Thanks to @blockdev for the feedback and discussion.

ArcPay is a payment validium with a fully trustless escape hatch.

We achieve this by giving users ownership proofs, which can be used to recover funds in a shutdown procedure in case the operator stops updating the state and the state can't be recovered. Users can forcibly acquire ownership proofs from the operator on-chain.

By having a slashable centralised operator, we also get strong instant finality guarantees, web2-style privacy guarantees, and cheap SNARK provers by using optimistic verification.

Concretely, this gives ArcPay extremely low fees regardless of L1 gas prices, practically unbounded throughput, and rollup-style security guarantees.

Trustless Escape Hatch

The problem with Validia

Like existing validia, users' funds are held on-chain by a smart contract, but the state is stored off-chain in a centralised server, and the smart contract only knows the Merkle root of the state. To update the state, the operator makes a SNARK proof for the state transition function which checks signatures, etc, and this SNARK is checked by the smart contract.

This is an extremely scalable architecture that can handle an arbitrary number of transactions per second with very low costs. Gas costs for on-chain SNARK verification are amortised across all transactions, and the only marginal cost per transaction is for proof generation. Moreover, that marginal cost is small and constant with modern recursive SNARKs such as Nova + Spartan.

The downside of existing validia is that user's funds will be locked forever if the operator refuses to update the on-chain state. Rollups solve this by making all the data necessary to rebuild the state available on-chain, but this imposes a gas cost on every transaction. Existing validia partially solve this by having a DAC (data availability committee) that replicates the state and can recover the state if m

-of-n

committee members act honestly. However, this is still a trust assumption on a set of external parties, and the costs of the DAC scale with its redundancy/security, i.e., n/m

.

By contrast, ArcPay decentralises the DAC by giving each user the necessary data to recover their funds. Specifically, users hold Merkle proofs that show they owned a particular set of coins at a given point in time. They can use these to claim their coins back, even if the state is not entirely recovered.

State Structure

In ArcPay, every coin is numbered, which is necessary to determine ownership during shutdown. The state of the validium is just a Merkle tree, where each leaf specifies the owner of a set of coins and has a unique id to prevent replay attacks.

[

State

2332×1856 93.1 KB

](https://ethresear.ch/uploads/default/original/2X/e/ea9cb998a759e4a593e8e0e5fb6f95460d70b3d2.png)

Note, the Merkle tree doesn't have to be ordered, either by coin or ID, but the coin ranges must be disjoint, which is ensured by the state transition proofs.

There are only 3 types of transaction: mint, send, and withdraw: mint moves coins from L1 to the validium; send moves coins within the validium; and withdraw sends coins back from the validium to L1. Send and withdraw require a signature, and that the signer actually owns the coins they're trying to send. Mint requires that the user actually sent coins on L1. These checks are done by the state transition proof.

Normal Operation

ArcPay has two distinct modes: normal operation, and shutdown. In normal operation, we assume that the operator doesn't want to lose their stake. In shutdown, we only assume L1 correctness and liveness, and that the user has gathered

appropriate ownership proofs during normal operation.

Censorship Resistance

During normal operation, censorship resistance is guaranteed by on-chain forcing.

Any user can call the force_include

function on the smart contract, which stores transactions in an accumulator until they are included in the state. The operator must address all forced transactions to update the state. If the state is not updated frequently, the operator is slashed and the validium is shut down.

Ideally, users never have to use force_include

, since posting a transaction on-chain costs gas, making the cost model essentially equivalent to a ZK rollup. Thankfully, the operator is incentivised to process transactions off-chain since they don't earn any fees for processing forced transactions.

In short, the operator generally will not censor, and even if they do, the users can circumvent it, assuming that the operator isn't slashed and L1 operates correctly.

Ownership Proofs

Users need to get ownership proofs from the operator to make sure they can recover their funds in case of a shutdown. An ownership proof is just a Merkle proof for a leaf in the state at a given point in time, i.e., user X owns coins [a,b]

.

If the operator is honest, a user can query an API and get their ownership proofs. But if that fails, the user can use the smart contract's force_ownership_proof

function, where the user asks "who owned coin X

at block b

?" This request is stored on-chain, along with the time it was made.

If the operator doesn't respond promptly, they are slashed and the validium is shut down. To respond, the operator must provide the relevant ownership proof. The smart contract checks the Merkle proof against the relevant historical state root stored on-chain. Note, we can avoid the gas cost of checking the Merkle proof on-chain by optimistically accepting the operator's response and allowing users to slash the operator if the response is incorrect.

Like with force_include

, the operator is incentivised to respond to requests off-chain. For force_ownership_proof

, this is because responding on-chain costs gas, and the operator can avoid those costs by addressing users' needs off-chain.

Note, letting anyone query who owns any coin is a privacy concern, but this can bemitigated with ZKPs.

Shutdown

During shutdown, users have a month to prove ownership of their coins. Every coin is numbered and newer proofs invalidate older proofs for the same coins. If a user owned a given coin when the validium shut down, they know no one else can have a newer ownership proof for that coin.

During the one-month claiming period, users post their ownership proofs on-chain. Once the claiming period is done, we run a deterministic resolution algorithm that figures out the most recent owners of each coin and outputs a Merkle tree that can be used to withdraw the funds.

[

Claiming

1496×1224 7.98 KB

](https://ethresear.ch/uploads/default/original/2X/7/71f24f61c1ed565dc5f55235a0854128a3e5b4a1.png)

The horizontal lines represent valid claims, and the green sections are the claims in the final Merkle tree.

There are many ways to approach the claiming/resolution algorithm. One simple solution is to verify every ownership proof

on-chain as it's posted and add it to an accumulator, then once the claiming period is over, someone can post a ZKP that takes the accumulator, checks every claim against one another, and outputs the Merkle root.

