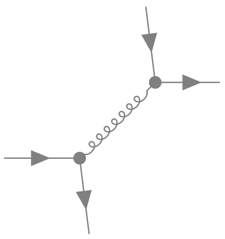


Gluon Plasma

A Plasma Variant for Non-custodial Exchanges

Bharath Rao
Leverj
bharath@leverj.io

WORKING DRAFT

October 23, 2018

Abstract

We introduce a plasma sidechain designed specifically for non-custodial, high-speed, low-latency trading. We explore extending the value proposition of plasma to a financial institution using comprehensive set of fraud-proofs to enforce correctness and voting to handle data unavailability. We describe a hybrid DEX that enables us to trade at high-speed, at low gas cost, and with support for fast withdrawals without handing over funds custody to the sidechain operator.

1 Introduction

The largest use case of cryptocurrencies is trading, which is currently dominated by centralized exchanges. Users who trade on centralized exchanges tolerate custodial risk for the benefits of low-latency and high-speed trading. We aim to break this dichotomy and present a protocol that enables low-latency, high-speed trading without custodial risks. Sidechain models such as plasma allow us to leverage the benefits of blockchain while using only a small footprint on the main chain. We introduce a flavor of plasma designed especially for exchanges to enable trustless non-custodial trading.

2 Previous Work

Nakamoto's breakthrough innovation, Bitcoin[1] made trustless transactions feasible and triggered the cryptocurrency boom that is likely still in its infancy. A currency whose rules are impractical to break even for state actors has triggered thousands of projects that hope to leverage its underlying blockchain technology to create self-compliant systems. Ethereum[2] extended the blockchain into a Turing-complete world computer. This was formalized in Gavin Wood's seminal yellow paper[3]. Ethereum became the platform of choice for smart contracts that have generalized state and logic. Plasma[4] enables a high transaction rate using a sidechain with safe exit to the main chain. Multiple flavors of plasma exist, with various tradeoffs between user burden, speed and convenience.

3 Trustless Finance

The appeal of cryptocurrency is its ability to self-regulate and confirm with its internal consensus rules without the need of a centralized enforcer. This has been variously characterized as being *permissionless, decentralized* or *trustless*.

What exactly makes something trustless? A system is trustless if all assets are always in the custody of their owners, transfers do not need a trusted intermediary and the integrity of all asset transfers can be verified by network participants with certainty.

The above definition translates to the following four constraints[1]:

1. *Segregation:* Coins are created or sent directly into the owner's custody.

2. *Agency:* Coin ownership can only change with the provable intention of the owner.

3. *Solvency:* Only legitimately created and previously unspent coins can be spent.

4. *Integrity:* Coin movement should comply with all network consensus rules[2].

Segregation enforces self-custody and removes the need for trusted third parties. Agency enables tracing a coin's provenance to its creation. Solvency ensures that the scarcity and value are preserved. Integrity ensures that reliable coin transfer can be verified by observers. All four combined enable the creation of a trustless value network.

While solvency may superficially appear to be simply another integrity constraint, it is different in an important aspect. Most integrity violations can be detected immediately from an examination of the transaction bytes, leading to the rejection of the associated transaction. Solvency (i.e., double-spend protection)

[1]Appendix A explores these in detail.

[2]Consensus rules modeled as constraints: `https://en.bitcoin.it/wiki/Protocol_rules`

is handled differently as it requires a conflicting transaction to invalidate an otherwise valid transaction. Nakamoto consensus[5] eventually resolves any solvency issues in Proof-of-work (POW) blockchains[6]. It is important to address solvency correctly in Proof-of-Authority (POA)[7] systems to avoid catastrophic losses.

3.1 A Graph-Theoretic Model of a Trustless UTXO System

We represent addresses holding coins using nodes in a graph and movements of coins as directed edges. In an unspent transaction output system, coins can be merged and split in a transaction and every transaction output is a separate coin that needs to be spent with a separate signature. An output therefore is an inbound edge into an address.

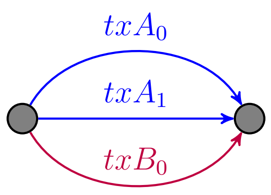


Figure 1: The same address may hold unspent outputs from multiple transactions. The address on the right holds two UTXOs from txA and one from txB

A new transaction can spend multiple unspent outputs from different transactions and distribute them to multiple recipients[8]. A transaction can be validated by checking that outputs are sent directly to the recipient (*Segregation*), all spent inputs are signed (*Agency*), only valid unspent balances are spent (*Solvency*) and the transaction complies with all consensus rules (*Integrity*). Once these rules are validated for each UTXO, they can all be marked as verified edges.

In UTXO networks, coins are created into the control of the owner. For example, the bitcoin network's coinbase transaction creates block reward coins going from *Origin*[3] into the owner's address.

An UTXO model's safety can be modeled as a valid path in a directed graph. An UTXO is valid if and only if there is a verified path to it from origin:

1. An unbroken chain of signatures (agency)
2. to the origin transaction (segregation)
3. of valid consensus confirming transfers (integrity and solvency)

[3]Bitcoin uses the UTXO with transaction Id 0, Output index -1 to denote the *Origin*[9].

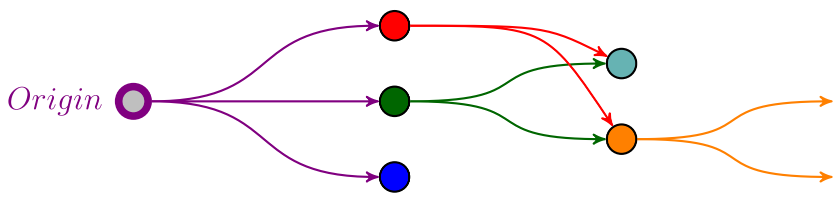


Figure 2: UTXOs as validated paths flowing from origin transaction to addresses

Verification is onerous in a large graph. Since coins are split in most transactions, it's not sufficient to do a breadth-first search. Rather, all inputs of every transaction that participates in the path should be verified as valid. This rapidly becomes a large set with every new transfer, similar to a human's list of ancestors. The number of potential paths is very large, so blockchains amortize validation by downloading every transaction and verifying them as they are added. This allows us to assume that every reference to an already verified transaction is a valid path and only verify the most recently added UTXOs.

4 Exchange Security Models

4.1 Centralized Exchanges

Custody and ownership are intertwined and transferred together in an UTXO coin, but centralized exchanges split the two concerns. A user gives up custody to the exchange when he deposits coins into his deposit address but retains ownership.

This situation gives rise to the chances of someone else losing your coins through malice or general incompetence. Exchanges try to limit this risk by splitting their holdings into a majority cold wallet[10] and minority hot wallet holdings. The cold wallet custody is vested among a small set of trusted officers to reduce risk of theft.

Centralized exchanges have no security properties since none of the four constraints are enforced. Segregation does not exist, since depositing involves essentially giving up custody to the operator; and anyone can deposit coins into any account or withdraw to any address enabling phishing attacks.

Without self-custody, there is no agency. However, an exchange may simulate agency via associated side channels such as email confirmations or two-factor authentication (2FA). Ultimately, all side channel verifications boil down to an identity check plus association of the identity with the account. This interface is not seamless and provides for a variety of attacks. Intertwined identity and agency using crypto addresses and signatures are stronger than any mobile 2FA side channel association and authentication[11].

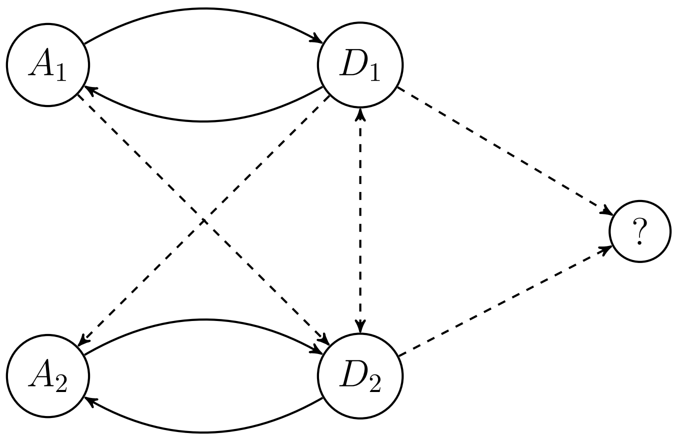


Figure 3: Centralized exchange showing no constraints on funds movement among user addresses A_i and deposit addresses D_i, rendering security proofs infeasible.

