Stegos – A Platform for Privacy Applications

The Stegos Team, March 20, 2019
v1.0

Abstract— **In this paper we propose a design for a private, confidential and scalable privacy blockchain that is friendly to the environment. We start with an overview of existing privacy coins, then design a foundation to send payments and data with absolute privacy and complete confidentiality. We extend this foundation to provide a platform for building privacy applications that communicate via messages sent with minimal latency. We describe a Trusted Application Container (TAC), a mobile application that simplifies building privacy applications while providing additional security to users. Last but not least, we sketch the design of the Privacy App Store and private marketplaces.**

I. Introduction

In this section we give a 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 coin ideas and employs latest research in cryptography and blockchains in order to create privacy-preserving a blockchain that can be scaled.

A. Terminology

Here we give definitions for the 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 process that 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 is a consensus algorithm, where the creator of the next block is chosen via various combinations of random selection, wealth, age and staked funds. PoS blockchains can be more energy efficient than currencies based on PoS algorithms[2].
- **UTXO**: An Unspent Transaction Output that can be spent as an input in a new transaction.

B. Existing privacy coins

1) Monero: originated as a Bytecoin fork in 2014 under the name Bitmonero. The cryptocurrency is using a 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 enables 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 that 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 reduced a simple transaction size from 13KB down to 2.5KB, the problem with ever-growing blockchains cannot be solved.

2) ZCash: originated in 2016 as a Bitcoin fork and obviously is using a UTXO model and PoW consensus. The goal of the project is to improve on the flaws of Bitcoin, with a focus on privacy. The project is building on 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.

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 stealth addresses.

zk-SNARKS enable a large anonymity set of all minted

[1]https://digiconomist.net/bitcoin-energy-consumption
[2]http://cfa-consulting.ch/dlfiles/NxtEnergyandCostEfficiencyAnalysis.pdf

coins[3], providing a high degree of privacy. However, the size of up to 2KB for an average transaction and ever-growing accumulator, holding serial numbers of all spent coins, that cannot be pruned, makes ZCash much less scalable. The issue of scalability is the main reason why privacy is currently optional, not by default. At the time of writing, less than 25% of all transactions were shielded.

The questionable part of the zk-SNARKS protocol is the initial trusted setup. ZCash utilizes 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 the UTXO model and PoW consensus.

Besides standard nodes and miners, Dash has *masternodes*, the nodes that must have static IP address and fulfill 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 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[4]. 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 either in InstantSend, or in PrivateSend. Moreover, it is an 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 fulfill in case of amounts that are cloaked.

Users of Dash have to trust masternodes that they will keep users' IP addresses undisclosed and unlinked to users' UTXOs when sending transactions in Dash. This adds another privacy vulnerability to Dash.

4) Mimblewimble: is a protocol that was proposed by an anonymous user in a Bitcoin developers chat room by the name of Tom Elvis Jedusor, that left a link to a paper[5] 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[6], except, it is the recipient that generates a 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 the 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 newly mined block.

The two existing implementations of the protocol known at the moment are Grin[7] and Beam[8]. These projects use the UTXO model and PoW consensus. Spent UTXOs in both projects can be pruned by recursively applying a simple pruning algorithm for each UTXO referenced in inputs of the new minted block.

There are several drawbacks in the design of Mimblewimble:

- The first drawback is related to the transaction creation mechanism of MimbleWimble — in order to create a transaction, senders and recipients 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.
- The second is very similar to the problem from Dash where users must trust Dash masternodes. Currently, Grin and Beam users must trust miners that they will not trace the history of inputs and outputs in transactions, and discard this data completely after mining a block. Therefore, there is a threat to the coins' fungibility and the privacy of users.

II. 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.

A. Absolute privacy

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

- Stegos uses one-time payment addresses which make it impossible to identify recipients of a transaction, e.g. to link addresses of recipients to their identities, because all transactions are directed to new and unique addresses. The technique we use for one-time addresses is very similar to stealth addresses used in Monero and ZCash.

[3]The *anonymity set* here defined as the set of participants who could be a sender in ZCash transaction, as seen by a global observer who has also compromised a set of nodes.

