Stegos – a design for a private, confidential and scalable cryptocurrency

The Stegos Team, October 2018

***Abstract*— On October 31th of 2008, a person or group named 'Satoshi Nakamoto' released a paper called "Bitcoin: A Peer-to-Peer Electronic Cash System."[1] In ten years that passed since that date, Bitcoin became the main cryptocurrency in the world and inspired a lot of researchers and developers to create their own blockchains that are building and improving on Bitcoin and its ideas.**

One of the areas of improvement is privacy — despite the fact that payments in Bitcoin are conducted between pseudonymous users, the payment transactions are stored in a public decentralized ledger, from which a lot of information can be deduced. More than a dozen of *privacy coins* exists at the moment, each of which uses sophisticated algorithms in order to keep privacy of its users protected.

Despite that long list of privacy-centered blockchains, all of them have some substantial drawbacks. Some of the privacy coins protect identities of senders and recipients, but keep transaction amounts in open. Almost all of the existing privacy coins cannot be scaled, because their ledger must be ever-growing and cannot be compacted. Most part of these coins use Proof-of-Work consensus which is obviously unfriendly to the environment because it wastes enormous amounts of electricity.

In this paper we propose a design for a private, confidential, and scalable cryptocurrency that's friendly to the environment. This design builds and improves upon other privacy coins and can be used to send payments and data with complete confidentiality.

I. Introduction

In this section we give brief overview of most prominent privacy coins and analyze their privacy-protecting and performance characteristics.

We also explain, on a high-level, how Stegos builds and improves on existing privacy coins ideas and employs latest research in cryptography and blockchains in order to create privacy-preserving blockchain that can be scaled.

A. Terminology

Here we give definitions for miscellaneous terms that we are using throughout this paper:

- **Privacy**: protection from detection of the identity of blockchain participants and parties to transactions.
- **Confidentiality**: protection from analyzing blockchain data (e.g. transaction details, including amounts) by unauthorized third parties.
- **Unlinkability**: for any two outgoing transactions it is impossible to prove they were sent to the same person[2].
- **Untraceability**: for each incoming transaction all possible senders are equiprobable[2].
- **Compactness**: spent coins can be pruned from the blockchain and the blockchain can be compacted.
- **Sharding**: a transaction verification and block sealing processes can be partitioned between participants or groups of participants.
- **Interactivity**: senders and recipients of transactions must interact with each other *off-chain* before posting a transaction to the blockchain.
- **Trusted Setup**: blockchain participants need to trust someone to generate some initial parameters and then destroy those parameters.
- **PoW**: Proof-of-Work, an original consensus protocol for Bitcoin, where each node participating in the protocol should present a piece of data which is difficult (costly, time-consuming) to produce but easy for others to verify and which satisfies certain requirements. Due to complexity of computations needed and amount of miners, the current energy consumption by Bitcoin PoW consensus is approximately 73TWh per year[1].
- **PoS**: Proof-of-Stake **TODO**: [insert def.]
- **UTXO**: Unspent Transaction Output **TODO**: [insert def.]

B. Existing privacy coins

1) Monero: originated as a Bytecoin fork in 2014 under the name Bitmonero. The cryptocurrency is using UTXO model and PoW consensus, and based on the CryptoNote protocol[2], employing the ring signatures method. In 2017 Monero implemented RingCT[3], an improved version of the ring signatures. RingCT is enabling a confidentiality of amounts and untraceability of transactions. In combination with the stealth addresses (also introduced in the original CryptoNote paper), which provide unlinkability of recipients, this provides full privacy and confidentiality.

Monero's blockchain cannot be compacted because spent UTXOs cannot be pruned from it. Keeping all UTXOs forever is one of the requirements of RingCT protocol, which obfuscates the fact which particular UTXOs mentioned in transaction inputs were actually spent. Despite the recent introduction of Bulletproofs[4] which replaced Monero's original zero knowledge range proofs and got a simple transaction size from 13KB down to 2.5KB, the problem with ever-growing blockchain cannot be solved.

2) ZCash: originated in 2016 as Bitcoin fork and obviously is using UTXO model and PoW consensus. The goal of the project is to improve the flaws of Bitcoin, with a focus on privacy. The project is building on a work done on Zerocoin[5], addressing some of its faults, like the size of the proofs, which ZCash decreases to 1KB and speeds up the verification.

[1] https://digiconomist.net/bitcoin-energy-consumption

For establishing confidentiality and untraceability, ZCash implements "Zero-Knowledge Succinct Non-Interactive Argument of Knowledge" (zk-SNARKS)[6]. For providing unlinkability of recipients, ZCash employs the stealth addresses.

TODO: [explain what anonymity set and ever-growing accumulator are] zk-SNARKS enable a large anonymity set of all minted coins, providing a high degree of privacy. However, the size of up to 2 kB for average transaction and ever-growing accumulator that cannot be pruned, makes ZCash much less scalable. The issue of scalability is the main reason why the privacy is currently optional, not by default. At the time of writing, less than 25% of all transactions were shielded.

The questionable part of zk-SNARKS protocol is the initial trusted setup. ZCash utilized a multi-party ceremony involving a several trusted people. This is controversial, as you have to trust any of these people that they destroyed the initial parameters and also trust that the ceremony was carried out correctly.

3) Dash: originally a codebase fork from Litecoin (which is in turn a codebase fork of Bitcoin), Dash were launched as XCoin in January 2014. Dash is using UTXO model and PoW consensus.

Besides standard nodes and miners, Dash has *masternodes*, the nodes that must have static IP address and fullfil particular requirements for CPU, RAM, and disk space. Each masternode must have at least 1000DASH in its ownership. A Proof-of-Service protocol ensures that masternodes have the most current blockchain protocol and are online.

It is important to note that privacy is optional in Dash and the high volume of transactions is driven mainly by the fast public transactions called *InstantSend* which are provided by masternodes.

PrivateSend is an implementation of CoinJoin, the untraceability solution first proposed for Bitcoin by Bitcoin Core developer Gregory Maxwell[2]. In PrivateSend, three users add their coins together in one big transaction, that sends the coins to freshly generated addresses belonging to the same three users. As such, the coins are effectively mixed between the three participants, breaking the blockchain trail of ownership between them. This process can be automatically repeated up to eight times, with (hopefully) different mixing participants, for extra privacy.

Dash is not providing confidentiality of amounts neither in InstantSend, nor in PrivateSend. Moreover, it is essential requirement of CoinJoin protocol that inputs of users involved in the round of the mixing protocol must have exactly the same denomination (amount of coins). This requirement is impossible to fullfil in case if amounts are cloaked.

Users of Dash have to trust masternodes that they will keep users' IP addresses not disclosed and not linked to users' UTXOs when sending transactions in Dash. This adds another privacy vulnerability to Dash.

4) Mimblewimble (Grin): is a protocol that was proposed by an anonymous user in a Bitcoin developers chatroom by the name of Tom Elvis Jedusor, that left a link to a paper[3] in which he outlines that by using the Mimblewimble protocol, the scalability, as well as the privacy of the Bitcoin network could significantly be enhanced.

Mimblewimble is based on the ideas of Greg Maxwell's design for Confidential Transactions[4], except, it is the recipient that generates random blinding factor used to cloak the amount of the transaction. This blinding factor is then used as proof of ownership by the recipient, therefore working simultaneously as recipient's public key. Therefore Mimblewimble provides confidentiality of amounts and unlinkability of recipients in its transactions.

Untraceability of transactions in Mimblewimble is building on the ideas of CoinJoin and is implemented by breaking transaction boundaries and storing only inputs and outputs of all transactions verified by miner in a new mined block.

The only existing implementation of the protocol known to the moment is Grin project[5], which has recently released the third iteration of their testnet. Grin uses UTXO model and PoW consensus. Spent UTXOs in Grin can be pruned, by recursively applying simple pruning algorithm for each UTXO referenced in inputs of the new minted block.

