[EDIT: later replies describe a simpler version of this idea without the Plasma elements]

[EDIT 2: Plasma Cash by Vitalik and Karl Floersch improves significantly on these ideas in the Plasma context.]

This post describes an adaptation of Minimal Viable Plasma (MVP) by @vbuterin et al (and assumes familiarity with that post). The adaptation makes use of some of Peter Todd's ideas around single-use seals and client-side validation. Almost none of the ideas below are original to me (although the mistakes probably are).

This approach makes some serious tradeoffs relative to the blockchain structure described in MVP, but it could be a useful configuration for some minority of Plasma chains, particularly for its unique probabilistic "tumbling" capability. Mostly, I wanted to introduce these ideas of Peter's (which I think deserve wider attention and could have interesting applications in other contexts, such as stateless validation in sharding) to the community, and try to sketch out a design for a working system that would demonstrate them.

Abstract

Like Minimal Viable Plasma, client-side Plasma would use a UTXO-based blockchain operated by some owner or owners, whose block headers are committed to the Ethereum blockchain. The primary difference is in the structure of those blocks.

Instead of transactions including their inputs and outputs in cleartext, they only include a commitment

to the outputs, and signatures from the inputs' public keys. It is the user's responsibility, when sending a transaction, to also provide the recipient with a recursive proof that the inputs are valid (such as by revealing all hidden information about the full chain of previous transactions, back to the deposit transactions).

There is an optimization that can be used to prune part of these proofs. In addition to making these proofs more efficient, this allows you to permanently hide part of a transaction's history, thus letting users "tumble" funds to obscure their source.

Benefits

- Plasma transactions are even smaller
- Plasma blocks can be validated statelessly. This is particularly important because in Plasma, this is a responsibility of all parties maintaining UTXOs on the chain
- Transactions themselves reveal almost nothing about their sources, destinations, or amounts. Histories eventually need to be revealed, but can be probabilistically pruned before doing so, permanently hiding some history

Costs

- Exit transactions are larger (approximately O(N) in the average number of steps in the UTXO's transaction history)
- Would-be challengers need to preserve O(N) storage, where N is the number of transaction inputs in the Plasma chain's history
- Transacting parties need to maintain and exchange off-chain proofs of not-insignificant size (the exact size depends on the amount of state maintained by the receiving party)
- Dishonest users may be able to "gamble" on the chain, with negative expected value but with some risk of hurting depositors or other users

Data structures and validity rules

Transactions

A transaction is a tuple: (destination, signatures)

The public keys—i.e., inputs—for the transaction can be derived from the signatures

, with the destination

as the message.

A transaction is valid and can be included in a block if:

- the signatures

array is non-empty and valid public keys can be recovered from each of the signatures

, or

- the signatures

array is empty, indicating a deposit transaction, it is the only transaction in that block, and there exists a corresponding deposit transaction in the parent chain

A destination

is the root of a Merkle sum tree of UTXOs (i.e., a Merkle tree where each intermediate node (and the root) also commits to the sum of the amounts of the UTXOs included in that branch).

A UTXO is a (salted) commitment to a tuple: sha3(publicKey, amount, salt)

.

Blocks

A block header is a tuple: (height, previousBlockHash, transactionRoot, publicKeyRoot)

.

A full block also includes a list of transactions. A block is valid if it contains only a single valid deposit transaction (see above), or if:

- The transactions are all valid
- The transactions are sorted in ascending order based on their derived public keys (for efficiency of computation of the publicKeyRoot

)

- The derived public keys used in the block are unique within that block
- The transactionRoot is a Merkle root of the transactions
- The publicKeyRoot is a Merkle root of the derived public keys in the transactions in the block, sorted in ascending order

A proof of non-inclusion of a public key is a Merkle path to two adjacent public key, showing that there is no transaction spending from a particular public key in that block.

(There are likely ways to make these proofs smaller using different kinds of trees that commit to the frontiers of public keys rather than the keys themselves, and/or adding probabilistic data structures like Bloom filters to the headers, but we'll leave those for follow-up posts.)

Light proof of validity

A key distinction of this protocol from shared-validation protocols like Bitcoin and Ethereum (but which it shares with some proposals around separating consensus and state execution) is that the mere inclusion of a UTXO in the Plasma blockchain does not

prove that the UTXO is valid. To spend a UTXO from the Plasma chain, you need to provide the recipient with an off-chain proof that the UTXO is valid as of the block in which it is spent. Similarly, to withdraw a UTXO from the Plasma chain, you need to prove to the Ethereum chain that the UTXO is valid as of the block in which it is withdrawn.

A light proof of validity of a UTXO as of block b

at height h

is:

- A proof that the UTXO is included as part of a valid Merkle sum tree path from destination

.

- A transaction

that includes that destination

.

- The block height h′

of the block b′

in which transaction

was included

- The index

of transaction

in that block.

- If transaction

's publicKeys

list is empty, indicating a deposit transaction, then the proof is complete. If not: * A list of UTXOs, inputs

, whose respective publicKey

s match the publicKey

s in transaction

, and whose amount

s sum to the total amount committed to by destination

.

