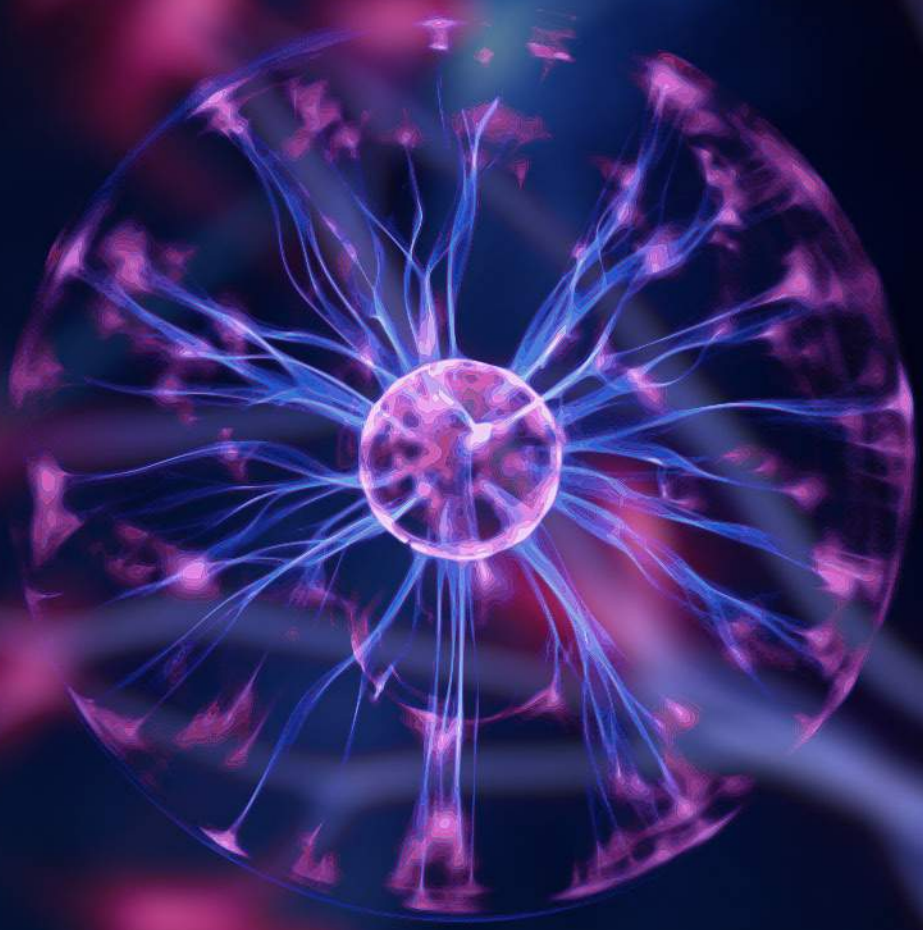
PARSEC LABS

PLASMA - FROM MVP TO GENERAL COMPUTATION



PARSECLABS.ORG

INFO@PARSECLABS.ORG

MARCH 24, 2018

CONTENTS

1. INTRODUCTION

The Plasma proposal by Poon and Buterin promises "...[a] framework for incentivized and enforced execution of smart contracts which is scalable to a significant amount of state updates per second (potentially billions) enabling the blockchain to be able to represent a significant amount of decentralized financial applications worldwide". Yet, until now only the Minimal Viable Plasma specification exists which allows to scale simple transfer-type transaction.

To enable ethereum-like general computation and rich state transitions this paper introduces a new transaction type. While maintaining the UTXO design of plasma for cheap mass exits, the new transaction type simulates accounts with persistent storage and enables user-to-contract message passing. In addition, an truebit-like interactive game is proposed to allow challenges of invalid state transitions on the parent chain.

Due to the data availability problem users of any plasma chain need to monitor the chain and be able to exit their funds as soon as they notice block withholding or invalid data by the plasma operator. The need for economic mass-exit transactions also requires that all funds are allocated to a specific account on the main net at all times. While this can be assumed for a simple UTXO chain, scripting can create ownership shared by multiple keys, or ownership entirely controlled by the code of a contract (DAO). There is no trivial way to exit these funds. The paper will propose a scheme of counterfactual instantiation that can help to safely exit funds held in smart contracts.

The proposed specification will allow developers to take smart contracts that are too expensive to operate on the ethereum main chain and move them to a plasma chain without much adaptation. We see this as an opportunity to create venues that enable gaming and other use-cases with high transaction volume, but medium to low transaction value. Another opportunity is to operate new technologies like eWASM which would require a hardfork to be used on main net, but can be enabled without hardforking on a plasma sidechain.