There is no simple way to prove solvency of user funds continuously. Even with periodic proof-of-reserves, there is no way for the exchange to prove that it's not running a fractional reserve since there is no solvency constraints for withdrawals or other transactions. Consequently, theft is often not detected for quite some time and often operators may lose user funds in risky ventures without anyone's knowledge.

The absence of publicly verifiable integrity checks eliminates any practical way to prove the invalidity of balance changes caused by maleficent operators or random bugs in the exchange code. This situation enables exchanges to engage in or turn a blind eye to all manner of manipulation; for example, giving friends the ability to place large orders without funds backing such orders.

In summary, all major exchange problems are a result of their inability to enforce the four trustless constraints. We believe this is true for all other financial systems that may be built on smart contract capable blockchains.

4.2 On-chain Exchanges (DEX)

On-chain exchanges can enforce the above constraints quite easily since interacting with the network and posting a blockchain transaction do most of the heavy lifting. The early decentralized exchanges models are fully on-chain, i.e., they use a smart contract to hold order books or reserves of tokens that users can interact with to perform trades. Naturally, the per transaction costs of such exchanges are very high and latency is limited by blockchain speeds, usually at intolerable levels. Such exchanges are also susceptible to a variety of front-running and DOS attacks[12].

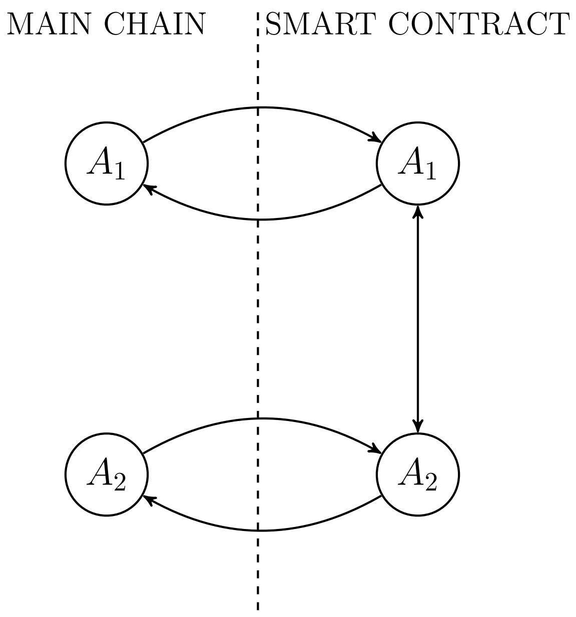


Figure 4: On-chain exchanges provide safety by syncing every transaction to the blockchain but are constrained on scale and latency.

4.3 Hybrid Exchanges

A hybrid exchange uses a smart contract to hold user funds and enables trustless trades between participants facilitated by a centralized entity. Although this can be a lot faster since orders don't need to go into the contract until filled, proving safety is a bit more involved[12].

Since the smart contract can identify the source address and the network has already verified the user signature, its trivial to credit deposited funds directly into the custody of the sender, making enforcement of the segregation constraint straight forward — the operators of the exchange never take custody of funds.

This scheme also facilitates solvency checks ensuring that the exchange is never at fractional reserve. The integrity checks are a bit more involved and require proving orders and fills are unique, have not been replayed and there are no race conditions between fills and cancels. This is accomplished by storing filled and cancelled orders or execution in the contract resulting in high costs and low speeds[13]. In addition, price-time priority proofs need to be added to verify that the exchange is not skimming the users.

5 Plasma and Plasma Exchanges

One way to dramatically reduce on-chain cost and improve latencies is to use a specialized sidechain to execute trades and perform settlements. This would reduce the main-chain footprint to deposits, withdrawals and occasional commitment of state integrity to the main chain. Plasma as a sidechain on Ethereum is the most advanced option as of this writing.

5.1 Plasma Classic Outline

The original idea of plasma (Plasma Classic) has blossomed into multiple flavors to suit the needs of various projects, but the central idea is essentially the same[4]: *The sidechain is valid as long as all changes on it are verifiable. There is a way to safely exit when the sidechain is no longer verifiable.*

1. A plasma sidechain P is anchored to root chain R
2. A deposit in R creates UTXO in P
3. A withdrawal from R removes UTXO from P
4. State changes in P are periodically committed into R as block headers.
5. Block headers contain sufficient information to prove correctness of state transitions
6. Incorrect state transitions in P are rolled back (up to a point) using fraud-proofs on R
7. Exits are prioritized by earlier P blocks, to ensure fraudulent outputs fail
8. If information needed for fraud-proofs is withheld, mass exit from P to R

5.2 Plasma Exchange

The plasma exchange can scale up to the speed of centralized exchanges by avoiding the need to sync trades and orders onto the on-chain smart contract. The main-chain plasma contract that holds custody of user funds at any time has no knowledge of the exact balance of any user. It's the responsibility of the user to provide proof-of-solvency for withdrawals. In interactive plasma variants, the user may have to post a bond that would be forfeited in case of a successful challenge by another network participant.

[4]Originally posted at: `https://ethresear.ch/t/plasma-classic-compact-spec/1711`

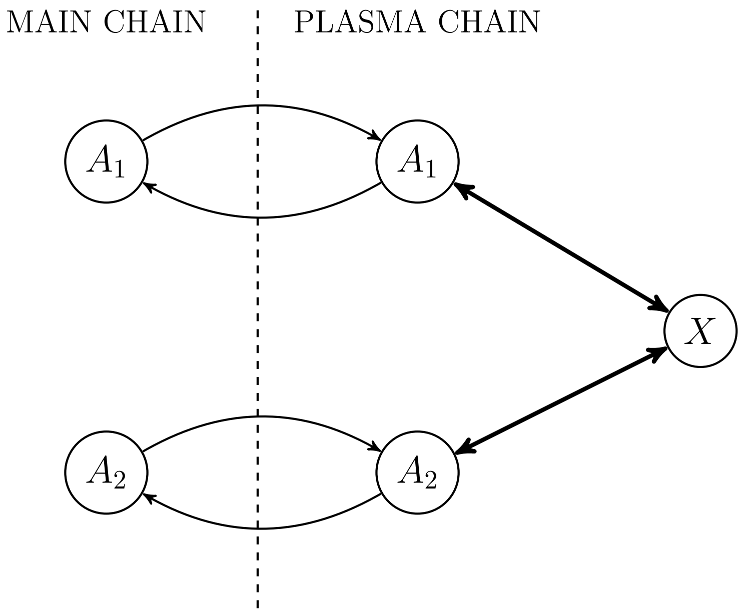


Figure 5: Plasma exchanges use smart contracts to enforce integrity without syncing trades to the main chain, combining centralized performance and decentralized safety.

The proof of safety of the plasma exchange rests on the proof that the exchange can enforce *Segregation, Agency, Solvency* and *Integrity* in all of its operations in a manner verifiable by all participants.

5.3 UTXO Based Plasma Issues for Exchanges

1. *Witness Baggage.* Agency (ownership) is proved by signatures going back to the main-chain deposit transaction. This list can get very large in an exchange with high volume.

2. *Onerous Ownership.* User is required to regularly check transactions on plasma sidechain for fungible coins or main chain for non-fungible coins.

3. *Stampede to Exit.* The huge number of UTXOs that need to exit on data unavailability suggest that the actual mechanics of exiting cannot be reliably modeled due to unpredictable congestion and fees. There is a good chance that if disaster strikes, orderly exit may be impractical.

4. *UTXO Shredding.* This is the tendency of an UTXO based system to fragment all outputs to the smallest possible size. This is a serious scalability limitation for exchanges. This is a natural outcome of price-time priority

matching. Orders are matched by best price, followed by best time of order placement. Order matches will generate a partial match for the larger sized order and a full match for the smaller size order. The exchanged assets will be received in smaller outputs with every future match and this process will repeat over time resulting in each user holding a large number of smaller and smaller outputs. This amplifies the load on the sidechain even more dramatically.

We believe UTXO model sidechains scale poorly for exchanges and therefore have constructed an account model sidechain.

6 Gluon Plasma

We introduce a variant of plasma that is designed for high speed trading engines and avoids the issues of UTXO based plasma. Our plasma sidechain is designed with the following goals in mind:

1. Account based
2. Small footprint
3. Instant finality
4. Fast withdrawals
5. Compact fraud-proofs for every transition
6. Non-Onerous safety
7. Congestion tolerant
8. Incentive balanced
9. Chain halt on data unavailability

The custody of users' assets on the plasma sidechain is managed by the Gluon plasma contract. The plasma contract serves as the interface for assets between the main chain and the plasma sidechain.

