(Note: This is a relatively new idea, so there might be some flaws!)

Imagine a Plasma Cash chain in which each coin is a payment-channel between you and some arbitrary user (Unlike Plasma Debit where coins are payment-channels between you and the operator)

When you deposit your funds to the contract, it creates a bunch of “uninitialized” payment channels for you. Your funds may get equally distributed between all your new channels or be distributed with some strategy you have told to the contract, based on your needs.

All channels are uninitialized

in the first place. (Which means, the channel owner has not specified a second participant for these channels yet)

For initializing a channel, the channel owner has to create a message containing the public key of the second participant and submit it to the operator. The operator puts the message in the corresponding leaf of an Sparse-Merkle-Tree and regularly publishes the merkle roots.

An initialization message of a channel in k

th leaf in block N

is considered valid if:

- It has been signed by the creator of the channel.
- All k

th leaves of blocks 0 \leq i \leq N -1

are either null, or have invalid content (Not signed by the channel creator)

This can make sure that a channel can be initialized only once and once a channel is initialized, you can’t change the second participant!

.

Let’s say Alice deposits 200ETH to the chain and gets 6 different channels, 100ETH, 50ETH, 20ETH 20ETH and 10ETH.

She wants to send Bob 25ETHs, there are four different possible scenarios here:

- She already has a channel(s) with enough capacity with Bob, so assuming that Bob is online

, they create a message(s) containing the updated balances and an incremented nonce, and they both sign and exchange it. (This message doesn’t need to be notarized in the merkle-tree! We just put initialization messages in our merkle tree for now

)

- She has some unfilled channels with Bob so she may pay part of her payment with those channels , and then initialize a new channel (By signing an initialization message and submitting it to the operator) (She might choose to initialize the channel with capacity 50ETH as she knows she is going to make further transactions with Bob in the future) and pay the rest with them. (This is still off-chain as Alice doesn’t need to call the contract!

)

- She doesn’t have enough money in the contract, so she creates a bunch of new channels (With a single contract call) and pays Bob with them.

Exit games

In a normal exit, one of the parties submit the latest state-update to the contract, and after 7 days, the exit is finalized.

There are three possible malicious scenarios in this scheme.

- Bob starts an exit from a channel that is not initialized yet. Alice should now call challengeChannelUninitialized()

, if Bob is not able to respond with an initialization message (Signed by Alice) within a fixed amount of time, his exit is canceled.

- Bob starts an exit that is initialized, but the claimed participants of that channel is not the real participants of the channel. Alice now calls challengeInvalidParticipants()

and submits an initialization message in block N

. If Bob is not able to respond with an initialization message signed by Alice, in an older block, his exit is canceled.

- Bob starts an exit from an old state-update. Alice can now cancel his exit by showing a state-update with higher nonce.

Channel capacity distribution

As mentioned earlier, once a channel is initialized, the second participant can't be changed. It might be an overkill to initialize a channel with a high capacity with someone you don't pay often. It is also unnecessary to initialize a lot of channel with someone you have large transactions with. So a uniform distribution of channel capacities isn't the best way of doing this. We might consider a solution similar to ATM machines.

Say Alice deposits 50ETHs to the contract, the contract allocates 62 uninitialized channels for her:

2 x 5ETH 4 x 2.5ETH 8 x 1.25ETH 16 x 0.625ETH 32 x 0.3125ETH

With a smart way of selecting capacities on each channel initialization, I claim that we can have a fair payment system with arbitrary denomination

and constant history size

(The history size is O(Channels)

as the channel owners need to store only the merkle-inclusion-proofs of initialization messages of their channels)

There is still a problem in this scheme, participants need to be online for each transaction. We solve this problem in the next section.

Payment Channels without unanimous consent

We define a plasma-channel, a payment channel in which the state-updates don't necessarily need to be signed by both parties. Plasma-channel states need to be notarized on-chain

in order to work. A state-update that is only signed by a single party, is only valid if the new balance of that party is less than his balance in the latest state

.

We extend the previous proposal using this scheme. Now our merkle-tree would have a valid initialization message, and a chain of state updates which should be stored (along their inclusion proofs) by both participants.

We slightly change the previous exit games and add two new challenge function, very similar to Plasma Cash:

- Say Alice tried to exit from an old state, if she had really spent some of his funds in a newer state, Bob should definitely have a message, signed by Alice, which has a higher nonce, along its merkle-proof. Bob submits this and Alice's exit is canceled. This is very similar to challengeSpent()

in Plasma Cash.

- Say Alice tried to exit from an invalid state in block N

. Bob challenges Alice by showing the contract the latest valid state before block N

and asking her to show a valid state-update after that. Since she can't do that, her exit is canceled after sometime. This is also similar to challengeInvalidHistory()

in Plasma Cash.

You might say this brings back the long coin-history problem of Plasma Cash. We provide a checkpointing strategy that is extremely simpler

than other checkpointing mechanisms in Plasma Cash. If you look carefully, you will notice that a state-update that is signed by both participants, is a checkpoint!

. We add a new rule here:

- A state-update that is signed by both parties in block N

, can cancel all exits initiated from blocks before block N

So participants can just checkpoint their plasma-channel from time to time and get rid of the older update histories.