2. BACKGROUND & RELATED WORK

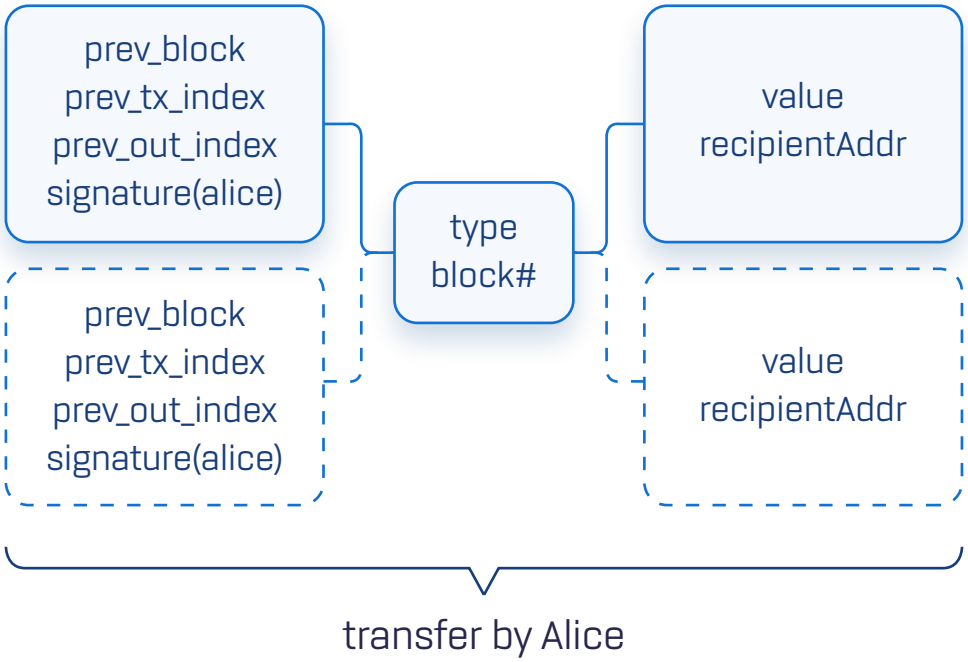
Plasma:	http://plasma.io/plasma.pdf
Truebit:	https://people.cs.uchicago.edu/~teutsch/papers/truebit.pdf
Blog:	https://medium.com/l4-media/making-sense-of-ethereums-layer-2-scaling-solutions-state-channels-plasma-and-truebit-22cb40dcc2f4
Omisego:	https://blog.omisego.network/construction-of-a-plasma-chain-0x1-614f6ebd1612
Qtum:	https://qtum.org/uploads/files/a2772efe4dc8ed1100319c6480195fb1.pdf
Plasma MVP without confirmation - David:	https://hackmd.io/o16IqtiJSgG2ez5w9Ug5aw

3. SYSTEM MODEL & PROBLEM STATEMENT

Here we discuss mitigation strategies for the data availability problem in exits and propose ways to integrate computation with the UTXO model. Finally the Truebit verification game is applied to the validator set.

3.1 Reasoning for the UTXO model

A plasma chain has general advantages in throughput, more transactions can be processed at a lower cost. While the security assumptions of the main chain are maintained, users that have funds on the plasma chain need to monitor the chain and exit, if they observe data withholding by the operators. Small balances might make some individual exits not economical. The plasma paper proposes the UTXO model to allows mass exits with as little as 1 bit of data per account needed to be submitted to the root chain.



To maintain these security and cost assumptions, and for the ability to construct efficient merkle proofs we decide to keep the UTXO model for the plasma chain. We use this logical representation for a transfer transaction here:

A transfer transaction needs to have 1 to 2 inputs and 1 to 2 outputs to be valid. The body of the transaction commits to the type of the transaction and to the block height committed to the root chain at the time of sending the transaction.

A transaction is serialized as follows. We made slight changes to the wire encoding in comparison to bitcoin and other UTXO models by using only fixed size fields.

tx structure

type	4
block#	8
#txIn	1
in_0	17-82
in_1	17-82
#txOut	1
out_0	52-64
out_1	52-64

3.2 Fighting the data availability problem with the 3-step exit game

The data availability problem opens up certain attack vectors where fraudulent operators may exit from the plasma chain with users' funds by withholding block data while commiting the block hash to the plasma contract. One possible scenario is this:

> *Transfer without commitments:*
> *t+0: user A spends UTXO_1 to user S in TX creating UTXO_2*
> *t+1: operator O mines an invalid block B_0, submits the block hash, but doesn't share the data.*
> *t+2: operator starts exiting invalid utxos on parent*
> *t+2: user S notices block withholding, but can not exit UTXO_2, as he doesn't know the position of TX in the invalid block*
> *t+2: user A notices block withholding, tries to exit with UTXO_1, but can be challenged by O with TX (which O knows position of).*