The plasma contract also accepts fraud proofs and enforces correctness. Any proof of operator compromise may be submitted to the plasma contract which results in an immediate halt of the sidechain enabling withdrawal of funds at leisure. The plasma contract also enforces a chain halt to address data unavailability.

All other features not relating to custodial issues are not a concern for the plasma contract, enabling products that use the plasma sidechain to add features without the need to redeploy a new contract.

6.1 Gluon Plasma Outline

1. Gluon plasma starts with a known zero state where all balances are zero.
2. Deposits to the main chain of an asset by user increase the user's asset balance on the plasma sidechain by a corresponding amount.
3. Withdrawing funds from the plasma sidechain decrease the balance on the sidechain and enables withdrawal on the main chain.
4. Transfer of assets (trading) between users on the plasma sidechain require matching signed orders of the users, countersigned by the operator.
5. All state changes require accompanying witnesses. Any invalid witness can be used to halt the plasma sidechain and let users withdraw at leisure.
6. In the event of data unavailability, participants can vote to halt the sidechain, enabling users to withdraw on the main chain at leisure.

6.2 The Gluon Plasma Ledger

All state changes in the plasma sidechain are recorded in a ledger that enables verification of the provenance and validity of every single balance.

1. The plasma sidechain state is maintained on its own ledger. The plasma sidechain starts with a known zero state, where all balances are zero. This is reflected by a special ledger entry called the *Origin*.
2. Every state change creates a new ledger entry retaining a link to the prior entry that it invalidates.
3. Every new ledger entry enforces trustless constraints of *Segregation, Agency, Solvency* and *Integrity*, whose validity can be independently verified by network participants.
4. Every ledger entry is signed by the operator. This reflects the fact that the operator is signing off on the changes.
5. The plasma contract will accept invalid entries signed by the operator as a fraud proof.
6. Deposits in the main chain create a new ledger entry that reflects the user's increased asset balance.

7. Ledger Entries are periodically committed to the main chain via Merkle roots. Each set of such transactions is called a *Gluon Block* or *G-block*. No transactions can be skipped or reordered.
8. All acceptable deposits occurring in a G-block are committed as an ordered Merkle root.
9. All current balances of each Account/Asset pair are committed as a Merkle root.
10. The plasma contract enforces a minimum number of main chain blocks between each G-Block to ensure sufficient latency for a vote to halt.
11. The last committed G-block is said to be *unconfirmed* and cannot be used to initiate a withdrawal. It is open to challenges from validators. Once a newer G-Block is committed, the prior G-Block is deemed *confirmed* and may be used to withdraw funds.
12. Withdrawing some of the balance on the plasma sidechain reduces the asset balance on the sidechain and enables withdrawal from the main chain. This creates a corresponding ledger entry to reflect the reduced asset balance.

6.3 Ledger Entries

Each state change is recorded in a ledger entry that includes the following:

Table 1: Ledger Entry fields in Gluon Plasma

Field	**Meaning**
Entry ID	ID for the new state
Prior Entry	Prior state
Type	Origin, Deposit, Trade, Fee, Withdraw, Exited
Account	Ethereum public address of owner
Asset	Token Address. All zeros for Ethereum
Quantity	Net Change from prior balance
Balance	New balance
Witness	Acceptable Proof of validity

6.4 Ledger Entry types

There are six ledger entry types: $Origin$, $Deposit$, $Withdraw$, $Exited$, $Trade$ and Fee. The initial state, also known as $Origin$, is a special entry that records a universal zero balance.

The *Deposit* and *Withdraw* entries are on and off ramps into the plasma contract. They increase and decrease the balance respectively. The *Exited* entry type represents a permanent cessation of all activity for an $(Account, Asset)$ pair and reduces balance to zero. *Trade* represents a trade of an asset pair between two accounts. *Fee* represents fees paid to the operator.

6.5 State Transitions

Table 2: Ledger Entry fields in Gluon Plasma. Account data are signed by the account private key and entries by operator key. The symbol $*$ indicates "any."

State	Prior	Type	Account	Asset	Qty.	Bal.	Witness
0	0	Origin	$*$	$*$	0.00	0.00	
1	0	Deposit	A_1	Z_1	$+0.10$	0.10	Deposit Hash
2	1	Trade	A_1	Z_1	-0.05	0.05	Orders, Fill
3	0	Trade	A_1	Z_2	$+0.02$	0.02	Orders, Fill
4	3	Withdraw	A_1	Z_2	-0.01	0.01	Prior balance
5	4	Exited	A_1	Z_2	-0.01	0.00	Exit Block

Every state transition has sufficient witness data to prove correctness of the new entry and a link to the prior entry. A partial set of entries for an account A_1 that deposits asset Z_1 and trades 0.05 of it for 0.02 of Asset Z_2 and finally withdrawing and exiting is shown in Table 2.

Note that $state_2$ obsoletes $state_1$ for the tuple (A_1, Z_1) but $state_3$ is a new state for (A_1, Z_2), therefore its prior entry points to *Origin*.

Fig. 6 illustrates the coin flow when an Account A_1 deposits Z_1, an Account A_2 deposits Z_2 and they trade portions of the assets with each other.

The set of state changes has the signature of all parties plus the signature of the operator and is replicated among the network. The new states obsolete the old states, i.e., the balance at $state_2 : (A_1, Z_2)$ is spent/obsoleted by an updated balance at $state_6 : (A_1, Z_2)$. Further trades among the same assets update the existing balances to new states: if A_2 sells some more Z_2, $state_6$ will be updated to $state_{10}$. Additional non-obsoleted balance entries are created only when an account acquire an asset it did not previously own.

The validity of a balance entry can be verified by tracing the integrity and witness data all the way back to the origin just like an UTXO system. Such tracing is expensive and impractical for any high-volume system. We take the same approach as blockchain based coins and amortize the cost by validating changes in real-time.

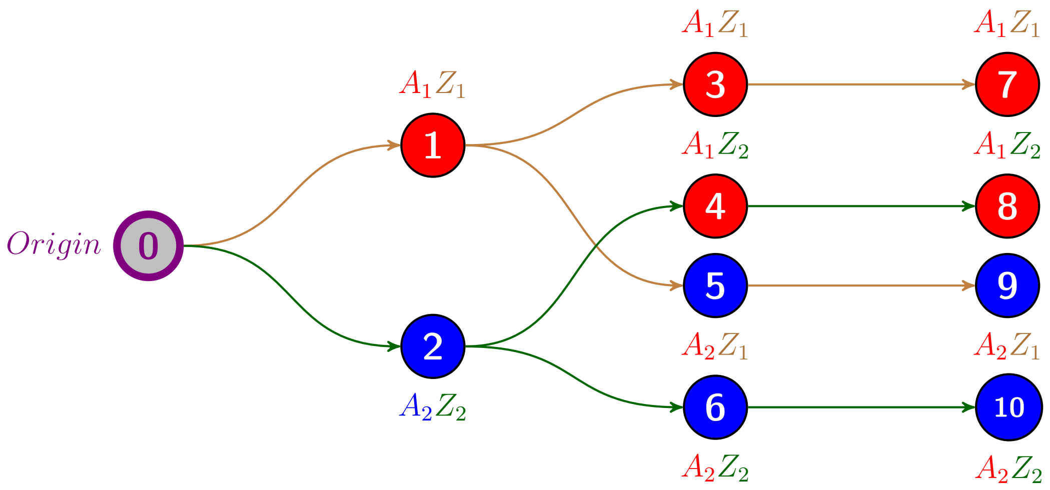


Figure 6: Plasma assets flows are a simpler version of the classic UTXO model.

6.6 Gluon Blocks

State transitions are organized into blocks that are committed to the plasma contract called *Gluon Blocks* or *G-Blocks* at regular intervals called *submissionInterval*. The commitment contains the following:

Table 3: Block Header committed to the main chain on the Gluon Plasma Smart Contract.

