---

title: Plasma chains description: An introduction to plasma chains as a scaling solution currently utilized by the Ethereum community. lang: en incomplete: true sidebarDepth: 3

A Plasma chain is a separate blockchain anchored to Ethereum Mainnet but executing transactions off-chain with its own mechanism for block validation. Plasma chains are sometimes referred to as "child" chains, essentially smaller copies of the Ethereum Mainnet. Plasma chains use fraud proofs (like optimistic rollups) to arbitrate disputes.

Merkle trees enable the creation of an endless stack of these chains that can work to offload bandwidth from parent chains (including Ethereum Mainnet). However, while these chains derive some security from Ethereum (via fraud proofs), their security and efficiency are affected by several design limitations.

Prerequisites {#prerequisites}

You should have a good understanding of all the foundational topics and a high-level understanding of Ethereum scaling.

What is Plasma?

Plasma is a framework for improving scalability in public blockchains like Ethereum. As described in the original Plasma whitepaper, Plasma chains are built atop another blockchain (called a "root chain"). Each "child chain" extends from the root chain and is generally managed by a smart contract deployed on the parent chain.

The Plasma contract functions, among other things, as a bridge allowing users to move assets between Ethereum Mainnet and the plasma chain. Although this makes them similar to sidechains, plasma chains benefit—at least, to some extent—from Ethereum Mainnet's security. This is unlike sidechains that are solely responsible for their security.

How does Plasma work?

The basic components of the Plasma framework are:

Off-chain computation {#off-chain-computation}

Ethereum's current processing speed is limited to ~ 15-20 transactions per second, reducing the short-term possibility of scaling to handle more users. This problem exists mainly because Ethereum's consensus mechanism requires many peer-to-peer nodes to verify every update to the blockchain's state.

Although Ethereum's consensus mechanism is necessary for security, it may not apply to every use case. For example, Alice may not need her daily payments to Bob for a cup of coffee verified by the entire Ethereum network since some trust exists between both parties.

Plasma supposes that Ethereum Mainnet doesn't need to verify all transactions. Instead, we can process transactions off Mainnet, freeing nodes from having to validate every transaction.

Off-chain computation is necessary since Plasma chains can optimize for speed and cost. For example, a Plasma chain may—and most often does—use a single "operator" to manage the ordering and execution of transactions. With just one entity verifying transactions, processing times on a plasma chain are faster than Ethereum Mainnet.

State commitments {#state-commitments}

While Plasma executes transactions off-chain, they are settled on the main Ethereum execution layer—otherwise, Plasma chains cannot benefit from Ethereum's security guarantees. But finalizing off-chain transactions without knowing the state of the plasma chain would break the security model and allow the proliferation of invalid transactions. This is why the operator, the entity responsible for producing blocks on the plasma chain, is required to publish "state commitments" on Ethereum periodically.

A commitment scheme is a cryptographic technique for committing to a value or statement without revealing it to another party. Commitments are "binding" in the sense that you cannot change the value or statement once you've committed to it. State commitments in Plasma take the form of "Merkle roots" (derived from a Merkle tree) which the operator sends at intervals to the Plasma contract on the Ethereum chain.

Merkle roots are cryptographic primitives that enable compressing of large amounts of information. A Merkle root (also called a "block root" in this case) could represent all the transactions in a block. Merkle roots also make it easier to verify that a small piece of data is part of the larger dataset. For instance, a user can produce a Merkle proof to prove the inclusion of a transaction in a specific block.

Merkle roots are important for providing information about the off-chain's state to Ethereum. You can think of Merkle roots as "save points": the operator is saying, "This is the state of the Plasma chain at x point in time, and this is the Merkle root as proof." The operator is committing to the *current state* of the plasma chain with a Merkle root, which is why it is called a "state commitment".