- A light proof of validity of each UTXO in inputs

, as of block b′

at height h′

.

- A list of UTXOs, inputs

, whose respective publicKey

s match the publicKey

s in transaction

, and whose amount

s sum to the total amount committed to by destination

.

- A light proof of validity of each UTXO in inputs

, as of block b′

at height h′

.

A light proof only contains the non-public

information needed to validate a transaction. It does not demonstrate everything needed to determine that a UTXO is valid. Specifically, for a UTXO to be valid, the following must also be true:

- transaction

was included in the block at height h′

that is in the history of b

, and

- there is no transaction included in a block between blocks b′

and b

that spends publicKey

.

Fortunately, these facts can be verified based on public information, so if a verifier is running (or trusts somebody who is running) an “archival node” for the Plasma chain, they can verify these facts.

Fraud proofs

Additionally, if one of those facts is false, any archival node can detect it and construct a relatively efficient fraud proof to demonstrate that fact. This means that a light verifier can satisfy itself as to the validity of a light proof using an incentivized challenge-response protocol. This is how the parent chain verifies the validity of a UTXO for a withdrawal. During the withdrawal waiting period, any party can claim a portion of the withdrawer’s deposit by revealing a Merkle proof that either:

- a different transaction was included at position index

in the block at h′

in b

's history (or no transaction at all was included at that position), or

- there is a transaction spending publicKey

in some block between b′

and b

.

If the verifier has a list of all previous block hashes (as the Plasma contract on Ethereum does, for example), a fraud proof has size log(N), where N is the number of transactions or public keys, respectively, included in the block used in the fraud proof. If the verifier does not have such a list, the prover must also provide the chain of block headers from the block mentioned in the fraud proof to block b

.

Full proof of validity

In some cases, a verifier may not be running an archival node, and may not be able to take advantage of a challenge-response protocol. In that case, the prover has to provide some additional information.

A full proof of validity of a UTXO is a light proof of its validity, plus:

- The block header for b′

, and a Merkle proof that transaction

is included in its Merkle root.

- All of the block headers between b′

and b

.

- For each of those block headers, a Merkle proof of non-inclusion of publicKey

in the publicKeyRoot

. Since the public keys in that tree are ordered, this can be done with a Merkle proof of adjacent public keys.

Probabilistic proof of validity

This protocol gives us essentially the same functionality as minimum viable Plasma. It reduces transaction size, and makes block validation—i.e., the task that must be performed constantly by any participant on the Plasma network—efficient and memoryless.

However, while it delays the public revelation of transaction histories, those histories must eventually be revealed when a UTXO is withdrawn. Once every UTXO in a Plasma chain is withdrawn, every transaction in its history will have been

revealed. Additionally, since transactions can have multiple inputs, the size and verification time of validity proofs (even light proofs) will tend to blow up quasi-exponentially, as you must provide every thread of a UTXO's history.

However, there's a trick that allows you to linearize this history, so the size of a light or full proof is only proportional to the length of the average

history of the coins in that UTXO, rather than the total

history. This will additionally allow us to permanently prune (and thus untraceably hide) a portion of each coin's history.

To do so, we change the rule for validity of a UTXO, so that a UTXO is considered valid if one of its inputs, chosen randomly (and weighted by the amount of that input), is valid. For example, suppose a transaction has one output worth 4 ETH, and has two inputs, one of which is a valid source of 3 ETH and the other of which is a fraudulent source of 1 ETH. 75% of the time, the first one will be checked, and 25% of the time, the second one will be checked. The expected value of fraud will be $\frac{3}{4} \cdot 4 + \frac{1}{4} \cdot 0 - 3 = 0$

. Indeed, the expected value of fraud should always be 0, which means that the total supply of coins in the contract will not tend to inflate.

To strengthen this guarantee, and to discourage users from treating the Plasma chain as a casino, you would likely want to tweak the probabilities so that the expected value of fraud is somewhat less than 50%. For example, you could have a rule that 10% of the time, every input must be checked, which would mean that an attacker would expect to lose 10% of their capital with each attack.

The random choice of which coin is checked must be deterministic but uncontrollable and unpredictable by the transaction's creator. (Finding a secure randomness beacon is a difficult problem, but one with several plausible solutions.) At some point after a transaction is included in the Plasma chain, this random number would be finalized, and the holders of a UTXO would be able to prune all but one of the proofs from its history (although it would need to replace it with a proof of the result of the random beacon).

This technique allows you to shorten both full and light proofs of UTXO validity. It also turns the Plasma chain into a sort of trustless probabilistic tumbler. Given a large enough supply of "clean" coins, you would eventually be able to make any coin untraceable.

Unfortunately, this may still allow the attacker to grief the depositor and other coinholders on the Plasma chain. Computing the griefing factor is surprisingly difficult and depends on some surprising factors (happy to discuss more) but intuitively it seems like these attacks would tend to increase the contract's overcapitalization—since the expected value for attackers is negative, so each successive griefing attack will be less and less likely to hurt the honest users of the Plasma chain.