Field Name	**Description**
gBlockNumber	G-Block number, increasing positive integer
depositRoot	Ordered Merkle root of all valid deposits in this G-block
withdrawRoot	Merkle root of all withdraw entries in this G-block
balanceRoot	Merkle root of current balances snapshot as of this G-block

As users trade on the plasma sidechain, their balances change, and ledger entries are created to reflect every new balance. These balances are not yet in any G-Block and are considered to be in the *Mempool*. These transactions are immediately eligible for fraud-proofs if the operator has shown fraudulent behavior.

The operator accumulates entries from the *Mempool* and periodically commits the most recent transactions to the plasma contract as a new G-block. The most recent G-block committed is said to be *unconfirmed* and not usable for withdrawal, since the operator may have generated it by including invalid entries. A minimum challenge period needs to elapse to give participants a chance to verify the new G-block or submit a vote to halt the sidechain in case of data unavailability. Once the period has elapsed and a new *unconfirmed* G-block has been committed, the prior G-block is deemed *confirmed* and can be used for withdrawals. A user may also withdraw funds by showing proof of the exit claim by providing the Merkle proof of its entry in a confirmed G-block.

The root hashes above are also computed by network participants in real-time as new ledger entries are published. If the root hashes do not match the values committed to the plasma contract, there is a potential data unavailability issue and the participant can vote to halt. Naturally, participants should sync and obtain any missing entries if they have suffered from recent downtime or network disruption to avoid false signals.

Table 4: Ledger entries from Mempool *are committed to G-blocks periodically. The last G-block committed (G-block 3) is unconfirmed and unavailable for withdrawal.*

G-Block 1 (confirmed)
TX1 to TX 130
G-Block 2 (confirmed)
TX131 to TX 526
G-Block 3 (unconfirmed)
TX 527 to TX 785
Mempool
TX 786 onwards

If proof of operator compromise is submitted to the plasma contract, then all transactions in the unconfirmed G-block and Mempool are effectively rolled back. This prevents a maleficent operator from withdrawing funds using fake balances.

7 Characteristics of Gluon Plasma

7.1 Account Based

Account based plasma can be seen as a structural constraint of having only one UTXO per asset per account called the *balance*.

When a recipient receives funds, the balance of that asset type is also spent, resulting in a new unspent balance output whose value is the sum of the spent balance and received funds.

This concept collapses the multi-branch UTXO graph into a single chain and allows us to reason about account safety in far simpler terms and create a comprehensive set of simple and compact fraud proofs.

The external address that deposits the funds can be used as an account identifier enabling seamless enforcement of *Segregation* and *Agency* constraints.

This approach also simplifies fraud proofs and orchestration of transactions:

1. *No UTXO Shredding.* Even with millions of matches, the user only has one balance per asset instead of the millions that would need to be managed in a pure UTXO system.

2. *Predictable Sidechain Characteristics.* For example, we can compute bounds for exiting to the main chain in case of a compromise, which is simply the product of the number of users and assets.

3. *Compact Transactions.* Instead of picking the best set of UTXOs to fit the size of the trade and assemble them into acceptable transaction sizes, we simply have fixed size transactions.

4. *Compact Proof-of-Ownership.* We do not require a chain of signatures from the creation of the UTXO into the plasma sidechain or periodic consolidations in the main chain to manage UTXO bloat. The latest balance is the single source of truth.

5. *Compact Fixed-Size Fraud-Proofs.* A later balance invalidates all prior balances. This enables compact, fixed-size fraud proofs that can be submitted to the plasma contract at predictable gas costs.

7.2 Small Footprint

One of the important distinctions of Gluon plasma versus other DEX is the small footprint in terms of state required on both the main chain and the plasma sidechain. The main chain footprint is limited to deposits, withdrawals and periodic commitments of the plasma sidechain. The commitments are a constant cost and do not change with trade volume or user activity. Deposits and withdrawals costs are similar to centralized exchanges. On the plasma sidechain, since every new balance invalidates the prior balance, validating nodes do not need to hold the the full sidechain history.

7.3 Instant Finality

In coins with probabilistic finality (most POW and POS based coins) an observed transaction needs to be buried under work and/or time before it can be acted upon by an observer. For payment systems, a few seconds or even minutes may be acceptable for in-person transactions; much longer delays maybe acceptable for online transactions. In a trading application, finality must be instant, i.e., an observed transaction *must* be accepted and actionable. This is because a filled transaction needs to trigger other actions, which when delayed can lead to losses. An example is setting stop orders when an entry order is filled or cancelling stop orders when a profit is taken. Having to wait even for a few seconds could mean the stop order cannot be set due to market movement.

Instant finality is accomplished by having the operator sign each execution with the exchange operator's private key. This key should not have access to any funds.

Instant finality also eliminates the "free option." In systems where the two matched parties have to transmit the transaction to each other for mutual signatures, the trader who signs it last has a free option, where he can choose to sign only if the trade has moved in his favor.

Having the operator match and countersign user-signed orders also eliminates the need to be online during a match. Participants can treat the exchange as a central counterparty and the fills are non-interactive once orders are placed. Natural trading behavior such as placing an order and closing the laptop computer is fully supported. Complex order types such as OTO/OCO orders[14] are greatly simplified by making the order execution non-interactive.

While transfers within the sidechain have instant finality, transfers between main chain and sidechain have between 2 to 5 G-blocks to finality.

7.4 Fast Withdrawal

Having a POA system also enables fast withdrawals in the substantial portion of the time when the operator is behaving normally. Any faulty withdrawal can be immediately detected by any observer who can submit a fraud proof to the plasma contract.

7.5 Compact Fraud-Proofs for Every Transition

While POA is the direct route to instant finality, the authority of the exchange needs to be strictly limited to the authorized intentions of market participants according to the publicly agreed upon rules of the system. The exchange should not permit users to trade amounts in excess of their balance or create fake balances out of nothing. If a user authorizes buying 1,000 tokens for 1 ether by signing the order with their private key, then the operator should only be able to fill the order within those parameters.

Detection of any of these violations are simple, without undue burden of storage or computation on the part of the observer. We accomplish this by verifying the enforcement of all four trustless constraints for every state transition. On detecting any violations, the last valid state and the succeeding invalid state may be submitted to the plasma contract as a fraud-proof. The plasma contract halts the sidechain and enables all users to withdraw funds at leisure on submission of valid fraud-proofs.

Any aspect of the blockchain that can grow boundlessly with no or low cost will result in spam attacks and make verification onerous, causing only well-funded players to participate in the network. This includes proofs and therefore, proofs should be of constant size and easily computable. Fraud-proofs are of constant size when all information regarding correctness can be deduced from the combination of the prior state, change requests and the new state.

All validator proofs are non-interactive which makes operating a validator simple and cheap.

7.6 Non-onerous Safety

The cost of ensuring safety of the network should be cheap in terms of both effort and resources. Some plasma variants require users to check the plasma sidechain every week or so to ensure their funds are not stolen. This implies that withdrawals to main chain would take about two weeks. In contrast, Bitcoin and other classic cryptocurrencies require no verification of the blockchain to ensure your funds are not stolen. Onerous safety is equivalent to cheap attacks and vice-versa. Safety should rest on a very few actors acting honestly. In Gluon plasma, only one validator needs to be honest and present to prevent a compromised operator from stealing funds.

7.7 Congestion Tolerant

The protocol measures intervals in Gluon blocks where possible. This mitigates the effects of congestion on the main chain, enabling the plasma sidechain to continue operation mostly unaffected. Short downtimes can be accommodated without risk of protocol failure.

7.8 Incentive Balanced

Incentive-balanced protocols where malicious actors have to expend resources and effort and normal users do not have to, are robust. The larger the incentive to cheat, the more expensive an action needs to be. In Gluon plasma, simple holding an asset is free.

7.9 Chain Halt on Data Unavailability

The plasma sidechain can also be halted in case of data unavailability. Block withholding is an attack by the operator where blocks are created but some or all ledger entries are not broadcast. Block withholding can be detected but not proven since there is insufficient data available for fraud proof submission. Block withholding attacks are handled by users exiting their funds from the main chain.

Voting. This is accomplished by a threshold voting to halt the sidechain. To enable everyone to vote in a reasonable amount of time, the sidechain will be slowed based on the stake size that has chosen to vote. Since account sizes follow a power-law distribution and since the biggest accounts are likely to monitor the plasma sidechain, the very first or second large stake will push the next block by a few days giving others time to vote their stake. It takes just a few of the largest stakeholders to bring the sidechain to a halt.

Abandonment. If the operator stops creating blocks for a very long time, the chain is said to be *abandoned*, and a public method can be invoked on the plasma contract to halt the chain, allowing everyone to transfer funds to the main chain.