Entries and exits {#entries-and-exits}

For Ethereum users to take advantage of Plasma, there needs to be a mechanism for moving funds between Mainnet and plasma chains. We cannot arbitrarily send ether to an address on the plasma chain, though—these chains are incompatible, so the transaction would either fail or lead to lost funds.

Plasma uses a master contract running on Ethereum to process user entries and exits. This master contract is also responsible for tracking state commitments (explained earlier) and punishing dishonest behavior via fraud proofs (more on this later).

Entering the plasma chain {#entering-the-plasma-chain}

To enter the plasma chain, Alice (the user) will have to deposit ETH or any ERC-20 token in the plasma contract. The plasma operator, who watches contract deposits, recreates an amount equal to Alice's initial deposit and releases it to her address on the plasma chain. Alice is required to attest to receiving the funds on the child chain and can then use these funds for transactions.

Exiting the plasma chain {#exiting-the-plasma-chain}

Exiting the plasma chain is more complex than entering it for several reasons. The biggest one is that, while Ethereum has information about the plasma chain's state, it cannot verify if the information is true or not. A malicious user could make an incorrect assertion ("I have 1000 ETH") and get away with providing fake proofs to back up the claim.

To prevent malicious withdrawals, a "challenge period" is introduced. During the challenge period (usually a week), anyone can challenge a withdrawal request using a fraud-proof. If the challenge succeeds, then the withdrawal request is denied.

However, it is usually the case that users are honest and make correct claims about the funds they own. In this scenario, Alice will initiate a withdrawal request on the root chain (Ethereum) by submitting a transaction to the plasma contract.

She must also provide a Merkle proof verifying that a transaction creating her funds on the Plasma chain was included in a block. This is necessary for iterations of Plasma, such as Plasma MVP, that use a Unspent Transaction Output (UTXO) model.

Others, like Plasma Cash, represent funds as non-fungible tokens instead of UTXOs. Withdrawing, in this case, requires proof of ownership of tokens on the Plasma chain. This is done by submitting the two latest transactions involving the token and providing a Merkle proof verifying the inclusion of those transactions in a block.

The user must also add a bond to the withdrawal request as a guarantee of honest behavior. If a challenger proves Alice's withdrawal request invalid, her bond is slashed, and some of it goes to the challenger as a reward.

If the challenge period elapses without anyone providing a fraud-proof, Alice's withdrawal request is considered valid, allowing her to retrieve deposits from the Plasma contract on Ethereum.

Dispute arbitration {#dispute-arbitration}

Like any blockchain, plasma chains need a mechanism for enforcing the integrity of transactions in case participants act maliciously (e.g. double-spending funds). To this end, plasma chains use fraud proofs to arbitrate disputes concerning the validity of state transitions and penalize bad behavior. Fraud proofs are used as a mechanism through which a Plasma child chain files a complaint to its parent chain or to the root chain.

A fraud-proof is simply a claim that a particular state transition is invalid. An example is if a user (Alice) tries to spend the same funds twice. Perhaps she spent the UTXO in a transaction with Bob and wants to spend the same UTXO (which is now Bob's) in another transaction.

To prevent the withdrawal, Bob will construct a fraud-proof by providing evidence of Alice spending the said UTXO in a previous

transaction and a Merkle proof of the transaction's inclusion in a block. The same process works in Plasma Cash—Bob would need to provide proof that Alice earlier transferred the tokens she's trying to withdraw.

If Bob's challenge succeeds, Alice's withdrawal request is canceled. However, this approach relies on Bob's ability to watch the chain for withdrawal requests. If Bob is offline, then Alice can process the malicious withdrawal once the challenge period elapses.

The mass exit problem in plasma {#the-mass-exit-problem-in-plasma}

The mass exit problem occurs when a large number of users try to withdraw from a plasma chain at the same time. Why this problem exists has to do with one of Plasma's biggest problems: **data unavailability**.

Data availability is the ability to verify that the information for a proposed block was actually published on the blockchain network. A block is "unavailable" if the producer publishes the block itself but withholds data used to create the block.