There are several drawbacks in the design of Mimblewimble and Grin:

- First is related to the transaction creation mechanism of Mimblewible and Grin — in order to create a transaction, a sender and recipient must interact with each other. The sender cannot post a transaction to the blockchain without first contacting the recipient with half-baked transaction data and receiving that data updated by applying a blinding factor back from the recipient.
- Second is very similar to the problem that Dash where users should trust Dash masternodes. Currently, users of Grin should trust Grin miners that miners will not trace the history of moving coins inputs to outpus in transactions, and discard this data completely after mining a block. Therefore there is a potential threat to coin's fungibility and privacy of the users.

C. Our contribution

In this paper we propose a design for *Stegos*, a blockchain which uses the *UTXO* (coin) model and *PoS* (Proof-of-Stake) consensus.

Transactions in Stegos are *unlinkable*, *untraceable*, and *confidential*:

- Stegos uses one-time payment addresses which make it impossible to identify recipients of a transaction (in another words, to link addresses of recipients to their identities) because all transactions are directed to new and unique addresses. The technique used for one-time

[2] https://bitcointalk.org/index.php?topic=279249.0

[3] https://download.wpsoftware.net/bitcoin/wizardry/mimblewimble.txt

[4] https://people.xiph.org/~greg/confidential_values.txt

[5] https://github.com/mimblewimble/grin

addresses is very similar to stealth addresses used in Monero and ZCash.
- Stegos makes it impossible to trace history of transactions since many individual transactions are pooled together to form a super-transaction. For this purpose we adopted and enhanced ValueShuffle protocol[7], the first coin mixing protocol compatible with Confidential Transactions.
- All amounts in Stegos are hidden using Pedersen commitments[8] and Bulletproofs range proofs[4]. Validator stakes and transaction fees are the only exception since these must be visible for blockchain validation.

TODO: [write why compaction is important]

Stegos is a *compact* blockchain. Spent coins are safely removed from the blockchain using secure cryptographic pruning. To keep our blockchain free of spent coins we are using the technique proposed originally by Satoshi Nakamoto in Bitcoin paper[1].

Stegos uses transactional *sharding* to scale. Separate groups of Stegos validators keep the whole blockchain state but verify only a subset of incoming transactions, using cross-shard atomic commits to eliminate double-spending. This scalability approach lets Stegos process hundreds of thousands of transactions per second.

Stegos is friendly to the environment and does not require terawatts of electricity to be spent for mining blocks. Stegos is using PoS (Proof-of-Stake) consensus, building on ideas of pBFT[9] and Collective Signing[10][11]. Each new Stegos block must be verified and confirmed by a group of validators, all of which must put coins in escrow (stake).

The size of the coins staked has a direct effect on the probability of a validator to become a leader of the consensus and earn transaction fees. Stegos does not have block rewards but replaces them with the *Jackpot*.

This is a feature unique to Stegos and a lottery concept that everyone is familiar with. A portion of the fees from each block are added to the Jackpot and any stake forfeited by a validator caught cheating goes into the Jackpot as well.

The Jackpot is distributed every few thousand blocks when validators run a cryptographic lottery based on verifiable distributed randomness. The amount in the Jackpot is then transferred to the winner. The longer a validator keeps its stake and participates in consensus, the higher the probability of winning the Jackpot lottery.

Electing a leader of the consensus and choosing a winner in the Jackpot lottery use our algorithm for generating distributed, untainted, unbiasable randomness, in the spirit of the RandHound protocol[12]. Our algorithm, *RandHound++*, actually derives much of its protocol from SCRAPE[13] as its primary deviation from original RandHound. SCRAPE defines a PVSS[14] protocol using symmetric bilinear pairing based cryptography and coding theory. But beyond SCRAPE, we had to adjust the algorithm to deal with asymmetric pairings which offer necessary stronger cryptographic security.

In the following sections we will explain each of the Stegos features presented above in more details. **TODO**: [add descriptions of other sections when they will be done]

II. Networking

The Stegos network is composed of full nodes and light nodes, e.g. those running on mobile devices or in the browser. Each full node maintains a full copy of the blockchain and associated data structures. Those full nodes which put coins in the escrow, become *validators* and may participate in the collective signing protocol (*CoSi*) and respond to requests of the light nodes.

Light nodes do not maintain the full blockchain, they only keep blockchain headers and know how to talk to validators.

Stegos maintains the core of the network by running a number of validator nodes, which both simplifies the bootstrap of new nodes and ensures continuity of the network. The addresses of the core nodes are hard-coded into each release of the Stegos blockchain software.

Nodes keep list of addresses of the nodes they know about (peers) and add new peers to the list as they become aware of them. A new node starting will connect to one of the core nodes to fetch a list of peers. Each full node quickly re-broadcasts received transactions to a set of its peers, after only a few light checks. This both ensures that a node cannot be DDOS-ed with transactions to validate and that it does not re-broadcast junk transactions.

To discourage bad actors, we employ a mechanism to both throttle peers and punish peers for bad transactions, etc. by blocking them from further participation in the network.

The Stegos blockchain uses gossip protocols to spread information without depending on fixed networks of communicating nodes. This protocol does not require every node to be reliable or always up and running and does not require every node to know about every other node. The key to a gossip protocol is that every node knows only a few other nodes (or even just one) in the network, but as long as most nodes know about at least two other nodes besides themselves, information can propagate through the network.

III. Randomness

Bias-resistant distributed randomness is a critical component of the Stegos. We employ *RandHound++*, our own, improved variant of RandHound[12], a large-scale distributed protocol that provides publicly-verifiable, unpredictable and unbiased randomness by implementing an efficient and decentralized randomness beacon. The protocol arranges participants into a verifiably unbiased random secret-sharing groups, which repeatedly produce random output at predefined intervals.

We use RandHound++ for multiple purposes:

- to form validators groups;
- to elect leaders of validators groups;
- to choose transactions pooling facilitators among validators;
- to find a winner of the Jackpot lottery.

A. Distributed Randomness Generation

The idea of performing distributed randomness generation is to have many nodes participate in the construction of an untainted and unpredictable random value in order to overcome Byzantine faults which might try to steer, or predict, the outcome.

RandHound++ is based on the notion of publicly verifiable secret sharing (PVSS)[15], which ensures that every step of the process can be audited using zero knowledge proofs (ZKP), while keeping vital details of the process fully cloaked through strong cryptography.

But unlike distributed Consensus Protocols, like pBFT, we don't need full Byzantine threshold participation at every stage. In what follows we assume a maximum number, f, of faulty nodes, in a collection of $(3f + 1)$ participant nodes.

The usual Byzantine Fault Tolerance (BFT) requires responses from a super-majority of $(2f + 1)$ nodes in order to reach consensus. But in our application, generating randomness, as long as at least one honest participant blends its randomness into the computation, the result will be purely random. We do not need consensus on the result as long as its process can be verified.

Hence we only need $(f+1)$ responses from participants to overcome any Byzantine faults, where at least one participant will be honest, and supplies true randomness. No need for consensus on the actual value generated. It is sufficient to see only that at least one node was honest. That is clearly true with $(f+1)$ participants. Attackers can perturb the final value, but it will still become an honest random value by virtue of the inoculation property.

B. *RandHound++*

Our variant of RandHound protocol is implemented with pairing-based cryptography (PBC). As such it departs in a number of significant ways from the original RandHound algorithm. It is a distributed protocol for generating auditable untainted and unpredictable randomness, as required for fair stake-weighted elections of the validators group leader, among the other purposes.

After every election, a validators group's *leader* is selected, and then a *beacon* node is chosen among the remaining validators. The *beacon* node is responsible for partitioning all validators (including itself and the *leader* into RandHound++ groups.

The first validator in any group becomes a local group leader who collects randomness from all its group members and itself, and then forwards that result back to the *beacon*. The *beacon* then combines all the group randomness values into into a single result that becomes the seed value for the next elections, then the *beacon* sends a *HOLD-ELECTION* message, with that seed value, to all validators in the network.

A major advantage of PBC is that we can use efficient BLS signatures[16] in the protocol. Such signatures can be collectively added, along with the group sum of their public keys, to produce a short collective-signature that can be validated just as easily as a single signature. BLS signatures are efficient because multi-signatures can be generated immediately and do not require a second query from participants in order to produce their collective signature.