The minimal viable plasma specification mitigates certain attack vectors of the data availability problem with a prepare/commit transaction model. An exit from the plasma chain requires not only the UTXO and merkle proof of inclusion, but also a confirmation by the sender that the transaction and block have been included in the parent chain and block data has been propagated to all participants. The same withholding scenario will play out as follows:

> *Transfer with commitments:*
> *t+0: user A spends UTXO_1 to user S in TX_1 creating UTXO_2*
> *t+1: operator O includes TX_1 in block B_1 and publishes*
> *t+2: A sees hash in root chain, and signes commitment C for S*
> *t+3: S spends UTXO_2 in TX_2 creating UTXO_3 spendable by A*
> *t+4: O creates an invalid block B_2 with a TX_2 where receiver is O and withholds it*
> *t+5: operator can not exit this transaction, as a C is needed signed by S*
> *t+5: S notices lack of block and starts exiting using TX_1 and C*

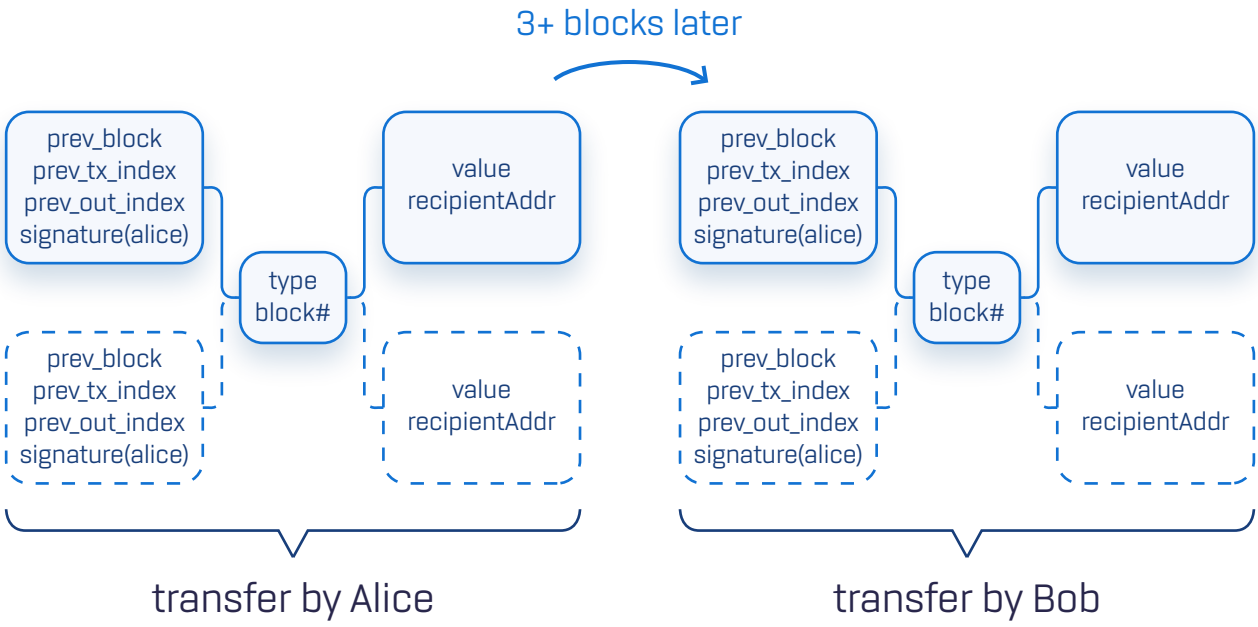
The prepare/commit model has some negative effects on user experience, and does not scale to accounts that do not have a private key, like multi-signature contracts or DAO-like contracts. David proposes a different way to mitigate the data availability problem by introducing an additional step to the exit game:

> *Rule 1: A transaction must be included within two blocks of the time it was created.*
> *Rule 2: A transactions inputs must be created at least 3 child chains blocks before.*
>
> *source: https://hackmd.io/o16IqtiJSgG2ez5w9Ug5aw*

Instead of requiring a commitment to be submitted with an exit, the exit function now requires the exiting UTXO and the transactions of the input(s) to be submitted. The first rule prevents the operator from withholding a transaction from a block for an infinite time. The second limits the exit game to at most 3 steps: the exit request, the challenge, and rechallenge request. A potential szenario looks like this:

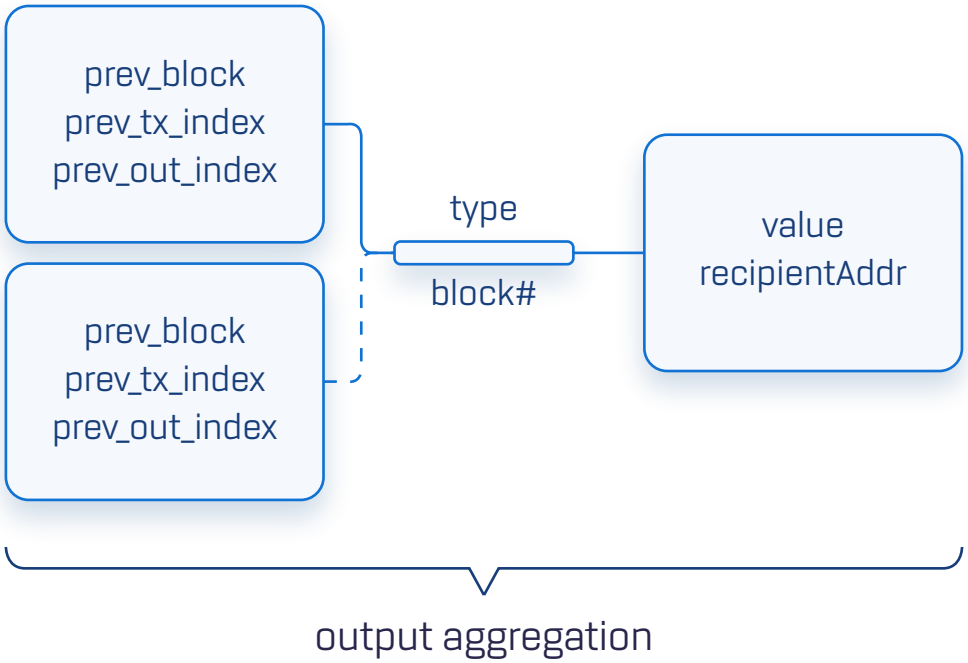
> *Transfer with David's rules:*
> *t+0: user A spends UTXO_1 in TX_1 to user S, creating UTXO_2*
> *t+1: operator O withholds TX_1*
> *t+2: operator creates an invalid block B and doesn't publish it, but submits hash*
> *t+2: S notices lack of B data, but can not exit with UTXO_2 as position in B unknown*
> *t+2: A starts to exit UTXO_1, last output with known position*
> *t+3: O mines a block including TX_1*
> *t+4: O challenges with UTXO_1 within 4 days*
> *t+4: Now S knows position of TX_1 from O's challenge, and can rechallenge with UTXO_2*

This Scenario has the disadvantage of requiring up to 3 merkle proofs, instead of only one. With the additional consensus rules by David Knott that restrict the ability of operators to cheat users we can define a secure transfer operation on a Plasma chain as follows:



3.3 Account Simulation

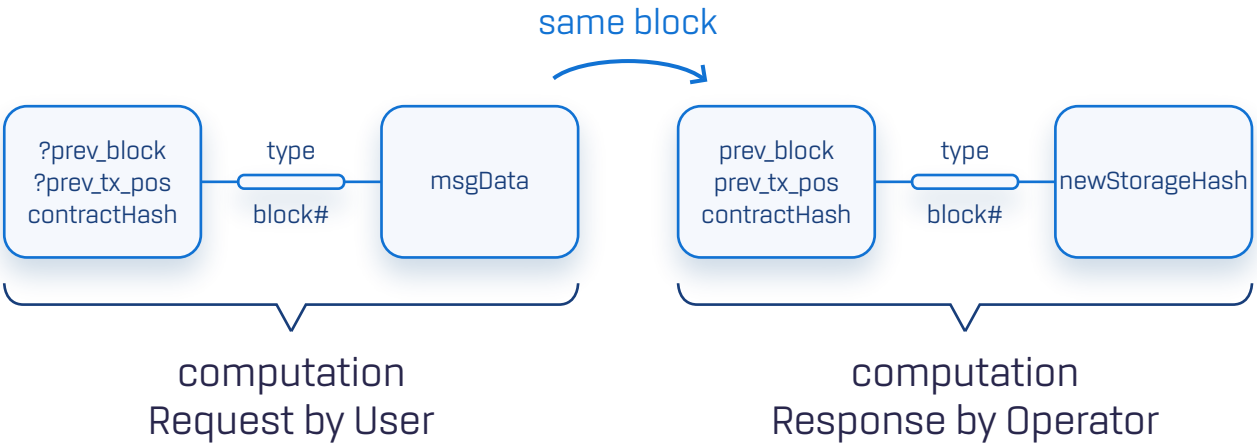
The cost imposed for exits by fragmentation of outputs can be mitigated with an additional transaction type for account simulation.
This transaction type has strictly 2 inputs and one output, which have all to reference the same owner. Because this transaction does not transfer any value, the inputs hold no signatures and it can be generated by operators or users to reduce the size of the UTXO space.



The availability of a transaction type without any authentication to the chain opens up some possibilities for grieving. Every time a user with multiple UTXOs on the chain sings a transaction to spend some of UTXO, the operator could frontrun and aggregate that UTXO with another, rendering the signed transaction invalid. This creates an incentive for a user to aggregate his own UTXOs at any time, to prevent grieving.