Blocks must be available if nodes are to be able to download the block and verify the validity of transactions. Blockchains ensure data availability by forcing block producers to post all transaction data on-chain.

Data availability also helps with securing off-chain scaling protocols that build on Ethereum's base layer. By forcing operators on these chains to publish transaction data on Ethereum, anyone can challenge invalid blocks by constructing fraud proofs referencing the correct state of the chain.

Plasma chains primarily store transaction data with the operator and**do not publish any data on Mainnet** (i.e., besides periodic state commitments). This means users must rely on the operator to provide block data if they need to create fraud proofs challenging invalid transactions. If this system works, then users can always use fraud proofs to secure funds.

The problem starts when the operator, not just any user, is the party acting maliciously. Because the operator is in sole control of the blockchain, they have more incentive to advance invalid state transitions on a larger scale, such as stealing funds belonging to users on the plasma chain.

In this case, using the classic fraud-proof system does not work. The operator could easily make an invalid transaction transferring Alice and Bob's funds to their wallet and hide the data necessary for creating the fraud-proof. This is possible because the operator isn't required to make data available to users or Mainnet.

Therefore, the most optimistic solution is to attempt a "mass exit" of users from the plasma chain. The mass exit slows down the malicious operator's plan to steal funds and provides some measure of protection for users. Withdrawal requests are ordered based on when each UTXO (or token) was created, preventing malicious operators from front-running honest users.

Nonetheless, we still need a way to verify the validity of withdrawal requests during a mass exit—to prevent opportunistic individuals from cashing in on the chaos processing invalid exits. The solution is simple: require users to post the last **valid state of the chain** to exit their money.

But this approach still has problems. For instance, if all users on a plasma chain need to exit (which is possible in the case of a malicious operator), then the entire valid state of the plasma chain must be dumped on Ethereum's base layer at once. With the arbitrary size of plasma chains (high throughput = more data) and constraints on Ethereum's processing speeds, this is not an ideal solution.

Although exit games sound nice in theory, real-life mass exits will likely trigger network-wide congestion on Ethereum itself. Besides harming Ethereum's functionality, a poorly coordinated mass exit means that users may be unable to withdraw funds before the operator drains every account on the plasma chain.

Pros and cons of plasma {#pros-and-cons-of-plasma}

| Pros | Cons | | ---------- | ---------- | | Offers high throughput and low cost per transaction. | Does not support general computation (cannot run smart contracts. Only basic token transfers, swaps, and a few other transaction types are supported via predicate logic. | | Good for transactions between arbitrary users (no overhead per user pair if both are established on the plasma chain) | Need to periodically watch the network (liveness requirement) or delegate this responsibility to someone else to ensure the security of your funds. | | Plasma chains can be adapted to specific use-cases that are unrelated to the main chain. Anyone, including businesses, can customize Plasma smart contracts to provide scalable infrastructure that works in different contexts. |

Relies on one or more operators to store data and serve it upon request. | | Reduces load on Ethereum Mainnet by moving computation and storage off-chain. | Withdrawals are delayed by several days to allow for challenges. For fungible assets, this can be mitigated by liquidity providers, but there is an associated capital cost. | | | If too many users try to exit simultaneously, Ethereum Mainnet could get congested. |

Plasma vs layer 2 scaling protocols {#plasma-vs-layer-2}

While Plasma was once considered a useful scaling solution for Ethereum, it has since been dropped in favor of layer 2 (L2) scaling protocols. L2 scaling solutions remedy several of Plasma's problems:

Efficiency {#efficiency}

Zero-Knowledge rollups generate cryptographic proofs of the validity of each batch of transactions processed off-chain. This prevents the users (and operators) from advancing invalid state transitions, eliminating the need for challenge periods and exit games. It also means users don't have to watch the chain periodically to secure their funds.