But PBC entails some complications in translating the RandHound algorithm to asymmetric pairs of elliptic curves. Though these complications are purely conceptual — the mathematics are more difficult. But PBC performs nicely, with the same efficiencies for producing PVSS data as are to be had with producing BLS signatures.

We are going to publish a separate paper, describing in more details our improvements on the original RandHound protocol in the nearest future.

IV. Consensus

The Stegos consensus protocol is based on pBFT[9] but adds strong consistency, which enables all validators to agree on the validity of blocks without wasting computational power resolving inconsistencies. Clients don't need to wait more than a few seconds to be certain that a submitted transaction is committed; as soon as it appears in the blockchain, the transaction can be considered confirmed.

A. *Collective Signing*

We adopt *CoSi*[10][11], a scalable witness cosigning protocol ensuring that every authoritative statement is validated and publicly logged by a diverse group of witnesses before any client will accept it. A statement S collectively signed by W witnesses assures clients that S has been seen, and not immediately found erroneous, by those W observers. Even if S is compromised in a fashion not readily detectable by the witnesses, CoSi still guarantees S's exposure to public scrutiny, forcing secrecy-minded attackers to risk that the compromise will soon be detected by one of the W witnesses.

CoSi builds on existing cryptographic multisignature methods, scaling them to support thousands of witnesses via signature aggregation over efficient communications. The default implementation of CoSi uses Schnorr signatures which we replace with BLS signatures for performance reasons.

More details on using CoSi protocol for sealing blocks in Stegos blockchain will be given later in this document, it the Section VII.

B. *Leader Elections*

Stegos is a public decentralized blockchain so anyone can join the network, become a validator and earn rewards for maintaining it. Stegos uses Proof-of-Stake (PoS) consensus and all validators need to post a security bond of a fixed number of coins. Requiring a security bond helps prevent against Sybil attack by requiring a significant financial commitment from each identity.

Any validator caught cheating by their peers loses their stake, which gets transferred into the Jackpot.

Validator elected to become a leader of the current epoch is rewarded by collecting transaction fees from the blocks it forms.

Leader elections are stake-weighted lottery. The larger the validator stake (security bond), the greater the chances of winning the current round of the lottery. After serving one round, the leader relinquishes his right to be re-elected until some number of rounds later.

In order to produce a stake-weighted lottery, we could imagine dividing a circle into angular sections proportional to individual stakes, with the full circumference of the circle representing the sum of all stakes. A uniformly generated random number over a finite interval (this is where our RandHound++ protocol becomes handy) acts as a spinner in a dial readout, and wherever it points after a spin, that member is selected as the next leader. Larger stakeholders occupy a larger portion of the circumference, and so have a more likely chance of winning.

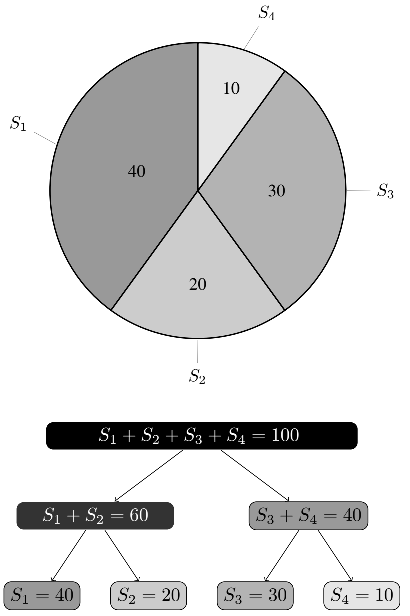

Fig. 1. Imagining a spin dial as a tree. Segments are sized in proportion to the posted security bond (stake).

In practice, we form a binary tree of participants, in a consistent order, where each interior node represents the sum of all its sub-node stakes. The topmost node of the tree shows the full sum of all stakes represented in the tree, with all participants located in leafs. Branching decisions begin at the top node of the tree and descend toward leafs using the random value as a probe of the interval described by the partial sum at each node, with the division between left and right based on the relative weights of its two subtrees. Descent continues until meeting a leaf node, and declaring that node the winner of the election.

If the random probe value is converted into a fractional value of its range, and each interior node is relabeled with its division fractional value, then it is simple to choose left or right subnodes based on a single comparison between the two fractional values.

That this works can be seen by noting that every interior node of the tree serves both as denominator to nodes below, and as numerator to connections above, hence cancelling out. The final probability for any participant winning becomes the same as their escrow stake divided by the sum of all participant escrow stakes.

1) Handling leader failures: Leader elections are held on a schedule and validators recognize the signal to hold a new election.

The list of validators is publicly known and can be rebuilt by processing staking transactions starting with the genesis block. Validators keep a number of data structures in memory, including the list of other validators, and update these data structures when accepting blocks to add to the the end of the chain.

Each validator can re-run the election for themselves upon seeing the random value of the election signal, which arrives via RandHound++ protocol. All nodes should be able to agree on election outcome, but if not, then one of two things could happen:

- First, if a validator thinks it won the election but was mistaken, then any attempts at forming new Cosi networks by the faux leader will be silently rejected by others, and it will fail to obtain a consensus on any new requests. If that happens, the rejected leader will resynchronize its list of validators, rerun its own election, and re-join after agreeing on the outcome.
- Second, a validator may think that someone else had won the election. In that case, it will silently reject signing requests by actual election winner, and never see any requests from the presumed one. The validators will re-sync its list of other validators if this happens for two consecutive election rounds.

If the election winner is absent or comes under attack, it may not respond to its own election as leader, form the block or broadcast the block for signing. Any validator can call for a new early election, and after $(2f + 1)$ (supermajority) of such calls from different validators have been seen, a new election round will start.

2) A fault-tolerant and self-healing protocol: Immediately after a leader election, each validator should determine its role for the new epoch:

- If elected leader, then begin next block assembly.
- If the node should become a member of a RandHound++ group, then start running the protocol within its group.
- If the node should become a facilitator for transaction pooling protocol, then start collecting transaction intent messages.
- If the node should become a CoSi witness node, then start a timeout timer. If that timeout ever fires, then it is time to call for a new election. Incoming messages can restart or cancel the timeout timer, though.
- Otherwise, the node is on standby for the next election cycle. Nodes on standby can ignore almost all messages, but must pay attention to election related messages.

All validators should pay attention to leader election messages:

- *New Election Message*: run an election with the random value provided in the election message, and determine next role for this validator — one of leader, witness signer, RandHound++ participant, facilitator or standby. Reset the counter of calls for new election.

- *Call For New Election Message*: Increment the count, n, of such calls seen from different validators, and if $n \geq 2f+1$, then run another election round based on the last election's random value, with a decision tree that excludes current leader. Determine new leader and next role for this validator. Reset the counter of calls for new election.

V. Basic Transactions

In previous sections, we explained the high-level concepts of Stegos, including networking, randomness and consensus. Now it's time to describe the inner details of Stegos, such as the composition of its coins and basic transactions.

A. UTXO Structure

For illustrative purposes, suppose that some coins have been sent from Alice to Bob. Alice's public key is P_A, while Bob's secret key is s_B and his public key is P_B. To help preserve anonymity, all public keys in Stegos are cloaked with a random value chosen from a very large finite field, Z_r.

When Alice sends coins to Bob, she cloaks their amount with a Pedersen commitment, which is both binding and hiding. It binds Alice to her commitment so that she can never alter the amount of coins. The commitment also hides the amount from the general public, while simultaneously providing proof to the public that the amount is legitimate. Only Alice and Bob know how many coins are being transferred.

To form a Pedersen commitment, Alice multiplies the amount by a generator, A for the Elliptic curve group, E_q, of prime order r. To that she cloaks the commitment by adding a multiple, γ, of the publicly known generator point, G, with the multiple being chosen randomly from the finite field, Z_r. Generators A and G must have no known relationship. Placing the cloaking factor on the main generator curve will become important as we proceed. This is the same curve which holds all public keys.

