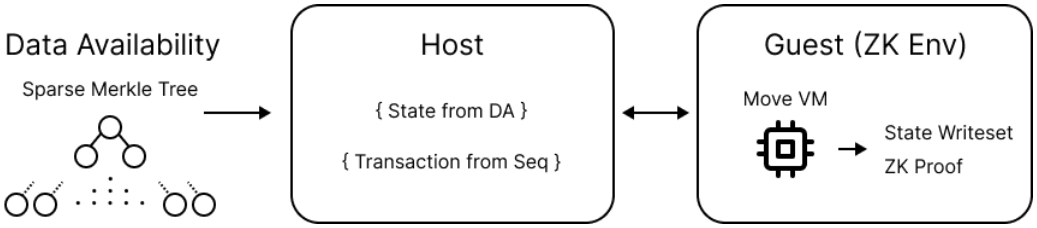
To ensure the integrity of the Umi rollup, we utilize Zero Knowledge (ZK) proofs to validate the correctness of off-chain computations for each individual transaction. The ZK proof is generated using the ZK Move implementation, which provides a secure and efficient way to prove the validity of Move smart contract executions. The ZK validity proof is then sent to Ethereum along with the proposed state update, allowing for fast and secure finalization of transactions. By leveraging ZK proofs, we can achieve fast and secure execution of Move smart contracts, while maintaining the security guarantees of the Ethereum mainnet. This approach enables transactions to be finalized in a matter of minutes, rather than waiting for a week-long dispute period as with traditional optimistic rollup architectures.

The use of ZK proofs in conjunction with optimistic rollup enables us to achieve a unique balance of scalability, security, and usability. By generating ZK proofs for each individual transaction, we can ensure that the Umi

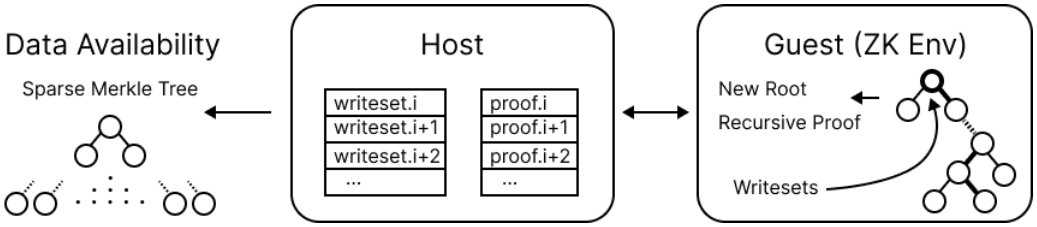
rollup remains secure and trustworthy, while also enabling fast and efficient execution of Move smart contracts.

3.1 ZK VM

One approach to implementing ZK proofs for Move smart contracts is to utilize a generalized ZK VM like Risc 0 [**?**]. This approach involves compiling the Move VM to the RISC-V instruction set architecture (ISA) and executing it within the zkVM. The zkVM generates cryptographic constraints that model the computational steps, which are then used to produce a succinct ZK-Proof. While this approach is viable, it may have performance limitations due to the overhead of the zkVM. However, it can still be useful in a hybrid model where ZK computation is only run on demand, allowing for faster finality for urgent transactions while maintaining the week-long finality for others.



(a) Step 1: Guest executions receive state and transaction data to run Move VM, producing state writeset and ZK-Proof outputs.



(b) Step 2: All the state writesets update Merkle tree and proofs are aggregated recursively.

Figure 2: Workflow of Risc Zero executing a Move smart contract to compute a new Merkle root representing the updated state.

The Zero Knowledge execution process within Risc Zero involves the interaction between a non-ZK Host environment and a ZK-friendly Guest en-

vironment, as depicted in Figure **??**. The Host retrieves all necessary state details from a Sparse Merkle Tree [**?**] representation and transmits them to the Guest. The goal of the ZK execution is to perform the state transition by taking as input the current state root together with a batch of transactions, and producing as output the new state root after the transactions in that batch have been executed. The ZK execution proceeds in two steps.