8 The Gluon Plasma Protocol

The Gluon plasma protocol only cares about custody. It has the following elements:

1. Deposit
2. Reclaim Deposit
3. Withdrawal
4. Exit Asset Balance
5. Gluon Block Submission
6. Vote to Halt
7. Halt Abandoned Chain
8. Fraud Proofs

All protocol steps have state transitions that the operator needs to incorporate into the ledger by creating a new entry. These ledger entries contain witness data and are signed by the *Operator*. These can be checked by any *Validator* and a non-interactive fraud proof may be submitted to the plasma contract if the witness data is invalid.

Users are normally only concerned with *Deposit* and *Withdrawal* steps. They will *Reclaim Deposit* and *Exit Asset Balance* on halted chains.

8.1 Deposit

1. Account A transfers quantity N of asset Z to plasma contract C.
2. C computes the designated G-block G in which this deposit would be committed[a] on the plasma sidechain.
3. C computes and stores hash H of $(A, Z, N, G, \mathit{nonce})$.
4. Operator X creates a ledger entry D referencing H and crediting N amount of Z to A. A can trade once D is published.
5. Operator X commits an ordered Merkle root *depositRoot* of all deposits for G-block G. Any ignored deposits need to be reclaimed by the user.

[a]The deposit visibility is on a future G-block $G = G_{i+k}$; $k \geq 3$ is necessary to ensure sufficient main chain confirmations before the deposit is visible on the plasma sidechain.

Remarks. The operator creates *Deposit* ledger entries for every deposit that it deems valid, signs them and publishes them on the plasma sidechain. These entries are committed to the main chain as *depositRoot*.

8.2 Reclaim Deposit

Account A can reclaim an operator uncommitted deposit once G is confirmed.

1. A submits $(A, Z, N, G, \mathit{nonce})$ with exclusion proof $(H \notin \mathit{depositRoot})$.
2. C computes hash H of $(A, Z, N, G, \mathit{nonce})$ and verifies that H has not already been reclaimed and has been excluded from *depositRoot* of G.
3. C marks H as reclaimed and transfers quantity N of asset Z to account A.

Remarks. While any account can transfer any asset to the plasma contract, the operator will only add entries to the ledger for supported assets. Any unsupported assets will be ignored by the operator and the user should *Reclaim Deposit* to retrieve them. An unacknowledged deposit is normal and not a griefing attack.

Ignoring unsupported tokens also eliminates some spam attacks. In addition, the operator may choose to ignore tiny deposits.

The ability to reclaim an unacknowledged deposit directly from the smart contract without requiring the co-operation of the operator is an essential requirement for non-custodial operation.

8.3 Withdrawal

1. Account A submits a signed withdrawal request for quantity N of asset Z to operator X off-chain.

2. Operator X creates a *Withdraw* ledger entry W that reflects the new reduced balance of (A, Z) and publishes it. A can withdraw once W is in a confirmed G-block.

3. A submits W with inclusion proof ($W \in \mathit{withdrawRoot}$) to plasma contract C.

4. C verifies W is valid and has not been already processed. C stores hash of W to prevent duplicate withdrawals and sends N quantity of Z to A on main chain.

5. If W is not published within a reasonable time (griefing attack by X), A should cancel all open orders for Z and proceed to *Exit Asset Balance*.

Remarks. In normal operation, the user can withdraw very fast with the co-operation of the operator. From the user's perspective, a withdraw request is submitted and approved after a short period of time following which, the user can withdraw the funds.

8.4 Exit Asset Balance

1. Account A submits ledger entry E for Asset Z with inclusion proof ($E \in \mathit{balanceRoot}$) of the last confirmed G-block to plasma contract C.

2. C registers the exit claim for E in G-block G_i.

3. Operator X cancels all open orders[a] and prevents further activity for (A, Z).

4. After k G-blocks[b], A submits proof of unchanged balance in the unconfirmed G-block G_{i+k}. ($e \in G.\mathit{balanceRoot}; e \in G_{i+k}.\mathit{balanceRoot}$).

5. C transfers balance of Z to A on main chain and marks (A, Z) as *Exited* by storing $\mathit{ExitBlock}(A, Z) = G_{i+k}$ thus preventing all further activity for (A, Z).

[a]Operator detects claims by monitoring events on the plasma contract.

[b]$k \geq 3$ is an implementation-specific parameter ensuring all transactions currently in the Mempool at the point of the initial exit claim are in a G-block.

Remarks. Exiting an account's asset balance is a permanent cessation of activity for the account/asset pair. Normally, this is only needed when the operator refuses to honor a fast withdrawal. Assets may be transferred on a halted chain using only the final step.

8.5 Gluon Block Submission

1. Operator X sends new G-Block G_{i+1} to plasma contract C.
2. C stores G_{i+1}:
   (a) Ensure G_{i+1} is valid (expected G-block number).
   (b) Ensure $block.number \geq submissionBlock$.
   (c) Store G_{i+1}.
   (d) Mark G_i as *confirmed* and G_{i+1} as *unconfirmed.*
   (e) Update $submissionBlock = block.number + submissionInterval$
   (f) Update $voteTally = 0$

Remarks. The operator must include all ledger entries since the last G-block without any omissions or reordering. The plasma contract enforces a minimum of $submissionInterval$ main chain blocks between two G-blocks.

8.6 Vote to Halt

1. Account A votes to halt chain by invoking $halt()$ on staking contract S.
2. S computes $voteSize$ and submits to C.
   (a) Let $voteSize =$ Tokens[a] staked by A
   (b) Reduce A's stake balance by $votingCost = voteSize \times votePrice$
   (c) S submits $votingCost$ to C.
3. C tallies votes and halts the plasma sidechain if threshold is met:
   (a) Update $voteTally \mathrel{+}= voteSize$
   (b) Update $submissionBlock = delayFunc(voteTally)$
   (c) Halt Chain if $voteTally > haltThreshold$

[a]Leverj uses LEV tokens for staking and governance.

Remarks. Ledger entries and G-block commitments can be used as fraud-proofs to ensure chain fidelity. If the operator withholds some or all entries from the committed block to prevent a fraud-proof submission, stake holders should vote to halt the chain.

A polynomial function $delayFunc$ ensures that when even a small size of the stake votes to halt, the next G-Block is delayed significantly, giving observers and large stakeholders sufficient opportunity to vote to cross the halting threshold.

Votes are tallied in $voteTally$. The expected outcome is big stake holders such as market makers vote with minimal delay and reach the threshold quickly and halt the plasma sidechain.

The parameter $haltThreshold$ (10% of total supply) should be large enough that a random account shouldn't be able to halt at will and disrupt the smooth functioning of the plasma sidechain but small enough that halting is highly likely if the largest four or five non-team stakeholders vote.

The parameter $votePrice$ (around 10%) determines the amount the halt invoker is willing to sacrifice to halt the sidechain. This should be low enough to be an acceptable alternative to a compromised operator but high enough to deter frivolous halts.

A speedy halt enables orderly exit and appropriate action to be taken, including deploying a new contract (in case of a bug) or a total new deployment (in case of a security breach). The ideal values for these parameters are best determined empirically with halting games with test users.

All voting schemes are tradeoffs and will never be perfect. Moving beyond voting into provable correctness that can be verified by the plasma contract itself is a research subject we are actively pursuing.

8.7 Halt Abandoned Chain

1. Account A calls $abandon$ on plasma contract C
2. C halts chain if $block.number > submissionBlock + abandonPoint$

Remarks. The Gluon plasma system transitions among states on actions by the operator, validators or stakers voting to halt. This protocol step ensures that if all other actors abandons the chain, a lone user can still transfer funds back to the main chain.

8.8 Fraud Proofs

The Gluon plasma smart contract is not aware of trades and real-time balances of any accounts but it does know about fraud proofs and how to enforce them.

The operator facilitates trades between buyers and sellers who submit their signed orders. The signature from their private key is essential to prove that the trade is duly authorized and trade execution is within the parameters authorized by the participants.

A trade execution contains the fill parameters and include both buy and sell orders along with the operator's countersignature. The operator signature prevents users from creating fills on their own and fabricating balances.

The signatures and rules of the system are encoded as fraud proofs into the Gluon contract and security is predicated on having at least one honest validator able and willing to submit fraud proofs to the contract.