The Pedersen commitment is thus:

$$C(x,\gamma) = x\,A + \gamma\,G \in E_q$$

$$x, \gamma \in Z_r$$

where x denotes the number of coins being transferred, A is the amount-curve generator, and G is the principal generator. We denote the commitment by $C(x,\gamma)$.

This commitment value will be wrapped inside a Bulletproof range proof on the amount, x, which also proves that the amount lies within a legitimate range of values, typically a 64-bit number.

Alice then cloaks Bob's public key with factor, $\delta \in Z_r$, and leaves a record of the amount x, and cloaking factors, γ and δ, in an encrypted payload as a part of the *UTXO* awaiting Bob. She forms a UTXO *ID* by hashing the rest of the UTXO record, which contains the cloaked version of Bob's public key, $P_{B,\delta}$, the Bulletproof, and the encrypted payload.

The UTXO *ID* becomes a unique identifier, since, if all else were equal, the γ and δ factors were randomly chosen from field Z_r.

$$UTXO = (ID, P_{B,\delta}, Bp, E_B(x,\gamma,\delta))$$

where

$$\begin{aligned} ID &= H_r(P_{B,\delta}, Bp, E_B(x,\gamma,\delta)) \in Z_r \\ H_r(arg_1, arg_2, ...) &= \textit{hash mapping of concat args} \rightarrow Z_r \\ P_{B,\delta} &= P_B + \delta\,G \\ G &= \textit{known generator for group } E_q \\ Bp &= \textit{Bulletproof of range on amount}, x \\ E_B(x,\gamma,\delta) &= \textit{Encrypted payload} \end{aligned}$$

$E_B(x,\gamma,\delta)$ represents an encrypted packet containing the information about x, γ, and δ, in a form that only Bob can read.

The Bulletproof contains the Pedersen commitment discussed above, as well as providing a range proof on the amount, x. Everyone knows the E_q group generators, G and A. Alice's complete transaction record contains additional information that we'll discuss below.

Nowhere is either of Alice's or Bob's public key shown. We only present a cloaked version of Bob's key. And since δ is a secret value, nobody can recover the actual public key underlying the cloaked version.

Therefore Bob may publish his public key openly, i.e. on his website or in an invoice, and don't worry that his identity will be linked to the recipient of the payment on Stegos blockchain, because his public key will appear in Stegos UTXOs always in a cloaked from – every time as a completely new and random number.

B. Transaction Structure

When Bob wants to spend his new tokens, he must form a transaction containing a list of inputs (TXINs) and outputs (TXOUTs). TXINs are nothing more than the *ID*s referring to other UTXOs. TXOUTs are a list of new UTXOs. He must also offer a valid signature on the entire transaction, which simultaneously proves his ownership of all TXIN's, proves that the transaction carries zero net balance of funds between TXINs, TXOUTs and fees, and protects the contents of his transaction against mutation by MITM attackers.

A UTXO can only be spent in its entirety, and if it carries excess value for his purposes, he will produce a TXOUT with change back to himself, thereby creating a new UTXO. Bob must show that the sum of all inputs to his transaction equals the sum of all outputs plus fee. And he can do so with Pedersen commitments so that his transaction is binding on him, while also disclosing nothing about the actual amounts involved.

To form his signature, he adds together all the δ cloaking factors from the UTXOs specified by his TXINs list of *ID*s, adds in all the γ cloaking factors from the Pedersen commitments in the Bulletproofs from those same UTXOs, and subtracts the γ cloaking factors used in his own TXOUT UTXOs.

Suppose Bob uses N TXINs. His own public key, $P_M = s_M\, G$. Then his effective secret key for the signature becomes:

$$s_{\mathit{eff}} = N\, s_M + \sum_{i \in \text{ins}} \delta_i + \sum_{i \in \text{ins}} \gamma_i - \sum_{j \in \text{outs}} \gamma_j$$

Using this effective secret key, he produces a Schnorr signature pair, (u, K), after choosing $k \in Z_r$ at random:

$$K = k\, G$$

$$u = k + H_r(T, K)\, s_{\mathit{eff}}$$

$$Sig(M, T) = (u, K)$$

so that validators can see that:

$$u\, G = K + H_r(T, K)\, P_{\mathit{eff}}$$

$$P_{\mathit{eff}} = \sum_{i \in \text{ins}} P_i + \sum_{i \in \text{ins}} C_i - \sum_{j \in \text{outs}} C_j - \mathit{Fee}\, A$$

where T represents the entire transaction record, sans signature.

Let's examine the commitment terms. Pedersen commitments are additively homomorphic:

$$C(x_1, \gamma_1) + C(x_2, \gamma_2) = C(x_1 + x_2, \gamma_1 + \gamma_2)$$

Hence, if Bob's transaction is valid, the amount terms in the effective public key show a zero balance on the A curve, after subtracting off the Fee. What remains is entirely on the G curve. The validator sum becomes another public key on the G curve, exactly matching her own computed effective secret key, s_{eff}. Only Bob could form a valid signature since it relies on his secret key. It is unforgeable. Both Alice and Bob know all the other secret items, γ's and δ's. Nobody else knows any of the secret values.

Suppose Bob now just wants to send the coins she received from Alice on to Charlie, after deducting fees. In that case, Bob forms a UTXO that uses a different blinding factor, γ_2, and different key cloaking value, δ_2, and which contains amount, $(x - \mathit{Fee})$. Bob must form a new Bulletproof on the amount, and encrypt these values into a payload that only Charlie can read:

$$ID' = H_r(P_{S,\delta_2}, Bp', E_S(x - Fee), \gamma_2, \delta_2))$$

where P_{C,δ_2} is Charile's cloaked public key for this transaction.

Charlie can spend this UTXO by providing a valid transaction signature against the UTXO *ID'*, just like Bob did for his own input.

Bob's TXOUT will now look like this:

$$\text{TXOUT} = (ID', P_{S,\delta_2}, Bp', E_S(x - Fee, \gamma_2, \delta_2))$$

To wrap up, Bob publishes the transaction:

$$\begin{aligned}
\text{T} = \{&\text{TXIN} : \{ID\}, \\
&\text{TXOUT} : \{(ID', P_{S,\delta_2}, Bp', E_S(x - \mathit{Fee}, \gamma_2, \delta_2))\}, \\
&\text{FEE} : \mathit{Fee}, \\
&\text{GAMMA} : \gamma_{adj} = \sum_{i \in \text{ins}} \gamma_i - \sum_{j \in \text{outs}} \gamma_j \\
&\text{SIG} : Sig(M, T)\}
\end{aligned}$$

The first line is Bob's TXIN referencing the UTXO produced for him by Alice. The second line is the TXOUT - a new UTXO aimed at Charlie. The third line shows the fees paid for this transaction, in clear text form.

The fourth line shows what value of γ adjustment, on the G curve, is needed to see that the sum of input commitments equals the sum of output commitments, proving a zero net balance between TXINs and TXOUTs and Fee. This information is redundant in the sense that a valid signature on the transaction already proves this. But future coalescing of transactions, e.g. in the blockchain, needs to know the individual γ_{adj} factors in order to produce a valid collection from individual transactions.

The final line is Bob's signature asserting ownership of all TXINs, valid zero balance on the transaction, and serves as a checksum against mutation of the contents of this transaction. If anything becomes changed in this record, the signature won't check. Hence our transactions are non-malleable.

VI. Pooled Transactions

A. A problem of establishing untraceability

We just saw that, even though participants' identities are cloaked in the blockchain, a record of ancestry for every UTXO could be potentially generated by tracing blockchain transactions backwards up to the genesis block if neeeded.

Despite we have a serious hurdle for such tracing — we don't store transactions in the blocks of out blockchain, but rather Merkle trees of inputs and outputs — the malicious node implanted in our permissionless blockchain may become a validator very soon in the blockchain history and it may collect and store transactions history, in order to analyze and trace UTXOs later on.

To counteract that, we want to utilize a protocol that totally hides knowledge about the relationship of between inputs and outpus.