First, all transactions (deployment of new Move contracts, or executions on existing ones) in the batch are executed in parallel on separate Guests. Within each Guest, the Move smart contract is executed by running the Move virtual machine inside a ZK computation model. The output from each of these executions is a changeset containing the state changes to be applied across different account resources. Once the ZK computation finishes processing we will have a proof of the execution for the transaction.

This proof also includes applying all the changes to the state from the individual changes sets and computing the updated Merkle root. This updated root, along with the ZK proof that is was correctly produced, is the output of the ZK execution.

3.2 ZK Assembly

While the initial Umi architecture relies on Risc Zero's general zkVM, we plan to implement Move-optimized ZK execution in a future phase.

Move Bytecode Level ZK Execution. An alternative approach to implementing ZK proofs for Move smart contracts is to utilize a ZK Assembly language like Miden [**?**].

This approach involves mapping each Move bytecode to a corresponding Miden assembly instruction, allowing for ZK computation to be performed directly on the smart contract bytecodes. By executing the bytecodes at a lower level, this approach eliminates the overhead of running the entire Move VM, resulting in improved performance and efficiency. The Miden assembly language is well-suited for this purpose, as it is similar to a programming language assembly and can be easily mapped to the Move bytecode.

Achieving bytecode-level ZK execution requires a specialized compiler toolchain that can analyze and translate individual Move bytecode instructions into an optimized ZK assembly representation. Umi's compiler parses through each Move bytecode operation, methodically converting it into equivalent ZK assembly code tailored for the Miden virtual machine. This conver-

```
Bytecode::Add => Node::Instruction(Instruction::Add),
Bytecode::Sub => Node::Instruction(Instruction::Sub),
Bytecode::Mul => Node::Instruction(Instruction::Mul),
Bytecode::Div => Node::Instruction(Instruction::U32Div),
```

Figure 3: Illustration of a straightforward mapping from Move bytecode instructions to the equivalent Miden ZK assembly representations.

sion process involves a mix of direct bytecode mappings for straightforward instructions (as illustrated in Figure **??**) as well as complex mappings for those that require more intricate ZK constraint modeling (as the conversion steps shown in Figure **??**).

Once the entire Move bytecode has been transformed into the ZK assembly representation, the resulting code essentially becomes the ZK smart contract deployed within the Miden execution environment. When users initiate transactions, the corresponding ZK assembly instructions are loaded and executed within the ZK virtual machine's constrained CPU. This ZK execution model generates succinct proofs at the granular bytecode operation level, attesting to the correct computational steps. Additionally, Umi's compiler performs further optimizations on the final ZK assembly, enhancing execution efficiency by minimizing redundant constraints and leveraging batch processing where applicable.

4 EVM Compatibility

Umi provides seamless compatibility with the Ethereum ecosystem through a novel Move smart contract-based approach. The EVM Emulator contract accurately replicates the behavior of the EVM, enabling the deployment and execution of Solidity contracts on the Move VM. By executing EVM bytecode exactly as expected and calculating gas costs based on EVM specifications, the EVM Emulator ensures that Solidity contracts run correctly and produce the same results as they would on the Ethereum mainnet.