8.9 General Protocol Characteristics

8.9.1 Spam Attacks

Flooding the plasma sidechain with *Deposit* and *Exit* requests is an unlikely attack vector, since the gas cost alone should be a discouragement for this attack. The net impact of a spam attack is reduced on-chain bandwidth for deposits and withdrawals.

Voting to halt costs a significant amount of governance tokens and should discourage careless voting or deliberate griefing.

8.9.2 Network Congestion/Server Temporarily Offline

If the operator is offline for a short duration of time (for maintenance or network/data center outage), balance updates, deposits and withdrawals can continue on the main chain but the corresponding ledger entries will be postponed until the operator comes back online.

8.9.3 Contract Upgrade

The operator halts all activity and submits a G-Block containing the final transactions. After the mandatory block submission delay, an empty G-block is also committed, thereby confirming the prior G-block. It then moves the plasma contract to a halted state. Users can use the *Asset Balance Exit* protocol to extract and move funds to the upgraded contract.

8.9.4 Halted State

The operator determines the cause of the halt and takes any necessary corrective action and starts a new sidechain. The old chain is abandoned and users move their funds to the new chain.

9 Gluon Plasma Fraud Proofs

We present a comprehensive suite of fraud proofs that address every possible state change. Fraud proofs are stated as assertions that need to be always true. Any failed assertion should halt the plasma sidechain.

All fraud-proofs that require any operator signed ledger entry e require verification of the operator signature to ensure authenticity.

$$ecrecover(e) = operator \tag{1}$$

9.1 Gluon Protocol Frauds

9.1.1 Block Commitment Frauds.

Data unavailability is detected as a mismatch between the observed ledger entries and their commitment. Validators compute commitment roots as they are published and if their commitment does not match the expected value, proceed to *Vote to Halt.* This also works as a catch-all fraud proof in case a new fraud that can be detected but not proven is found after the plasma contract is live.

Deposit Commitment Mismatch. Operator commits deposit in *depositRoot* but does not create a corresponding *Deposit* ledger entry.

Detection. Committed *depositRoot* does not match deposit commitment M_D computed by plasma participants using published Gluon plasma *Deposit* ledger entries for a given G-block, resulting in *Vote to Halt.*

$$M_D = G.depositRoot \tag{2}$$

Withdraw Commitment Mismatch. Operator commits a withdrawal in *withdrawRoot* but does not create a corresponding *Withdraw* ledger entry.

Detection. Committed *withdrawRoot* does not match withdraw commitment M_W computed by plasma participants using published Gluon plasma *Withdraw* ledger entries for a given G-block, resulting in *Vote to Halt.*

$$M_W = G.withdrawRoot \tag{3}$$

Balances Commitment Mismatch. Operator creates a fake ledger entry in order to steal funds and commits in *balanceRoot* but does not publish it to avoid detection.

Detection. Committed *balanceRoot* does not match commitment M_B computed for current snapshot as of a given G-block by plasma participants using all published Gluon plasma ledger entries, resulting in *Vote to Halt.*

$$M_B = G.balanceRoot \tag{4}$$

9.1.2 Invalid Deposit Fraud

Operator creates a deposit entry with a fake, incorrect or already reclaimed deposit hash H (i.e., credits an incorrect account, amount or asset)

Proof. Operator signed *Deposit* ledger entry e. The plasma contract C cross-checks the entry hash as its own creation. An invalid deposit entry causes a chain halt.

$$\begin{aligned} H = hash(e.account, e.asset, e.quantity, e.gBlock, e.nonce) \\ C.deposits[H] = true \end{aligned} \tag{5}$$

9.1.3 Deposit Reversal Fraud

Operator does not include the deposit ledger entry e in *depositRoot* enabling user to reclaim the deposit and trade without funds.

Proof. The deposit ledger entry e. Proof of exclusion of the deposit hash H of e can be used as proof. As a practical matter, since the committed *depositRoot* won't match the validator computed value, this will result in a *Vote to Halt*.

$$e.H \in G.depositRoot \tag{6}$$

9.1.4 Fake Withdraw Entry

Operator creates an unauthorized or fake *Withdraw* entry. The operator cannot steal these funds but can grief a user and prevent them from trading.

Proof. Operator signed fake *Withdraw* entry e. The plasma contract checks that the user signature does not match contents and chain is halted.

$$ecrecover(e.accountsignature) = account \tag{7}$$

9.1.5 Exit Insolvency Fraud

Exchange does not cancel *Exited* orders, enabling trading without funds.

Proof. Insolvent ledger entry e and Merkle proof in a post exit G-block. Existence of any ledger entry e after *Exited* balance is evidence of a compromised operator. To prevent loss, this proof must be submitted when the entry is still in an unconfirmed G-block G_{i+1}.

$$\exists ExitBlock(A, Z) = G_x \wedge e \in G_i.balanceRoot \implies i < x \tag{8}$$

9.1.6 Fake Exit Fraud

Operator creates a fake *Exited* entry to grief the user.

Proof. Operator signed Exited entry e. An exit (A,Z) is saved in the plasma contract and can be easily checked.

$$e.type = Exited \implies \exists ExitBlock(A, Z) \tag{9}$$

9.2 Ledger Entry Frauds

9.2.1 Fake Signature Fraud

A maleficent operator can perform a skimming attack by creating fake orders of victim accounts. The operator may sign an account order with a fake API key[15]. These orders may be used to pump prices or other forms of market manipulation.

Proof Any ledger entry e. An API key *originator* for any account is verifiable by the plasma contract from API Key registry R. The fake order in the ledger entry would not have a valid signature of the account. To ensure that ledger entries are not created by a validator to fraudulently halt the chain, we have to also ensure that the ledger entry has been signed by the Operator.

$$ecrecover(e.buyorder) = e.buyorder.account \tag{10}$$

$$ecrecover(e.sellorder) = e.sellorder.account \tag{11}$$

$$R.translate(e.buyorder.originator) = e.buyorder.account \tag{12}$$

$$R.translate(e.sellorder.originator) = e.sellorder.account \tag{13}$$

9.2.2 FEE Fraud

The operator may give an excess fee to itself say, 100% and steal everyone's funds. The operator may also extract fees from an unauthorized asset or send the fee to an unauthorized recipient.

Proof. Any ledger entry e. The maximum fees (MAX_FEE) that can be charged is enforced by the contract.

$$e.type = FEE \implies e.fill.quantity \times MAX\_FEE \leq e.quantity \tag{14}$$

$$e.type = FEE \implies e.account = STAKING\_ADDR \tag{15}$$

$$e.type = FEE \implies e.asset \in FEE\_TOKENS \tag{16}$$

9.2.3 Price Fraud

The operator may match victim orders at unfavorable prices and give its own orders a better price, enabling skimming.

Proof. Any ledger entry e that matched a limit order. Executions should fill at limit price or better .

$$e.fill.price \leq e.buyorder.price \tag{17}$$
$$e.fill.price \geq e.sellorder.price \tag{18}$$

9.3 Ledger Entry Ordering Frauds

9.3.1 Broken Chain Fraud

This is a data unavailability attack where the operator creates an entry with a non-existent prior entry.

Proof. There is no fraud-proof for this attack and would be handled by *Vote to Halt*.

9.3.2 Double-Spend Fraud

The operator creates a ledger entry linking to a spent/obsolete ledger entry, i.e., two ledger entries have the same parent. As a practical matter, this fraud will be detected exactly as the *Broken Chain Fraud* and will result in a chain halt.

Proof. Ledger entries e_1 and e_2 with same parent p (other than *Origin*).

$$e_1.parent = e_2.parent \implies parent = Origin \tag{19}$$

9.3.3 Counterfeit Fraud

When the ledger entries for a match are created, the operator may switch asset or accounts in ledger entry chain, inflate the balance of one of the accounts in a counterfeiting effort. Conversely, values may be fraudulently decreased for victims.

Proof. Most recent entry e and prior entry p for account A. Balances should add up to quantities transacted; asset and account provenance should hold.

$$e.balance \geq 0 \tag{20}$$
$$e.balance = p.balance \pm e.quantity \tag{21}$$
$$p.asset = e.asset \tag{22}$$
$$p.account = e.account \tag{23}$$
$$e.previous = p \tag{24}$$
$$e.buyorder.asset = e.sellorder.asset = e.fill.asset \tag{25}$$

9.3.4 Replay/Overfill Fraud

The operator may match the same order multiple times or match it to higher than specified quantity.