The difficulty in implementation of such protocol follows from the fact that UTXOs must have identifiable characteristics. Their IDs are unique. Without that, no witness could ever validate transactions, and there would be no way to prove ownership. But that ID leads to an inevitable trail. We want to obscure that trail by showing only that a UTXO could have come from one or more of many different and unrelated inputs. We want to ensure that nobody can tell what its source was but done so in a manner allowing public validation.

B. *Possible solutions*

Currently there are four kinds of approaches for solving the problem of establishing untraceability: mixers, Ring Signature, zk-SNARK and CoinJoin family of protocols.

1) Mixers: require that users trust the third-party which provides mixing services, which is obviously an unacceptable scenario for a truly private and confdential blockchain.

2) Ring Signature: collect together a large number of UTXOs from the blockchain at random, and add those into the list of actual inputs being spent. A signature is formed on all of the inputs, so that you can prove that one group of inputs are being spent properly, but nobody can determine which ones. The problem with this approach is that you can't ever know, therefore, if a UTXO becomes spent and could be pruned from the blockchain. All UTXOs that *ever existed* must be retained in the blockchain. That doesn't scale very well.

3) zk-SNARK: stands for "Zero-Knowledge Succinct Non-Interactive Argument of Knowledge." Currently, the only known way to produce zero-knowledge proofs that are non-interactive and short enough to publish to a block chain is to have an initial setup phase that generates a common reference string shared between prover and verifier. If someone had access to the secret randomness used to generate these parameters, they would be able to create false proofs that would look valid to the verifier. For a cryptocurrency that uses zk-SNARK, like, for example, Zcash, this would mean the malicious party could create counterfeit coins.

Moreover, in order to defend zk-SNARKs-based cryptocurrency from the double-spending, a cryptographical accumulator should be etsablished on all nodes. It should contain the serial numbers of all spent coins. This accumulator is ever-growing and cannot be trimmed, which hurts the sustainability of the blockchain.

4) CoinJoin: is a protocol for joining several Bitcoin transactions together before submitting them to miners to include in the block, that was originally proposed by Greg Maxwell in 2013. It is based on the following idea: "When you want to make a payment, find someone else who also wants to make a payment and make a joint payment together."[6] CoinJoin implementations were based on the usage of trusted server, that provided service of mixing multiple transactions together.

CoinJoin inspired researchers from Saarland University to develop a completely decentralized protocol they called CoinShuffle[17] in 2014, that also allows users to mix their coins with those of other interested users. It ensures security against theft and by the accountable anonymous group communication protocol Dissent to ensure anonymity as well as robustness against DoS attacks. Continuing their research, the same researchers in 2016 presented echanced version of the protocol, which they called CoinShuffle++[18]. The key innovation of CoinShuffle++ was the use of a more efficient mechanism for anonymity: it uses Dining Cryptographers Networks (DC-nets)[20] instead of mix-nets. A mix-net requires sequential processing. Thus, the original CoinShuffle needs a number of communication rounds linear in the number of users, even if everybody is honest. CoinShuffle++ instead uses DC-nets and thus allows users to process mixing in parallel. As a result, CoinShuffle++ requires only a constant number of communication rounds independently of the number of users in this case.

ValueShuffle[19] is an extension of CoinShuffle++ which is compatible with confidential transactions.

TODO: [why we've chosen ValueShuffle and what is our contribution]

TODO: [have to edit heavily all sections below]

C. *Our Pooling Protocol*

A wallet signals its desire to join a pooling group by advertising a fresh ephemeral public key, along with signature proof of validity, on a bulletin board. Once a group forms, the key will become part of an advertised key collection. This key collection is then sorted in ascending numerical order, the hash of the collection is formed, and that hash becomes a random seed for the session.

A leader is selected by forming the XOR of the hash value with their public key, and if the resulting value is the minimum in the list, that node has been chosen as leader. Leader is responsible for publication of final super-transaction. All other communications are P2P between members of the group.

All nodes in the group will broadcast their list of TXIN's along with signatures validating ownership of the TXIN's. Every participant can validate the lists from other participants by verifying the signature attesting to TXIN ownership. Participants should check that the public key in the signature is from one of the group members.

A valid signature on the TXIN's from one participant is formed by sending a Schnorr signature on the sum of the cloaked public keys shown in the referenced UTXO's in the blockchain:

$$\begin{aligned} TXIN &= \{ID_1, ID_2, ..., ID_N\} \\ Sig &= (u, K) \\ K &= k\,G \\ u &= k + H(K|ID_1|ID_2|...)\,(N\,s + \sum_i \delta_i) \end{aligned}$$

where the sum is over the list of TXIN's, s is the owner's secret key, and the δ_i were the cloaking factors used on the public key of the owner. N refers to how many TXIN there are in the list. Value k is chosen randomly from the field, Z_r.

Verification of a signature is by seeing that:

$$u\,G = K + H(K|ID_1|ID_2|...) \sum_i P_i$$

where the P_i are the cloaked public keys shown in the UTXO's corresponding to each ID_i. This signature proves ownership of the stated TXIN UTXO's.

[6] https://en.wikipedia.org/wiki/CoinJoin

Nothing in this broadcast identifies the sender, but one cannot count on that to hold. In the event of misbehavior, a blame cycle will require each node to submit all their shared secret keys, which effectively reveals their full transactions. A restart will compute new TXOUT so that a successful run ensures anonymity of participants. But if a blame cycle is performed there is no way to cloak the associations to TXIN's again.

Once contributions have been received from each participant, or a timeout occurs, the resulting pool of TXIN is known to all participants. Since these are merely UTXO ID's pointing into the immutable blockchain, no further changes can occur to this list except for removal of individual TXIN references as some participants are found to go offline during the protocol, or else when a cheater is detected and then excluded for a restart of the protocol.

All participants will establish pair-wise shared secret keys between themselves and every other participant in the P2P network. These shared keys are used to form cloaking factors in the anonymizing protocol, such that, only after pooling results from all participants, will the collection of data shared among them become apparent. Until that time, all information remains cloaked. And after that time, the data will be known, but no associations can be inferred regarding who supplied which portions of the data.

It is important to the protocol, that between any two users, both of them are using the same shared secret cloaking key. Only then will the sum of all cloaking factors, from all participants, cancel out in the DiceMix arrays. But users are prevented from seeing each other's secrets since the total cloaking factor also sums contributions related to all other participants, and those keys are unknown to the other party.

Shared keys can be securely established between every pair of participants using Diffie-Hellman secure key exchange. With two participants, A and B, this can be established by having A send to B

$$A \to B : (\alpha\, P_B, Sig(P_A))$$

for α randomly chosen by A, and where $Sig(P_A)$ securely authenticates this information as having come from A. The signature includes the public key, P_A.

Then B responds to A with

$$B \to A : (\beta\, P_A, Sig(P_B))$$

with β being chosen randomly by B.

After the exchange, the shared key is a hash of the product:

$$key = H(\alpha\beta\, G)$$

But since neither side knows both factors, at A we compute:

$$\beta\, G = (\beta\, P_A)/s_A$$

since $P_A = s_A\, G$. And at B we compute

$$\alpha\, G = (\alpha\, P_B)/s_B$$

Then each side can multiply their result by their chosen randomness to reveal $(\alpha\beta\, G)$. Nobody watching the exchange can deduce the shared secret key.

Next, each participant computes their TXOUT records with fresh randomness, chooses random k-factors for an eventual collective Schnorr signature, and produces both a DiceMix array containing the fragmented TXOUT records, and cloaked running sums of their γ_{adj} and K-signature values. The hash commitment of this information is signed and broadcast to all participants. This commitment will be used to verify information during the following passes to verify that information was properly transmitted.

The DiceMix array contains successive powers of their TXOUT record fragments, cloaked with self-canceling seeds that zero out after all participants pool their results from the next pass. After forming the DiceMix array and running sums, this information is signed and broadcast to all participants. Anonymity is assured because of the DiceMix cryptographic mixing process. Even though all participants can see, from the accompanying signature, who delivered a DiceMix array, they cannot see which components of the collection were contributed by any participant. The entire collection is revealed only after summing the DiceMix arrays from all participants.