3.4 Integrating computation with the UTXO model

We want to enable state transitions of the form g(S, msg.data) -> S' where g() can be any general computation. We introduce transaction types which have the 1st input and the 1st output committing to a computation thread, by this creating a thread of transaction that can not branch into a tree. Each thread is represented by the hash of the code representing the function/contract g() and each transaction has to commit to this hash in it's input.

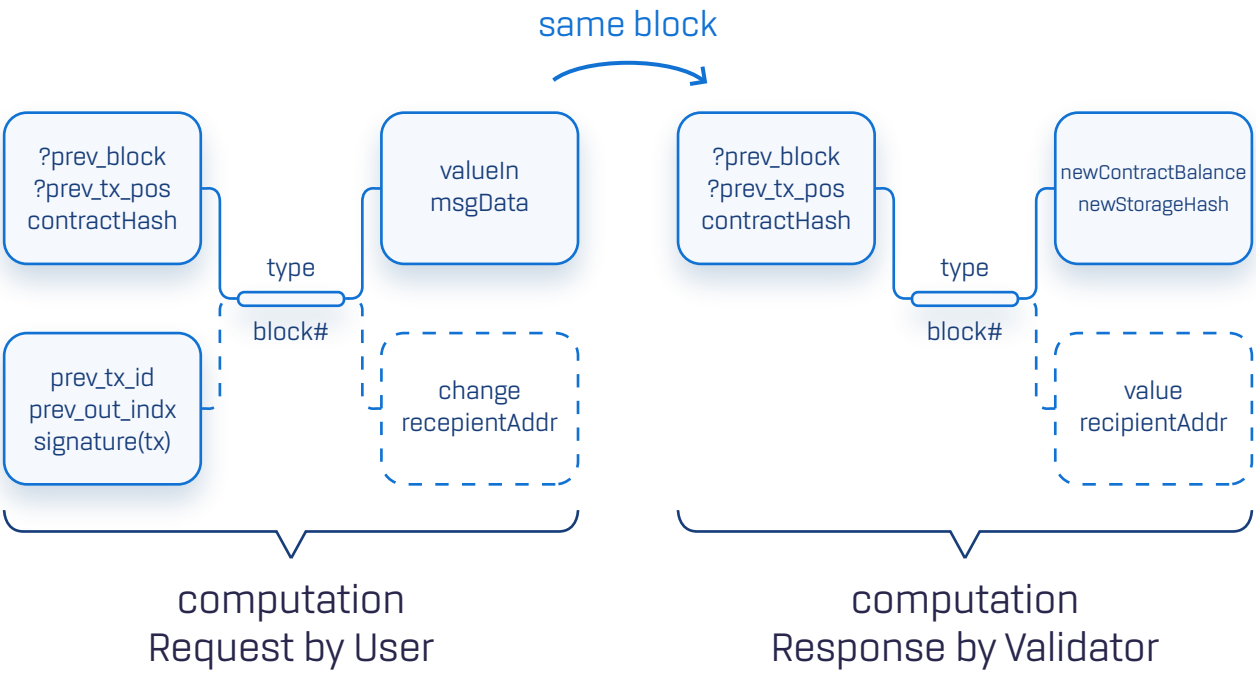


This design enables a user to submit a computation request to a specific contract, and an operator to respond to the computation request with response. The output of the response commits to the result of the computation and transforms the state of the contract thread to S'.

The reader might notice that by committing to a specific output in the computation request there will only be 1 transaction possible per block as the next utxo of the thread is not known to users before the block is not mined.

For that reason the computation request has special signing rules, where the signature of the transaction does not cover the reference to the previous output, so that the operator can chain together multiple computation requests and responses to a thread in a single block, adjusting the output reference in each of the requests.

This architecture would open up the possibility for replay attacks for transactions, hence we extend the transaction rules with a second mandatory input of type transfer:

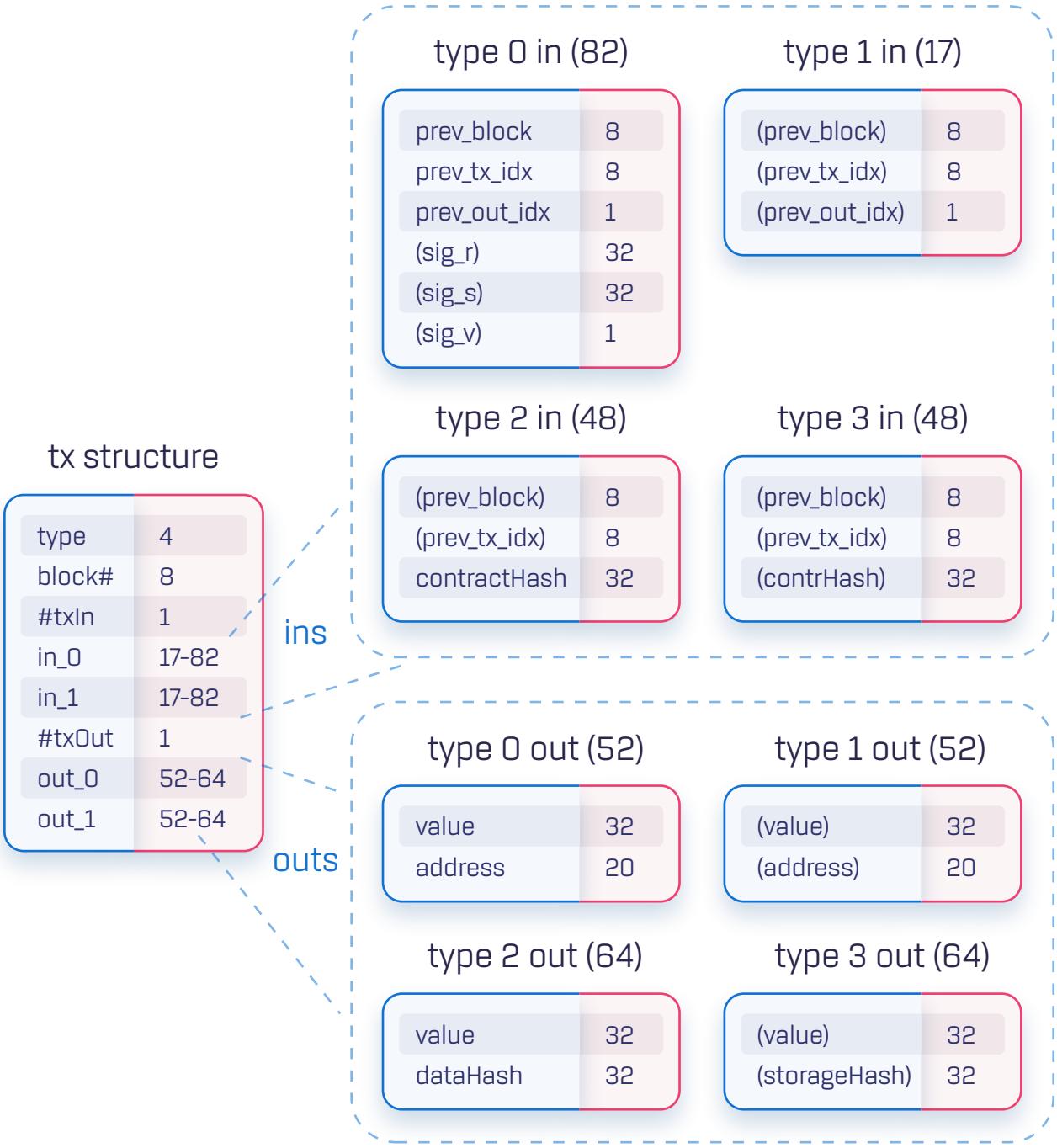


The transfer input creates replay protection on the account level of the sender while still giving the operator an ability to chain the calls in a single block. The transfer input also allows to commit funds to the function call as well as the contract thread to hold funds.

We have introduced 4 different transaction types:

Type 0: transfer
Type 1: account sim
Type 2: comp request
Type 3: comp response

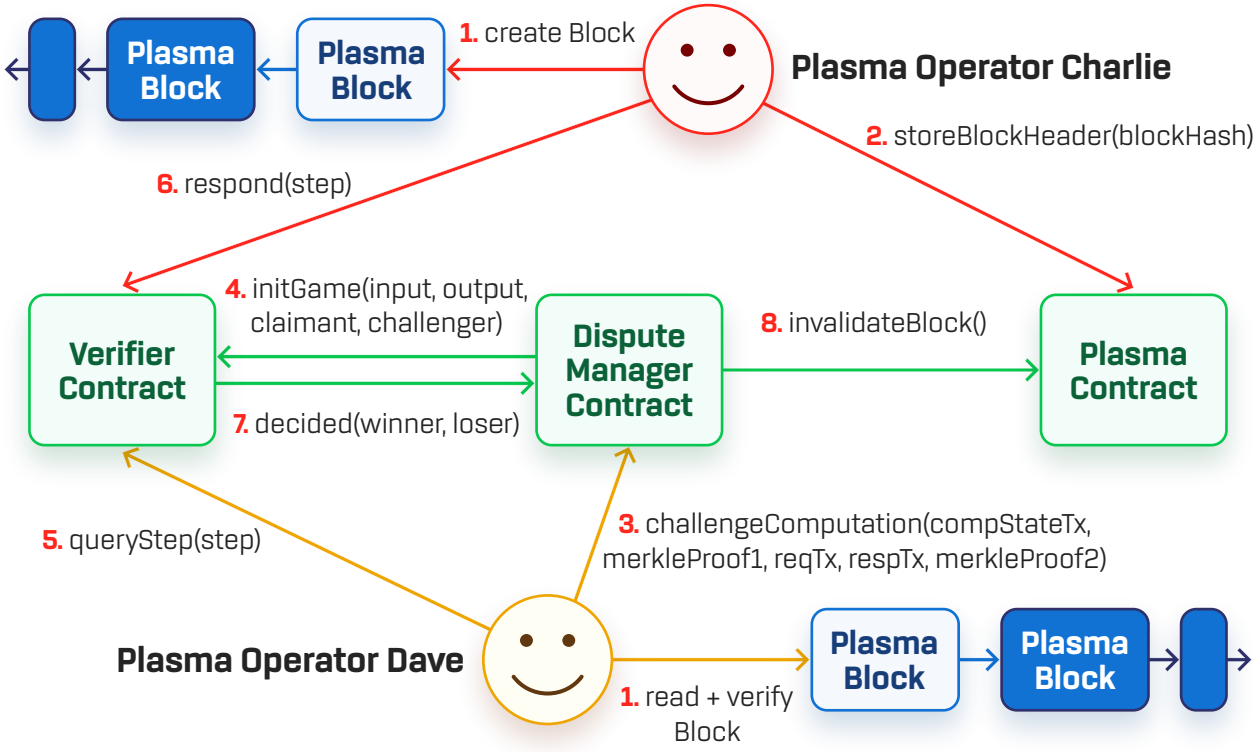
The different transaction types serialize as shown below:



3.5 Truebit verification game with operators

So far we haven't provided any guarantees for the correctness of the computation. The Truebit verification game which can reduce the verification effort of correct computation between two parties to the execution of a single operation by a trusted judge can be adjusted for the operators of the plasma chain.

The plasma contract is extended with the ability to verify computation in a multi-step interactive game between 2 participants. In this game both participants commit to the initial state of the execution engine that will run the code. The computation is rolled out from first step to last step, and the state of the execution engine is compressed into a hash after each step of computation. The game consists of a first phase, where the participants start from the half-way mark of the computation, comparing the state of the VM, and continue in a binary search until 2 consecutive states yield the same state followed by a different state of the execution engine for the respective parties. This is when the second phase starts and the state is handed over to a smart contract that is able to execute the single operation, to judge for the correct outcome.



A possible scenario to challenge a computation performed within a transaction in the latest block can look like this:

0. Two operators Charlie and Dave join a plasma chain by putting down a deposit
1. Charlie creates a block with invalid computation results and broadcasts it.
1. Dave receives the block and finds the incorrect state transition.
2. Operator submits the header of the invalid block to the plasma contract as new tip.
3. Dave opens a challenge against that block by submitting 3 involved transactions.
4. The dispute manager contracts initiates a verification game between the operators.
5. Dave challenges Charlie to reveal a hashed states of the VM at different positions of exec.
6. In a binary search a position in the execution is found where the operators disagree.
7. The contract executes the single operation and determines the winner of the game.
8. If Charlie lost the verification game, the submitted block hash will be reverted.

Notice that the provided transaction data in step 3 commits to the different elements of the transition function **g(S, msg.data) -> S'**. While the state and message data might be small enough so they can be provided to the verifier contract in a single transaction, the code defined in g() is usually quite big. We initially limit the scope of possible contracts deployed to the plasma chain with a registry. The contract body needs to be pre-deployed to the main chain before it can be used on plasma. While this sounds like a limitation, notice that this does not limit the size or execution time of contract size that run on the plasma chain. To understand the limitation of plasma computation the Plasma paper proposes:

> *«A mental framework for this is to treat the maximum memory size for computation as equivalent to the maximum of data allowed in a fraud proof.»*
>
> *http://plasma.io/plasma.pdf*

3.6 Proof-of-Stake incentives

The Nakamoto konsensus incentivises miners to front-run the computation market with block data if they have found a possible solution. This creates great incentives against data withholding in the whole network. The plasma paper proposes a construct for the proof-of-stake setup of the plasma operators to achieve similar data propagation incentives. Block rewards for bonded stakers get allocated depending on whether the past 100 Plasma blocks are representative of all stakes.

> *«For example, if someone is staked at 3 percent of the stakers, they should be 3 percent of the previous 100 blocks. If it is above that amount, the individual staker receives no additional reward for publishing extra block commitments. If the past 100 blocks, there is below 3, then the current block creator receives fewer rewards. Only one block can get allocated per block on the root chain. [] The chain tip is determined by maximum reward, if there are parallel branches, then the one that wins is the one that has the maximum fee reward from maximum coordination.»*
>
> *http://plasma.io/plasma.pdf*

Using a token rather than Ether for the stakes creates additional incentives around the liveliness of the chain. If all operators are bonded with a token and the token's value is based on the successful continuation of the chain, then a mass exit will negatively affect the market value of the token. The operators are collectively incentivised to continue the chain, avoid halting or mass exits. The presence of bonds allows every operator to participate in the computation verification game and by that keep the tip of the chain valid.