Proof. Most recent execution entry e and prior entry p for order o. The chain of ledger entries from multiple fills would need to have incorrect fill quantities to perform this fraud. We can ensure the numbers add up and no orders have been replayed or overfilled.

$$p.o.filled + e.fill.quantity = e.o.filled \tag{26}$$

$$e.o.filled \leq e.o.quantity \tag{27}$$

9.3.5 Price Time Priority Fraud

This is a versatile fraud proof that can detect a variety of exchange frauds. In particular, it can detect front running, holding orders hostage for skimming and in general, all frauds where the exchange is required to violate the predefined Central Limit Order Book (CLOB) behavior.

The proof is based on the fact that front-running and other skimming frauds are a form of illegal arbitrage that use advance knowledge of order flow. Profitable arbitrage is only possible when the operator can exit at a profitable price differential. Anything else is economically identical to a normal legal entry.

Market Orders. A victim order sufficiently large to move the price can be front-run. For example, if the bid/ask is 50@2.00/80@3.00, and if the victim places a buy market order for quantity of 100, the frontrunner would insert a buy order to take the existing 80 asks at 3.00 and place a take profit sell order just before the next order in line to be filled. Frontrunning on market orders are undetectable since market orders have the highest priority.

Limit Orders. Limit orders cannot be front-run as above since any unfilled quantity turns into a resting order. When filling deep into the orderbook, a fill-or-kill flag would ensure that the trader will not pay more than he expects.

To front-run limit orders, the frontrunner would have to hold on to the victim order and wait for the price to move against the victim and then fill both the victim order and a take profit order near simultaneously. This is observed as an order that filled out of turn, i.e., an order with an earlier entry time was filled at a worse price.

Proof. Recent entry e and prior entry p for the same asset and market side.

$$e.order.price \succcurlyeq p.order.price^{5} \implies e.order.created > p.filltime \tag{28}$$

[5] We use the symbol $\succcurlyeq$ to denote same or worse price (higher buys and lower sells.)

For price-time priority fraud to work correctly, it is necessary to ensure that the exchange does not accept orders with a clock that is far beyond the skew tolerance that would weaken the price-time priority fraud.

Proof. Any ledger entry e. The execution of orders should not precede the orders and the clock skew between the user's clock and the operator's clock must be within the system tolerance MAX_SKEW.

$$e.buyorder.exchangeTime \leq e.fill.exchangeTime \tag{29}$$

$$e.sellorder.exchangeTime \leq e.fill.exchangeTime \tag{30}$$

$$\mid e.order.originatorTime - e.order.exchangeTime \mid < MAX\_SKEW \tag{31}$$

9.3.6 Unit Fraud

Every ledger entry represents an entire state change. The exception is an execution, which is represented by four *Trade* entries and up to four *Fee* entries. The operator may put fewer or more entries to violate solvency.

Proof. *Trade* entries for a given execution *ex* should have exactly four entries and their quantities should add up to zero. Similarly, two or four *Fee* entries may exist per execution. If not, validators proceed to *Vote to halt*.

$$ex.length = 4 \tag{32}$$

$$ex(A_1, Z_1).Qty = -ex(A_2, Z_1).Qty \tag{33}$$

$$ex(A_2, Z_2).Qty = -ex(A_1, Z_2).Qty \tag{34}$$

$$fee.length = 4 \lor fee.length = 2 \tag{35}$$

$$fee(A_1, Z_1).Qty = -fee(A_2, Z_1).Qty \tag{36}$$

$$fee(A_2, Z_2).Qty = -fee(A_1, Z_2).Qty \tag{37}$$

9.4 Validator Considerations

9.4.1 Storage

Most fraud proofs require only the most recent entry, which contains sufficient information to prove its validity. These can be statelessly validated.

Solvency proofs require keeping track of the last balance of every asset for every user. When a new balance entry is observed, the new and old entries can be used to verify validity of the transaction. An invalid change results in a fraud proof submission and a valid entry results in the new entry replacing the old. The storage requirement would be: *number of minimum-balance accounts* × *number of listed assets.*

Overfill and price-time priority proofs requires retaining references to the last *Trade* and *Fee* ledger entries of every Asset regardless of user. The maximum space required is: *2* × *number of listed assets.*

9.4.2 Adverserial

Spam attacks to overwhelm validators would require an attacker to create an excessive number of accounts since listing assets is controlled by the exchange. Requiring a minimum balance to trade, minimum withdrawal sizes and a reasonable tick size should ensure that validators are not spammed by millions of dust accounts.

Future Work

Zk-snarks or zk-starks may be a succinct alternative to a large set of separate proofs. A complete chain of proof-of-correctness exist between main-chain commitments. If these can be coalesced into a single compact proof that can be committed to the main-chain and verified by the plasma contract, the data unavailability problem can be eliminated, enabling us to eliminate unconfirmed blocks and voting to halt. This is an active research subject at the time of this writing.

The same plasma chain can support multiple operators who need to coordinate the plasma operations. These exchanges are incentivized by revenue sharing to share their unfilled orders with peer exchanges. Consequently, deposits and trades on the plasma chain will appear on all exchanges, mitigating the effects of downtime of a single exchange.

Acknowledgment

Gluon plasma is a product of teamwork and the protocol would not be possible without the extensive efforts of the full leverj.io team. In particular, N. Gupta for fixing many security issues and making the protocol robust with great attention to detail, B. Sadeh for simplifying deposits and ensuring the protocol remains simple, S. Tiwari for help with proofs and C. Engel for critical input from product fit and market perspective.

References

[1] S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf, 2009.

[2] Ethereum. A Next-Generation Smart Contract and Decentralized Application Platform. https://github.com/ethereum/wiki/wiki/White-Paper, 2013.

[3] G. Wood. Ethereum: A Secure Decentralized Generalized Transaction Ledger. https://gavwood.com/paper.pdf, 2014.

[4] J. Poon, V. Buterin. Plasma: Scalable Autonomous Smart Contracts http://plasma.io/plasma.pdf, 2017.

[5] N. Stifter, A Judmayer, P Schindler, A Zamyatin, E Weippl. Agreement with Satoshi – On the Formalization of Nakamoto Consensus https://eprint.iacr.org/2018/400.pdf, 2018.

[6] C. Dwork, M. Naor. Pricing via Processing or Combatting Junk Mail. http://www.hashcash.org/papers/pvp.pdf, 1993.

[7] parity.io https://wiki.parity.io/Proof-of-Authority-Chains.

[8] G. Maxwell. CoinJoin. https://bitcointalk.org/?topic=279249, 2013.

[9] Bitcoin.it https://en.bitcoin.it/wiki/Protocol_documentation.

[10] Bitcoin.it https://en.bitcoin.it/wiki/Cold_storage.

[11] A. Dmitrienko, C. Liebchen, C. Rossow, A. Sadeghi. Security Analysis of Mobile Two-Factor Authentication Schemes. http://www.icri-sc.org/fileadmin/user_upload/Group_TRUST/PubsPDF/Dmitrienko-127-camera-ready.pdf, 2014.

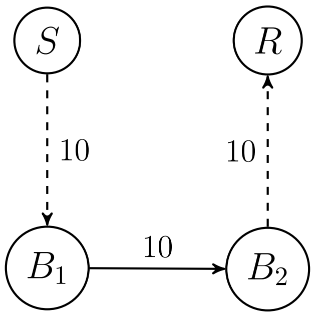
[12] B. Rao, N. Gupta. Leverj Decentralized Custody. https://leverj.io/LeverjProtocol.pdf, 2017.

[13] W. Warren, A. Bandeali. https://0xproject.com/pdfs/0x_white_paper.pdf, 2017.

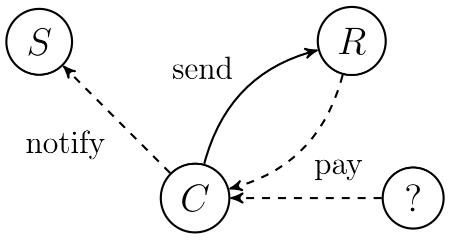
[14] investopedia.com https://www.investopedia.com/terms/o/oco.asp.

[15] N. Gupta. Zero Knowledge API Keys. https://blog.leverj.io/zero-knowledge-api-keys-43280cc93647, 2018.