The individual K-signature terms from each participant are summed in a blind sum. We do this to prevent combinatorial exploration, once signature u-values are disclosed, which could lead to associations between TXIN and TXOUT.

On receipt of all DiceMix arrays and the running sums, each participant can form the polynomial, using Newton's Identities, whose roots are the individual contributions. Solving that polynomial for its roots reveals each component of participant TXOUT's. These TXOUT's are reassembled, and a super-transaction is formed containing all TXIN's, TXOUT's, the γ_{adj} sum needed to show zero balance, and the K-signature sum needed for a collective Schnorr signature.

Each participant validates the entire transaction for correctness by examining the γ sums for zero balance, and verifying the TXOUT Bulletproofs. They must also find their own contributions in the list of TXIN and TXOUT. If the super-transaction does not properly validate, then someone has cheated, and we enter a blame discovery cycle. Otherwise we proceed to the formation of the collective signature.

Knowing the super-transaction, and the collective K_{sum} signature term, each participant broadcasts their u-signature component to be summed with those of other participants, yielding a collective Schnorr signature on the whole super-transaction.

Signature formation on the super-transaction for each participant is done as follows;

$$
\begin{aligned}
T &= \text{super-transaction} \\
Sig_i &= (u_i, K_{sum}) \\
K_{sum} &= \sum_i k_i\, G \\
s_{cmp} &= N\, s_i + \sum_{j \in \text{ins}} \delta_j + \sum_{j \in \text{ins}} \gamma_j - \sum_{k \in \text{outs}} \gamma_k \\
u_i &= k_i + H(K_{sum}|T)\, s_{cmp}
\end{aligned}
$$

where index, i, labels each participant, index, j, labels every TXIN, for N of them, belonging to the participant, and index, k, labels each TXOUT belonging to that participant. On summation from all participants, the multi-signature (u_{sum}, K_{sum}) represents a valid signature on the super-transaction in the same manner that would hold if this were a simple transaction.

At the end of this final signature pass, each participant should have a super-transaction that can be validated by public witnesses. But all connections between TXIN's and TXOUT's have been broken. All that anyone can see is that all TXIN are being spent, and each TXOUT must have derived from one or more of those TXIN, but no way to see which ones are associated. The leader node then publishes the super-transaction.

If a blame cycle must happen, each participant must divulge their shared secret keys. Then all other nodes can verify that all stages of the computation were performed correctly, according to the information sent previously. We then have known associations between TXIN's and TXOUT's. Any participant which cannot or will not do this is blamed for the fault, and the protocol restarts after taking note of the TXIN's associated with the cheating node.

But since secret shared keys were divulged during blame discovery, all participants must restart the protocol from the point of establishing new shared keying.

D. Group Formation

Each N epochs, the current Cosi leader and validators elect a validator which must serve as a bulletin board (bulletin board service provider, BBSP) for N next epochs, and include the public key of BBSP in the sealed block.

BBSP elections are the lottery based on distributed randomness, similar to the elections of the CoSi leader.

Each wallet intending to submit a transaction should broadcast a transaction intent message with a fresh ephemeral public key along with the signature proof of validity.

BBSP should listen to transaction intent messages from wallets and pool public keys from that messages into collections of P keys each.

On collection of P keys or on timeout of ?T? seconds, BBSP broadcasts transactions pool message, containing ephemeral public keys and signatures.

Each wallet that recognize its key in transaction pool message should sort the collection of keys from it in the ascending numerical order, form the hash of the collection, and use that hash as a random seed for the SPT session.

An SPT Session Leader is elected by forming the XOR of the hash value with the public key of each pool participant, and if the resulting value is the minimum in the list, that wallet should be chosen as an SPT Session Leader.

SPT Session Leader is responsible for publication of final super-transaction.

All other communications are p2p between members of the SPT pool.

TBD:

Values of N, P and T Concrete design of p2p broadcast protocol Should BBSP sign its transactions pool messages? Possibly, yes.

VII. Transaction Processing

A. Validation of New Transactions

On entry to the witness pool, a transaction needs to be validated. The steps are:

- For each TXIN, witness must locate the UTXO referred to by the ID, to find the public key, Bulletproof, and Pedersen commitment, used for signature formation. If any ID do not point to extant UTXOs, then the entire transaction is invalid.
- No two TXIN ID can point to the same UTXO, to prevent double-spend attempts.
- All of the cloaked public keys must be summed, along with all of the Pedersen commitments from the TXINs. Then all of the Pedersen commitments from the Bulletproofs in the TXOUT's must be subtracted from the sum, as well as Fee times the A curve generator. The Schnorr signature in the transaction must be checked against this summed public key.

$$\text{TXIN} = \{ID\} \tag{1}$$

$$ID \rightarrow P_{M,\delta}, \text{as a result of UTXO lookup} \tag{2}$$

$$ID \rightarrow C(x, \gamma) \tag{3}$$

$$P_{eff} = \sum_{i \in \text{ins}} P_i + \sum_{i \in \text{ins}} C_i - \sum_{j \in \text{outs}} C_j - Fee\, A \tag{4}$$

$$Sig(M, T) = (u, K) \tag{5}$$

$$T = \text{Entire transaction sans signature} \tag{6}$$

Now check that

$$u\, G = K + H_r(T, K)\, P_{\mathit{eff}}$$

If this equation holds true, then the transaction body has not been mutated, Mary has proven ownership of all UTXO mentioned in her TXIN's, and the transaction represents a proper zero balance.

- The Bulletproofs in every TXOUT must be validated to ensure that amounts are within proper range, that no money is being created on credit.

At this point, we now have a valid transaction to consider. Commitments are held inside each Bulletproof.

The paired TXIN's and their UTXO's must be marked as pending extinction, so that the UTXO's will not be available for matching with the next transaction. This prevents future double-spend attempts.

Because network communications do not guarantee message arrival order, it may be necessary to form the topological sort of incoming transactions, in case some of their TXIN's refer to UTXO's created elsewhere in the batch. An example would be where Mary spends some of her tokens, receives change back to herself, then spends that change in another transaction.

B. Block Validation

The leader node of the witness group is tasked with forming a tentative block for extension of the blockchain:

- Collect transactions from its mempool, validate each of them, discarding invalid transactions, and collect the remaining valid transactions into an ordered list of pending additions.

  The order must be such that all TXIN's must refer to UTXO's that are either already in the blockchain, or else the transactions containing the UTXO's must precede the TXIN transaction.
- The leader sums all *Fee* terms from the transactions and produces a new fee distribution transaction to send portions of the fees to deserving participants. That transaction is prepended to the ordered list of validated transactions proposed for the block. But unlike the other transactions, it will be sent in whole form to witnesses for validation since they won't have seen copies of it in their mempool.

  The leader node then constructs a tentative blockchain block from this list.
- The block body is constructed by stripping the TXINs and TXOUTs from each transaction, collecting these two components into two separate Merkel trees, and planting those two trees into the proposed block body.

  TXIN's only need to record the UTXO ID that they reference. Hence the TXIN Merkel tree is just a tree of hashes.

  Since UTXO ID is already a hash of the data, it can stand in for the initial hash codes in the leaf nodes. The TXOUT tree is a Merkel tree with UTXO's stored in each leaf node.

