This is a proposed extension to Plasma Cash, partially inspired by anon-going Twitter discussion about the taxonomy of layer 2 scaling solutions and how they relate to each other, as well as discussions with @kfichter, @vbuterin, @jcp, @karl, and others.

Mechanics

In Plasma Cash, the transaction Merkle tree is divided into slots, each of which stores a fixed denomination of ETH (the amount that was deposited) and tracks a public key. Each transaction in that slot updates the public key currently associated with that coin.

In Plasma Debit, each slot would track not only a public key, but a number a

between 0 and v

, where v

is the total amount of ETH that was deposited into that account on the main chain. v

does not change, but a

can vary in each transaction, and represents the portion of the current coin that is owned by the owner. The remainder of the money deposited in that slot is owned by the operator. A transaction in which a

is changed, like any other transaction, requires a signature from the current coinholder. (The operator does not need to sign the update, since their consent is implied by their inclusion of the transaction.)

When a coin is withdrawn, the coinholder receives a

, and the operator receives v

– a

. (The exit rules need to be altered slightly to allow either the owner or

the operator to exit a coin.)

When a coin is initially created, a

is equal to v

. The operator can increase v

(and thus increase their balance held in the coin) by depositing additional ETH into that coin, using a separate operatorTopUp

function on the parent chain's contract.

This would allow users to pay arbitrarily small fees to the operator. (Vitalik proposed almost this exact solution to the fees problem here).

Even more usefully, however, this allows any user on the Plasma Cash chain to pay any amount to any other user on the chain (as long as the recipient's coin is undercapitalized by a sufficient amount). They can do this by creating an atomic transaction (i.e. a transaction that updates multiple coins and is only valid if it is included in both slots in the same block), where the sender's account is debited by that amount, and the recipient's account is credited by that amount.

For example, suppose Alice has an 8 ETH coin with a 4 ETH current balance, and Bob has a 5 ETH coin with a 3 ETH current balance. Alice would be able to send 1.3 ETH to Bob, by creating a transaction where the balance of her coin is reduced to 2.7, and the balance of Bob's coin is increased to 4.3. The operator will accept and include the transaction (in both Alice's and Bob's slots) because it does not change the operator's total balance (which remains, with respect to those two coins, at 6 ETH).

Theory

The above explanation defined Plasma Debit as Plasma Cash with partial balances.

However, there's an equivalent way of approaching this construction. Each Plasma Debit coin is essentially equivalent to a bidirectional payment channel (similar to the multisig-based payment channels used in the Lightning Network) between the current coin owner and the operator. The only difference is that the state of the payment channel is regularly notarized on the main chain (in a Merkle root with the operator's other channels).

This notarization gives Plasma Debit coins two significant capabilities that payment channels do not have:

1. Assignability. A Plasma Debit coin can be transferred from one owner to another (the same way a Plasma Cash coin can). This is equivalent to allowing one of the participants in a payment channel to permanently assign their interest in that payment channel to some new party, which is not possible with previous payment channel constructions. Most significantly, this allows new parties to join the payment network without doing an on-chain transaction (which is a disadvantage of the Lightning Network).

2. Atomicity. Plasma Debit coins can be updated atomically with other coins on the same Plasma chain, without the need for HTLCs. (There are limbo-exit-like edge cases around data availability that make this slightly less of a win than it seems at first, but it is still potentially powerful.)

Limitations

- To receive one of these kinds of transfers, the recipient must have (or receive) a coin that is undercapitalized by at least the amount of the transfer. This is similar to the constraints on Lightning Network channels, where the net balance between yourself and your counterparty is constrained by the balance that was initially deposited into your channel, and may require significant capital lockup (particularly on the part of the operator). However, the assignability of channels should make it easier to work around these limitations.

- This simple design only works for Plasma Cash chains with single operators. However, you could implement a more general form of this idea using merged and split coins, as described below, which would not be subject to this limitation.

Extensions

- Updates to the balance would not actually need to be included in the Plasma blocks, since they require only the mutual consent of the coinholder and operator. The exit game could be altered relatively easily to allow the operator and current coinholder to instantly update their balance by exchanging signatures on a state update (which would then be almost exactly equivalent to a payment channel). The only transactions that need to be notarized (i.e. included in Plasma blocks) are those that either change the owner of the channel, or involve multiple coins.

- If coins can be split into arbitrary denominations and remerged with their siblings (as has been proposed, i.e.here), you could implement a more general version of this, where the “residual” amount of the coin could be owned by any party, not just the operator, and where any party could act as a liquidity provider for routing payments (by agreeing to remerge and resplit separate coins atomically). This is more complex may be more powerful, and might be necessary if the Plasma Cash chain has more than one operator.

- By increasing the expressiveness of the “scripting language” beyond just single public keys, we could adapt Plasma Debit to permit coins to act as more sophisticated state channels. Indeed, if we add support for hash locks and timeouts, Plasma Debit accounts could implement HTLCs, and thus have feature parity with Lightning Network channels. This would essentially allow a Plasma Cash chain operator to act as a Lightning hub, and interoperate with Lightning network participants (including participants on other cryptocurrencies), but with the added benefits of Plasma Cash (such as the ability to transfer ownership of a state channel from one user to another).

- It may be possible to implement transactions that are atomic across multiple Plasma chains

(though this will need to be a subject for another post). If these are possible, then it would be possible to use Plasma Debit to make cross-Plasma-chain payments without HTLCs (as long as someone has an account on both chains and is willing to route the payment).