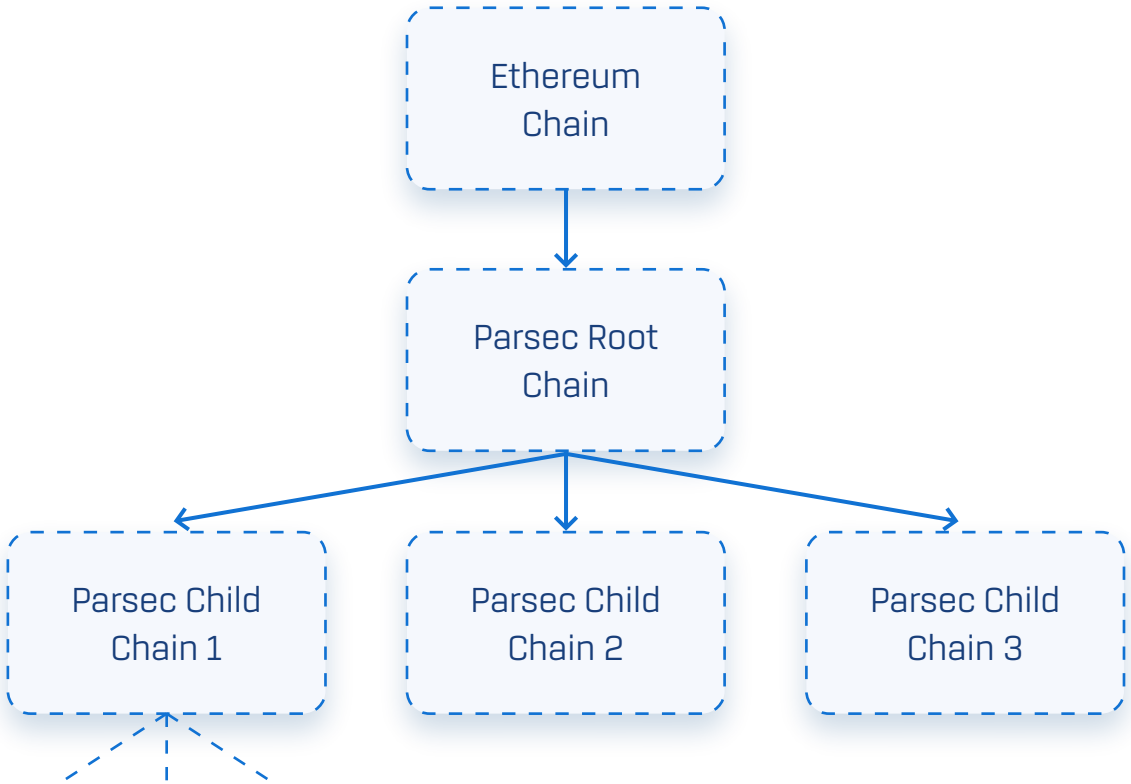
3.7 Counterfactual exits

While we have discussed several incentives for data availability on the plasma chain, the option to exit the chain should be maintained at any time. This requirement now needs to be extended to smart contracts threads that that can also hold funds. The idea is to exit the contracts through counterfactual instantiation. When a mass exit is detected on the chain, the stakeholders of the contract can initiate an exit of the funds held by the contract to a contract with predefined code at a predefined address only. Here is a potential scenario:

> *Exiting computation (david's rules) when withholding:*
> *t+0: user A spends UTXO_1 to computation request R_1 creating UTXO_2*
> *t+1: operator O withholds R_1 and computation response R_2*
> *t+2: O creates an invalid block B_1 and doesn't publish it, but submits hash*
> *t+2: A notices lack of block data, but can not exit with R_1 as position in B unknown*
> *t+2: A starts to exit UTXO_1, last output with known position*
> *t+3: O mines block B_2 including R_1 and R_2 creating UTXO_3*
> *t+3: O challenges the exit of A with R_1 within 4 days*
> *t+4: Now A knows position of R_1 and re-challenges O*
> *t+4: The contractHash in R_1 maps to contract code that will be deployed on the main net*
> *t+5: Funds exit into control of smart contract on main net*

4. CONCLUSION

The proposed architecture allows for general computation to be executed on the plasma chain.



Lifting the gas limit for execution length (as discussed, memory restrictions still apply) will enable more complex fraud proofs. More complex fraud proofs allow to prove general state transitions on account-based chains, message calls between accounts, and eventually full EVM functionality on a plasma chain.

General computation will also allow to write and deploy plasma bridges to the plasma chain, and hence start nesting chains into chains.

PARSEC LABS

PARSECLABS.ORG

INFO@PARSECLABS.ORG