  To support eventual trimming of matching TXIN and UTXO from the blockchain, the TXOUT Merkel tree needs to tag its recorded data in the leaf nodes, to indicate whether data are physically present, or just the hash of what once was there. Internal nodes of Merkel trees always point to two child nodes and carry only a hash value. Leaf nodes may, or may not, have data, and never have children.
- The block header is constructed by filling in slots:
  - Version number
  - Epoch number - a monotonically increasing value that represents the height of the blockchain.
  - Previous block hash, computed as the hash of the block header of the current blockchain tip block.
  - Leader public key
  - $\gamma_{blk} = \sum \gamma_{adj}$, the sum of all γ_{adj} terms found in the block transactions, which includes the γ_{adj} from the leader's fee distribution transaction.
  - Top hash of TXIN Merkel tree in block body
  - Top hash of UXTO Merkel tree in block body
  - Ordered list of witness public keys for current epoch. The leader node is also considered a witness during the current epoch.
  - BLS Multi-signature slot - to be filled in as a result of consensus on block. Note that we will be using pairing-based cryptography to support the use of short, fast, BLS signatures. BLS signatures allow the formation of multi-signatures in one pass over the witness pool.
  - Bitmap of signers in the multi-signature

  The initial contents of the BLS multi-signature slots should be zero filled and, even after being filled in, never contribute to the computation of any *block header hash*. All other slots are concatenated to form a header hash pre-image. But when a full hash of the block is formed for messaging purposes, the entire block, consisting of block header and block body, are considered for the hash pre-image.
- Form a message which contains:
  - The proposed block,
  - The fee distribution transaction,
  - An ordered list of transaction ID's that went into the block. Transaction ID's are computed as the hash of the transactions. Each witness should have copies of them in their own mempool.
  - The hash of this message. All communications about this proposed block will reference this hash value.

  The leader will sign and broadcast this message to all witnesses. The witnesses will attempt to validate the proposed block construction.

Once constructed, the witness leader must then seek consensus for its proposed block from all the other witnesses. This happens in two passes among the witness group.

- **PrePrepare Phase**: The leader broadcasts the signed prepared message, and its hash, to all witnesses in a PrePrepare message. This is the only time that the entire message will be transmitted. All further communications will refer only to its hash value.
- Each witness will attempt to validate the block construction from the ordered list of transactions.
  - The message must carry a valid signature from the leader node, to prevent spoofing attacks.
  - The message must be the first and only PrePrepare message seen from the leader for the current epoch.
  - The hash of the proposed block and transaction list must agree with the hash value shown in the message. This hash value will serve as a message

identifier for all related messages during this epoch.

  - The version number shown in the block header must show the expected value.
  - The epoch shown in the block header must agree with the witness node's sense of the current epoch.
  - The previous block hash slot in the header must equal the block header hash of the current blockchain tip block.
  - The leader key shown in the header must match the leader known for the current epoch.
  - The list of known witnesses for the current epoch must agree with the list of witness keys shown in the block header.
  - The fee distribution transaction must be valid.
  - Every transaction indicated in the list of constituent transactions must be looked up in the mempool, found, validated if not already known to be valid, and the two Merkel trees must be reconstructed as the transaction list is examined. At the end of transaction examination, the two Merkel trees in the block body must be seen as constructed in the same manner.

    If a witness does not have an indicated transaction in its local mempool, then it cannot validate the proposed block.
  - The γ_{blk} sum in the header must equal the sum of all γ_{adj} terms shown in the transactions.
  - The top hash values for the TXIN and UTXO Merkel trees, must match the values recorded in the block header.

If a witness agrees with the proposed block construction, they sign the hash of the *block header* with a BLS signature, and reply to the leader node with a Prepare message that contains the original message hash value, their signature, and a value that represents the position of their public key among the list of witnesses.

No signature on the message is required, as that is effectively already done with the block header signature provided in the message. The leader node will protect itself from spoofing attacks by checking that the returned signature on the block header is valid, using the public key looked up in the witness list at the indicated position.

If a witness disagrees or is unable to validate the proposed block, then they should simply refrain from responding. Responses from witnesses are expected to arrive within some timeout window. At the end of that window period, the responses obtained are collected together to form a multi-signature and bitmap.

The hash being signed covers only the block header, which contains top hashes of the body Merkel trees. By signing only the block header hash, we leave allowance for future block trimming actions, without disturbing the validity of the block. Trimmed Merkel trees always continue to show the same top hash value. When forming the block header hash, the slots which will contain the eventual multi-signature and signature bitmap are excluded from the hash.

An affirmative response from a witness signifies its willingness to commit the new block to their copy of the blockchain, when asked to do so. If the epoch changes to a new epoch before they receive a valid Commit message, then they will drop the block from further consideration.

- During the **PrePrepare timeout period**, the leader node accumulates valid Prepare responses from the witnesses. Every response is checked to see if it is a valid signature on the block header hash value. The public key for the signature is looked up in the list of witnesses according to the indicated position from the response message. If the signature is valid, and the response has not already been seen, then the signature is accumulated into an accumulating multi-signature, and the position of the witness is recorded as a bit in the multi-signature bitmap.

  At the end of the PrePrepare timeout period, the leader examines the census in the multi-signature bitmap. If that witness count is above a BFT threshold for the witness pool, then we consider that a consensus has been reached. If not, then the blockchain extension is aborted. The next epoch will elect a new leader, and the process will be retried then.

  If consensus has been reached, the multi-signature and the bitmap are stored in the block header. A Commit message is formed, containing the multi-signature, the bitmap, and the block message hash code. This message is signed by the leader and broadcast to all witnesses while we await another timeout period for responses.
- **Commit Phase**: All witnesses receive the Commit message and attempt to validate it:

  - The message hash code must agree with the message hash code from the PrePrepare message seen during this epoch.
  - The message should have been sent from the epoch leader node. The signature on the message must be valid for the known leader public key.
  - The census shown in the multi-signature bitmap must exceed the BFT threshold for the witness pool.
  - The muilti-signature should be a valid signature against the hash of the proposed block header. To obtain the public key for the signature validation, simply add up all the public keys from the witness list as indicated by the multi-signature bitmap.

  If the Commit message is seen as valid, then the witness stores the mutli-signature and its bitmap into its local copy of the block header, and commits the block to its local copy of the blockchain. A confirmation response is formed from their BLS signature on the block header hash, their position in the list of witnesses, and the message hash code. This response is sent back to the leader.

If the witness is unable to validate the Commit message, they simply refrain from sending back a response, and discard the pending block update.

- At completion of the Commit phase timeout period, the leader node accumulates all the response signatures into a multi-signature and signature bitmap. If the census of the bitmap exceeds the BFT threshold for the witness pool, and the mulit-signature is a valid signature on the hash of the block header, then a successful Commit has occurred. The leader and a BFT threshold number of witnesses have reached consensus on the block, and that block has now extended the blockchain.

A successful return from each phase, and the validation of multi-signatures, checks to be sure that the multi-signature is valid, and that the number of signing witnesses exceeds a Byzantine threshold for the group. If not, then the validation round is terminated, and no consensus has been reached in this round.

Since the blockchain is an accumulating history, and each block is an immutable record, it is not possible at this stage to discard matching TXIN and UTXO's. Rather, the posted TXIN's represent a marker that, when the blockchain will next be compressed, serve to indicate which UTXO's can be discarded in the compressed chain.

During the scan through the blockchain for matching UTXO's, these markers must be noted, and if a UTXO is found which matches a noted extinction, that UTXO must be ignored in the search.

VIII. Prevention of Direct Fraud

Fraud can work two ways. We have already shown how it would be impossible for someone else to redirect a spending transaction in one of Mary's TXOUT UTXO's. And to steal one of her TXIN UTXO's they would need her secret key to sign the transaction, and to learn the exact amount and cloaking factor involved. So the risk is dependent on the quality of her key management.

But what about the case when Mary is the dishonest one? Or suppose she just has faulty computer code and produces an errant UTXO? What can happen?

There are two places in a UTXO where a recipient is specified without any oversight from validation witnesses. Those are in the stated ephemeral public key of the recipient, P_{S,δ_2}, and in the keying for the encrypted payload which provides amount, x, and cloaking factors, γ and δ.

There are a number of ways to form the encrypted payload in the UTXO's. Sam may, or may not, be able to detect an improper decryption without also forming his own Pedersen commitment with the values he recovers for x and γ, and checking to see that his Pedersen commitment matches the one carried in the Bulletproof in the UTXO.

If the encrypted payload were keyed incorrectly, then Sam would not be able to read proper values for x and γ, and whatever values he might obtain would not produce the same Pedersen commitment value, C, as carried in the Bulletproof. Sam would be unable to spend the UTXO, even if he could produce a valid signature.

And if the public key were incorrectly stated, then nobody would be able to produce a valid signature on the UTXO.

A correct public key implies that the public key belongs to Sam, and he owns a corresponding secret key, s_S, so he can form a valid signature against the UTXO ID.