Optimisations

It's possible for each claim to only cost a few hundred gas by batching claims together into a ZKP so the Merkle proofs don't have to be directly verified or even posted on-chain. This makes the cost of a claim approximately as expensive as a transaction in a ZKP rollup. However, that algorithm is beyond the scope of this document.

Even if the claiming process is optimised to a few gas per claim, the limited throughput of L1 puts an upper bound on the number of claims that can be made. Moreover, the gas price spike caused by the shutdown of a very large ArcPay-style validium may make trustless withdrawal impractical for users with smaller deposits. Instead of shutting down entirely, we can put the validium into receivership, where users have the option of proving ownership on L1 as usual, or off-chain to a new operator. This requires ideas around promises described below, and is also beyond the scope of this document.

Instant Finality

Many rollups have centralised sequencers, which they use to offer a kind of instant finality. For exampleArbitrum offers "soft finality", where the user can instantly learn the result of their transaction if they trust the sequencer.

In ArcPay, instant finality works similarly, except that it's trustlessly enforceable.

The ArcPay operator responds to off-chain transactions with promises like Bob will have tokens [87, 97] in block 123

. Later, users can use ownership proofs to show that the promise was broken. If a promise was broken, the operator must reimburse the user with ~100x the coins that were promised, or the operator will be slashed.

Example Transaction

Suppose Alice is buying an orange from Bob. She sends her transaction directly to the operator, who signs it, promising that the tokens will go to Bob in block 123

. Alice gives the promise to Bob, who takes it as a strong guarantee of payment and gives the orange to Alice.

[

Alice and Bob Interaction

1804×1108 55.2 KB

](https://ethresear.ch/uploads/default/original/2X/1/15bf28d5edbaaf5427b55fea2accb0329658491f.png)

Bob holds on to the promise until block 123

, when he asks the operator for the ownership proof for coin 87

at block 123

. If the operator is honest, they respond with a proof showing that they honoured their promise.

[

Bob and Sequencer Interaction

1804×788 44.1 KB

](https://ethresear.ch/uploads/default/original/2X/9/9cc5fab679cb1a529b31f09557735e2042a121fd.png)

If the operator doesn't respond, he uses force_ownership_proof

as described above. If the operator didn't honour their promise Bob will prove it on L1 and demand compensation. If there is no compensation, the operator is slashed and the validium is shut down.

Forcing Delay

To make sure promises can be honoured, force_include

must have a delay. If there were no delay, Alice could get the operator to promise that Bob will have tokens [87,97] in block 123

, but then use force_include

to send coins [87,97]

to someone else, meaning the operator can’t honour their promise.

In ArcPay we do this with two accumulators, where one is locked and the other is unlocked at any given time. force_include

adds transactions to the unlocked accumulator. When the state is updated, the locked accumulator is passed as input to the proof, which shows that all the transactions have been included. Then locked accumulator is zeroed out, and unlocked, and the unlocked accumulator is locked.

[

Tx Holding Pen

2464×2060 109 KB

](https://ethresear.ch/uploads/default/original/2X/5/559c2120e3549c9f246a96d5e0b9e9d9b6471a0f.png)

The operator can safely make promises about the state of the next block because they can know all the forced transactions that will be included in it by looking at the currently locked accumulator.

Hot Potatoes

Users may want to send their coins within the same block that they recieve them. However, for that to happen, the operator has to promise that two different parties will own the same coins in the same block, and this will leave the operator liable for breaking a promise.

The operator isn’t forced to make the second promise, but if we want to implement same block transactions, we can let the operator dismiss an allegation of a broken promise by proving that the user owned the coins in between blocks. To do this, the smart contract stores a Merkle root of intermediate states for each block which lets the operator prove that the promise was fulfilled as some point between the blocks.

Privacy

Unlike rollups or L1 chains, validia have privacy by default (assuming they use true zkSNARKs and a hiding hash function for the Merkle tree). ArcPay has trusted privacy in the style of web2 platforms - you and the company running the server know your financial details. Full privacy in the Zcash/Aztec style would require another layer of ZKPs, and introduces regulatory complexity, especially for a centralised entity.

However, the force_ownership_proof

function allows anyone to forcibly request the owner of any coin. To fix this, we only let people use force_ownership_proof

for coins they own. Specifically, if the coin doesn’t belong to the requester, the operator responds with a ZKP proving that it doesn’t belong to the requester. This way, people can still acquire the ownership proofs they need to safeguard the coins in case of a shutdown, and they can still prove whether the operator broke a promise. To prove a broken promise, the user just presents the promise that Bob will have coins [1,10] in block 100

, and the ZKP showing that Bob doesn't own coins [1,10] in block 100

to the chain and demands compensation.

Cheap Provers

Currently, there is a tradeoff between prover time and verifier time for all production-ready SNARKs. Generally, validia and rollups use Plonk-like systems or Groth16 for their fast verifiers and small proof size. On the other hand, STARK or Spartan proofs are cheaper to generate but are less practical to verify on-chain.

Because ArcPay has a centralised operator with a large stake, we can optimistically verify our state transition proofs. The proofs are only checked on-chain if someone finds that the proof is invalid. Then they get the proof checked on-chain, slash the operator, and earn a reward. The state is reverted to the last valid state and the validium is shut down. There must be a period where anyone can take the reward from the original slasher by showing that there was an earlier invalid proof.

Note, this means there must be a withdrawal delay, and it weakens the security model from cryptographic to game theoretic. In the long run, there will probably be proof composition solutions that enable fast provers and fast verifiers, and optimistic verification will become unnecessary.