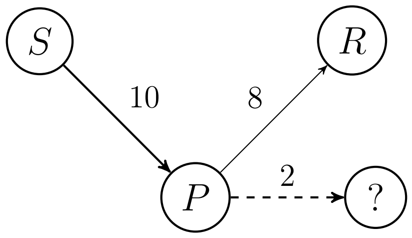
A Safety of Trustless Constraints



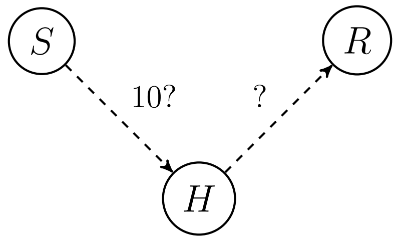
(a) Banking



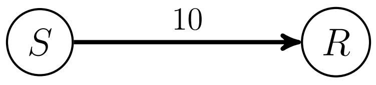
(b) Credit Cards



(c) Postal



(d) Hawala



(e) Crypto

Figure 7: Custodial characteristics of payment networks.

Banking. Payments are settled between member banks. The recipient waits for the bank to processes the transaction, deduct fees and release a portion for withdrawal. No *Segregation* implies risk of censorship and seizure.

Credit Cards. Anyone with the card information can initiate a payment. Elaborate fraud detection schemes are needed to compensate for the lack of *Agency* in the system design.

Postal Service. Some packages sent through the mail may be irreplacably lost. Even when using registered mail, there is no way to guarantee that all items sent will be received. The system lacks *Solvency* and is not usable for items of high value.

Hawala. The rest of the Hawala network cannot verify that a payment has indeed been correctly made. No *Integrity* enforcement inhibits the Hawala network growth beyond trusted family and kin.

Crypto. Crypto networks require a signature from the sender's private key. The payment is sent directly into the recipients public key. Network participants can observe the signature and validate the exact payment amount and update their ledgers. This network enforces all four constraints.

Theorem 1. *Provably safe payments between participants in a payment network is only possible when all intermediaries are perfectly safe.*

Proof. Let the digraph $G = (V, E)$ represent a payment network where V is the set of members and E is the set of interconnections. Let Π_s represent the set of all paths beginning with v_s. Let Π^t represent the set of all paths ending at v_t.

The set of all payment routes between v_s and v_t that v_s can use to send a payment to v_t is

$$\Pi_s^t = \Pi_s \cap \Pi^t \tag{38}$$

Let P_i be the probability of loss of safety during payment forwarding at each node v_i of any path Π_i in Π_s^t. The cumulative probability P of loss of safety at least a single node along Π_i is

$$P = \sum_0^n P_i \tag{39}$$

$$P = 0 \iff n = 0 \lor \forall i, P_i = 0 \tag{40}$$

Loss of safety is zero if and only if there are perfect or no intermediaries. □

Remark. *The outgoing edge from v_s represents* Agency*. The incoming path to v_t represents* Segregation*. Having only perfect or zero intermediaries represents* Solvency*. The ability of other participants to verify the existence of the edge (v_s, v_t) represents* Integrity*.*

Corollary 1. *A series of perfect intermediaries between two network participants is equivalent to having no intermediaries.*

Proof. A perfect intermediary is transitive.

$$v_s \rightarrow v_i \rightarrow v_t \implies v_s \rightarrow v_t \tag{41}$$

Perfect intermedairy nodes can be disregarded from analysis and a series of perfect intermediaries from vertex v_s to v_t is equivalent to a directed edge from v_s to v_t. □

Remark. *A path consisting of a single directed edge or equivalent from the sender to receiver represents the four trustless constraints.*

B Proof of Custody

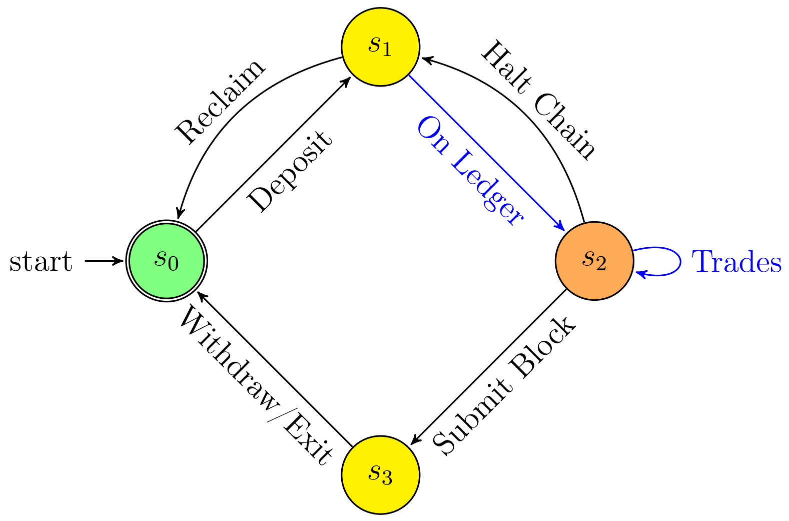


Figure 8: State Diagram of user funds custody

In Fig. 8, State s_0 represents funds on the main chain under direct control of the user's private key. States s_1 and s_3 represent funds on the plasma contract that the user can move back to s_0 without the need for any other parties. State s_2 represents uncommitted balance changes in the plasma chain. We prove continuous user custody by showing that the user can always move funds to s_0.

The user initiates a deposit into the plasma chain using the *Deposit* step of the protocol, moving funds from s_0 to s_1. State s_1 represents funds deposited by the user into the Plasma smart contract waiting to be credited into the plasma chain by the operator. Funds on state s_1 will move to state s_2 after k blocks. If not, the user can move funds back to s_0 using the *Reclaim Deposit* step.

State s_3 represents committed balances on a confirmed plasma block. Confirmed changes can be moved to s_0 using the *Withdrawal* or *Exit Asset Balance* protocol steps.

State s_2 represents funds on the actual plasma chain in unconfirmed blocks or the *Mempool*. During normal operation of the chain, balance changes are periodically committed to the plasma contract and the funds move to state s_3 from where they can be moved as above. When the operator is compromised, the chain can be halted and state s_2 is effectively deleted. User funds remain in either s_1 or s_3 and can be directly moved to s_0.

C Proof of Safety

The plasma system is safe only if every state change is safe. Enforcing the trustless constraints at every step of the protocol, every ledger entry and all ledger entry ordering creates a secure system. The tables below show all possible state changes and their enforcing fraud-proof or protocol step.

Table 5: ***Plasma Protocol.*** *Every protocol step enforces all constraints.*

Step	Segregation	Agency	Solvency	Integrity
Deposit	9.1.2	9.1.2	9.1.2	9.1.2
Reclaim Deposit	8.2	8.2	8.2	8.2
Withdraw	8.3	9.1.4	9.3.3	9.3.3
Exit	8.4	9.1.6	9.1.5	9.3.3
Submit Block	9.1.1, 9.1.3	9.1.1	9.1.1	9.1.1

Table 6: ***Plasma Ledger Entries.*** *Every ledger entry enforces all constraints*

Field	Deposit	Withdraw	Exit	Trade	Fee
Id	9.3.1	9.3.1	9.3.1	9.3.1	9.3.1
Prior	9.3.2	9.3.2	9.3.2	9.3.2	9.3.2
Account	9.1.2, 9.3.3	9.3.3	9.3.3	9.3.3	9.3.3
Asset	9.1.2, 9.3.3	9.3.3	9.3.3	9.3.3	9.2.2
Balance	9.3.3	9.3.3	9.3.3	9.3.3	9.3.3
Price	–	–	–	9.2.3	9.2.2
Qty	9.1.2, 9.3.3	9.3.3	9.3.3	9.3.3, 9.3.4	9.3.3, 9.3.4
Time	–	–	–	9.3.5	9.3.5
Witness	9.1.2	9.1.4	9.1.6	9.2.1	9.2.1
Segregation	9.1.2	8.3	8.4	9.2.1	9.2.2

Table 7: ***Order of Ledger Entries.*** *Combinatorial attacks prevented.*

Attack	Deposit	Withdraw	Exit	Trade	Fee
Unitary	–	–	–	9.3.6	9.3.6
Replay	9.1.1	8.3	9.1.5	9.3.4, 9.3.5	9.3.4, 9.3.5
Suppress	9.1.1	8.4	9.1.5	No effect[6]	No effect
Reorder	9.3.3	9.3.3	9.1.5	9.3.5	9.3.5

[6] Suppressed *Trade* entries are equivalent to cancelled orders and suppressed *Fee* entries are equivalent to free trades.