If Mary accidentally misstates his public key, then nobody would be able to form a signature on the UTXO ID. Or she might intentionally misstate the public key, perhaps to one of her own. Either way, there is nothing that could be detected about these situations by public witnesses.

Suppose Mary provides Sam's correct ephemeral public key, and suppose further that she encrypted the payload in a manner that allows Sam to read amount, x, and γ. If she misstates either of those two values, then Sam cannot spend the UTXO because they won't match the Pedersen commitment, C, stored in the Bulletproof. He can't sign a transaction with that UTXO, and so he cannot spend it. Again, no witnesses could detect these kinds of errors in the UTXO. Sam will find out later when he tries to spend the UTXO.

If Mary actually sent the money back to herself, through an alias key, and she tells Sam that she sent him the money, then it becomes a dispute between two individuals. But the UTXO will be properly formed, and Mary will be able to spend her own money, and the blockchain witnesses will be satisfied that a proper transaction had been formed. There is nothing fraudulent about sending money to yourself. The fraud in this case is confined to private communication.

IX. Prevention of Indirect Fraud

We have shown that it is mathematically intractable to try subverting the system with witnesses watching and verifying every step of transaction formation. But what about the situation where witnesses have either been prevented from validating or else subverted by attackers into overlooking obvious invalid transactions?

If nobody were watching, then actors could fabricate arbitrary amounts of new money. The blockchain would become horribly corrupted. So witnesses are needed to police the state of blockchain integrity. How many witnesses? One seems too few, it might be possible to corrupt them. But is 1,000 witness enough? or too many? I don't think one can give a black or white answer to such questions.

Cosi networks attempt to police the state of the blockchain by requiring a BFT threshold of consensus on transaction validity. But even that cannot guarantee with certainty.

Witnesses also offer some semblance of comfort to honest users, that their own transactions will not be mauled by attackers, or that their transactions won't be retracted after some later witness discovers corrupt blockchain state and attempts recovery.

So now we begin to approach the question of certification of blockchain state, and every transaction accepted into the blockchain needs to maintain an integral state. But what happens if that state maintenance facility becomes broken?

In the end, we have shown how to validate transactions in the blockchain. It is probably incumbent on all users to validate transactions that they are associated with before forming a new transaction based on those inputs. If every user did this, there would be no need of separate witness patrols. But even if there are witness pools, the possibility of corruption looms, and users should still rely on self-verification against party transactions that they rely upon. Users who do not perform their own validation will always run the risks of trust.

It would seem that a well policed blockchain can only offer best attempt service. There can be no absolute guarantees. But the mathematical machinery of our cryptography will provide absolute guarantees of correctness when they are used. The machinery will not corrupt. But it must be used to gain from its safety offerings.

But even here, the mathematical machinery is a hypothetical universe in which all computations are correctly carried out. Multiplication and addition never have bit errors. Real machines do suffer sporadic errors. And the best we have to overcome such errors are FEC (forward error correction) codes. The most notable is Reed-Solomon encoding which takes advantage of massive redundancy to detect and correct bit errors. Other codes perform better in some noise conditions, e.g., burst noise versus occasional bit errors.

The mathematical machinery of finite field arithmetic depends on each operation being correct. There are other kinds of crypto-systems that function more like fuzzy rendering systems, and allow for some level of noise corruption without invalidating the outcomes. Lattice-based crypto-systems come to mind here. But they are not compact systems. And our kind of cryptography would need to be adapted to a completely different paradigm, where Schnorr signatures may not even be possible.

References

[1] Satoshi Nakamoto. "Bitcoin: A Peer-to-Peer Electronic Cash System," https://bitcoin.org/bitcoin.pdf, October 31, 2008.

[2] Nicolas van Saberhagen. "CryptoNote v 2.0," https://cryptonote.org/whitepaper.pdf, October 17, 2013.

[3] Shen Noether, Adam Mackenzie and Monero Core Team. "Ring Confidential Transactions," https://lab.getmonero.org/pubs/MRL-0005.pdf, February, 2016.

[4] Benedikt Buenz, Jonathan Bootle†, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. "Bulletproofs: Short Proofs for Confidential Transactions and More," Cryptology ePrint Archive, Report 2017/1066, 2017. URL: https://eprint.iacr.org/2017/1066.

[5] I. Miers, C. Garman, M. Green and A. D. Rubin. "Zerocoin: Anonymous Distributed E-Cash from Bitcoin," 2013 IEEE Symposium on Security and Privacy, Berkeley, CA, 2013, pp. 397-411. URL: https://ieeexplore.ieee.org/document/6547123

[6] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. 2012. "From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again," in Proceedings of the 3rd Innovations in Theoretical Computer Science Conference (ITCS '12). ACM, New York, NY, USA, 326-349. URL: https://dl.acm.org/citation.cfm?id=2090263

[7] Tim Ruffing and Pedro Moreno-Sanchez. "Mixing Confidential Transactions: Comprehensive Transaction Privacy for Bitcoin," Cryptology ePrint Archive, Report 2017/238, 2017. URL: https://eprint.iacr.org/2017/238

[8] Torben Pryds Pedersen. "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing," Advances in Cryptology — CRYPTO '91", Springer Berlin Heidelberg, 1992, pages 129–140.

[9] Miguel Castro and Barbara Liskov. "Practical Byzantine Fault Tolerance," Proceedings of the Third Symposium on Operating Systems Design and Implementation, 1999, pages 173–186.

[10] Ewa Syta, Iulia Tamas, Dylan Visher, David Isaac Wolinsky, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ismail Khoffi, Bryan Ford. "Keeping Authorities "Honest or Bust" with Decentralized Witness Cosigning," arXiv:1503.08768v4 [cs.CR], 30 May 2016. URL: https://arxiv.org/pdf/1503.08768.pdf

[11] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. "Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing," arXiv:1602.06997v3 [cs.CR], 1 Aug 2016. URL: https://arxiv.org/pdf/1602.06997v3.pdf

[12] Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J. Fischer, Bryan Ford. "Scalable Bias-Resistant Distributed Randomness," Cryptology ePrint Archive, Report 2016/1067, 2016. URL: https://eprint.iacr.org/2016/1067.

[13] Ignacio Cascudo, Bernardo David. "SCRAPE: Scalable Randomness Attested by Public Entities," Applied Cryptography and Network Security, Springer International Publishing, 2017, pages 537–556.

[14] Youliang Tian, Changgen Peng, Renping Zhang, Yuling Chen. "A practical publicly verifiable secret sharing scheme based on bilinear pairing," 2nd International Conference on Anti-counterfeiting, Security and Identification, IEEE, 2008.

[15] M. Stadler. "Publicly verifiable secret sharing," in Advances in Cryptology–EUROCRYPT '96, volume 1070 of Lecture Notes in Computer Science, pages 190–199, Berlin, 1996. Springer-Verlag.

[16] Dan Boneh, Ben Lynn, Hovav Shacham. "Short Signatures from the Weil Pairing, " Journal of Cryptology", 2004, volume 17, pages 297–319.

[17] Tim Ruffing, Pedro Moreno-Sanchez, Aniket Kate. "CoinShuffle: Practical Decentralized Coin Mixing for Bitcoin," University of Saarland, Germany, 2014. URL: http://crypsys.mmci.uni-saarland.de/projects/CoinShuffle/coinshuffle.pdf

[18] Tim Ruffing, Pedro Moreno-Sanchez, Aniket Kate. "P2P Mixing and Unlinkable Bitcoin Transactions," University of Saarland, Germany, 2016. URL: https://crypsys.mmci.uni-saarland.de/projects/FastDC/paper.pdf

[19] Tim Ruffing, Pedro Moreno-Sanchez. "Mixing Confidential Transactions: Comprehensive Transaction Privacy for Bitcoin," Cryptology ePrint Archive, Report 2017/238, 2017. URL: https://eprint.iacr.org/2017/238

[20] David Chaum. "The dining cryptographers problem: Unconditional sender and recipient untraceability," Journal of Cryptology", 1988, volume 1, pages 65–75.