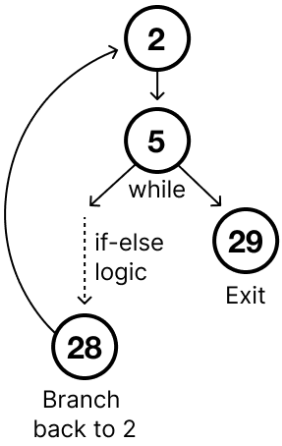
The EVM Emulator has been thoroughly tested and is able to run Ethereum tests as expected, demonstrating its compatibility and correctness. This means that developers can confidently deploy their existing Solidity contracts on the Umi network without modification, taking advantage of the scalability and security of the Umi platform.

```
fun collatz(n: u32): u32 {
  let count: u32 = 0;
  while (n != 1) {
    if (n % 2 == 0) {
      n = n / 2;
    } else {
      n = 3 * n + 1;
    };
    count = count + 1;
  };
  count
}
```

(a) Collatz conjecture sequence calculation in Move

```
vec![
  Bytecode::LdU32(0),
  Bytecode::StLoc(1),
  Bytecode::CopyLoc(0),   (2)
  Bytecode::LdU32(1),
  Bytecode::Neq,
  Bytecode::BrFalse(29), (5)
  ...
  Bytecode::Branch(2),   (28)
  Bytecode::MoveLoc(1), (29)
  Bytecode::Ret,
]
```

(b) Generated Move bytecodes with line numbers in parantheses



(c) Corresponding Control Flow Graph between lines

```
proc.collatz
  ...
  while.true
    // if-else logic
    ...
  end
  // Ln 29: Save in memory, exit
  mem_store.COLLATZ_INDEX
end
```

(d) Generated Miden assembly displaying only the branching sections

Figure 4: Move source code (a) is compiled to bytecode with branches (b), which constructs a Control Flow Graph (c) representing state transitions, guiding the heuristic generation of optimized Miden assembly (d).

By providing EVM compatibility, Umi aims to bridge the gap between the Ethereum and Move ecosystems, enabling developers to leverage the best of both worlds.

5 Umi SDK and Gas

Umi Network will provide an SDK in multiple programming languages to facilitate developers interacting with our system. In our system architecture, we prioritize seamless integration with widely adopted Ethereum wallets, facilitating smooth transaction signing and transmission to the Sequencer. To achieve this, we ensure compatibility with Ethereum RPC endpoints, enabling robust support for these wallets. It will also query the data availability layer to get details on accounts. All of this is done by connecting to the Umi Network's Ethereum-standard RPC endpoints.

An important aspect of the Umi SDK is accurately computing the gas costs associated with executing transactions on the rollup network. There are two primary fee components - the cost of data availability storage and the cost of settling state proofs on Ethereum Layer 1. Gas fees are denominated and paid in ETH tokens.

When initiating a transaction, developers can leverage the *estimateGas* API which provides an estimate of the gas contribution from their transaction to the overall batch size. The base fee is the minimum price per transaction, primarily accounting for storage costs on Layer 1 and the data availability layer. Notably, the Ethereum Dencun update [**?**] significantly reduces the costs associated with temporary storage, which is particularly relevant for layer-2 solutions like Umi. With this update, Ethereum charges substantially less for temporary storage, making it more economical for layer-2 solutions to store data on the Ethereum mainnet. As a result, the base fee for transactions on the Umi network is expected to be very low, making it even more attractive for developers and users to leverage the scalability and security of the Umi platform.

Separately, the execution gas is a dynamic fee calculated by the Umi compiler based on the complexity of the Move bytecode execution. This portion is paid to the Sequencer and Provers to compensate for the computational resources consumed in processing transactions and maintaining operational overheads. Through this multi-component gas model, Umi ensures an equitable distribution of fees across different layers of the rollup architecture.

6 Conclusion

The Umi Network represents a significant step forward in the evolution of blockchain technology, offering a scalable, secure, and EVM compatible platform for decentralized applications. By leveraging the Optimism OP Stack and Move virtual machine, Umi enables developers to build high-performance applications that inherit the security guarantees of the Ethereum mainnet.

The use of Zero-Knowledge proofs and a modular architecture ensures that the Umi network remains secure, scalable, and adaptable to the needs of its users. With its unique combination of scalability, security, and EVM compatibility, the Umi network is poised to unlock new possibilities for decentralized applications and bring the benefits of blockchain technology to a wider audience.