Support for smart contracts {#support-for-smart-contracts}

Another problem with the plasma framework was the inability to support the execution of Ethereum smart contracts. As a result, most implementations of Plasma were mostly built for simple payments or the exchange of ERC-20 tokens.

Conversely, optimistic rollups, are compatible with the Ethereum Virtual Machine and can run Ethereum-native smart contracts, making them a useful and *secure* solution for scaling decentralized applications. Similarly, plans are underway to create a zero-knowledge implementation of the EVM (zkEVM) that would allow ZK-rollups to process arbitrary logic and execute smart contracts.

Data unavailability {#data-unavailability}

As explained earlier, plasma suffers from a data availability problem. If a malicious operator advanced an invalid transition on the plasma chain, users would be unable to challenge it since the operator can withhold data needed to create the fraud-proof. Rollups solve this problem by forcing operators to post transaction data on Ethereum, allowing anyone to verify the chain's state and create fraud proofs if necessary.

Mass exit problem {#mass-exit-problem}

ZK-rollups and optimistic rollups both solve Plasma's mass exit problem in various ways. For example, a ZK-rollup relies on cryptographic mechanisms that ensure operators cannot steal user funds under any scenario.

Similarly, optimistic rollups impose a delay period on withdrawals during which anyone can initiate a challenge and prevent malicious withdrawal requests. While this is similar to Plasma, the difference is that verifiers have access to data needed to create fraud proofs. Thus, there's no need for rollup users to engage in a frenzied, "first-to-get-out" migration to Ethereum Mainnet.

How does Plasma differ from sidechains and sharding? {#plasma-sidechains-sharding}

Plasma, sidechains, and sharding are fairly similar because they all connect to Ethereum Mainnet in some way. However, the level and strength of these connections vary, which affects the security properties of each scaling solution.

Plasma vs sidechains {#plasma-vs-sidechains}

A sidechain is an independently operated blockchain connected to Ethereum Mainnet via a two-way bridge. Bridges allow users to exchange tokens between the two blockchains to transact on the sidechain, reducing congestion on Ethereum Mainnet and improving scalability. Sidechains use a separate consensus mechanism and are typically much smaller than Ethereum Mainnet. As a result, bridging assets to these chains involves increased risk; given the lack of security guarantees inherited from Ethereum Mainnet in the sidechain model, users risk the loss of funds in an attack on the sidechain.

Conversely, plasma chains derive their security from Mainnet. This makes them measurably more secure than sidechains. Both sidechains and plasma chains can have different consensus protocols, but the difference is that plasma chains publish Merkle roots for each block on Ethereum Mainnet. Block roots are small pieces of information we can use to verify information about transactions that happen on a plasma chain. If an attack happens on a plasma chain, users can safely withdraw their funds back to

Mainnet using the appropriate proofs.

Plasma vs sharding {#plasma-vs-sharding}

Both plasma chains and shard chains periodically publish cryptographic proofs to Ethereum Mainnet. However, both have different security properties.

Shard chains commit "collation headers" to Mainnet containing detailed information about each data shard. Nodes on Mainnet verify and enforce the validity of data shards, reducing the possibility of invalid shard transitions and protecting the network against malicious activity.

Plasma is different because Mainnet only receives minimal information about the state of child chains. This means Mainnet cannot effectively verify transactions conducted on child chains, making them less secure.

Note that sharding the Ethereum blockchain is no longer on the roadmap. It has been superseded by scaling via rollups and Danksharding.

Use Plasma {#use-plasma}

Multiple projects provide implementations of Plasma that you can integrate into your dapps:

- Polygon (previously Matic Network)

Further reading {#further-reading}

- Learn Plasma
- A quick reminder of what "shared security" means and why it's so important
- Sidechains vs Plasma vs Sharding
- Understanding Plasma, Part 1: The Basics
- The Life and Death of Plasma

Know of a community resource that helped you? Edit this page and add it!