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

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

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

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

[8]https://www.beam.mw

- Stegos makes it impossible to trace the history of transactions since many individual transactions are pooled together to form a super-transaction. For this purpose, we adopted and enhanced the 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.

B. *Proof-of-stake (PoS) consensus*

Stegos is friendly to the environment and does not require terawatts of electricity to be spent for mining blocks. Stegos uses 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 a performance bond or stake.

The size of the coins staked has a direct effect on the probability of a validator becoming the leader of a consensus round and earning block rewards and transaction fees.

C. *Data compaction*

Many projects claim to be able to process a million transactions per second (TPS) but none of them explain how they are going to maintain all the accumulated data. Bitcoin provides for 7–10 TPS and the Bitcoin blockchain is expected to grow past 170 gigabytes by the end of 2018. If we assume that Bitcoin suddenly supports 16,000 TPS, the Bitcoin blockchain will grow by 350 gigabytes every day[9], or 127 terabytes every year. This amount of data is completely unsustainable unless the blockchain will be centralized on a few supercomputers, something that's contrary to blockchain's decentralization ethos.

D. *Sharding for scalability*

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 the 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 enables Stegos process thousands of transactions per second.

E. *Fast message bus*

Unlike other blockchain, Stegos does not restrict the user to sending just payments. Data is a first-class citizen on the Stegos blockchain and users can exchange data messages with the same privacy guarantees that are afforded to payments. Sending data messages is actually much faster than sending payments. Fast data messaging underpins the Stegos Privacy Platform.

Think of Stegos as the message bus for cooperating privacy applications.

F. *Stegos mobile app*

Stegos is developing an native mobile application that integrates a wallet, one-one-one and group chat, the TACX-A, as well as the App Store.

1) Trusted Application Container (TAC): Developing native mobile applications is hard and error-prone. Developing secure mobile applications much harder still and blockchain only adds an extra level of difficulty. Stegos aims to make developing mobile privacy applications quick and simple.

The Stegos Trusted Application Container (TAC) is a native mobile application and a container for deploying privacy applications. The applications are built using familiar technologies like Javascript, HTML and CSS, and run as plugins in a *sandbox*, with tightly controlled access to the user's wallet as well as the outside world.

2) App Store: We look forward to a blooming ecosystem of privacy applications, developed to run in the Trusted Application Container (TAC)X-A. The experience of existing mobile app stores shows that users need a way to easily discover and install applications. The Stegos Privacy App Store provides a mechanism to search and install privacy applications, as well as rate their usefulness.

G. *Incentives to drive adoption*

Stegos includes incentives to drive adoption and encourage viral marketing.

Jackpot 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.

Users create *tickets* with their lucky numbers and post them to the blockchain. The Jackpot is distributed every few thousand blocks when validators run a cryptographic lottery based on verifiable distributed randomness to generate the winning number. Several tiers of prizes are awarded depending on how closely the ticker numbers match the winning number.

Red envelopes[10] is a WeChat-like feature that we implement to promote viral growth. Users create red envelopes with a given number of tokens and post the corresponding QR code or URL on social networks. Each attempt to open the red envelope may result in a few tokens or none at all.

In the following sections we will explain each of the Stegos features in more detail.

III. Networking

The Stegos network is composed of *bootstrap nodes*, *compact nodes* and *light wallets*. Bootstrap and compact nodes maintain a copy of the blockchain and associated data structures. They post a performance bond (stake) to become *validators*, then participate in the consensus protocol and

[9]https://hackernoon.com/if-we-lived-in-a-bitcoin-future-how-big-would-the-blockchain-have-to-be-bd07b282416f

[10]https://en.m.wikipedia.org/wiki/Red_envelope#Digital_red_envelopes

earn block rewards, as well as transaction fees. These nodes also respond for request from light wallet nodes.

Bootstrap nodes carry the unpruned version of the blockchain and respond to bootstrap requests new compact and bootstrap nodes. Compact nodes carry the pruned version of the blockchain, i.e. the current UTXO set.

Light nodes only keep blockchain headers and know how to talk to validators.

Stegos will initially maintain the core of the network by running a number of validator nodes to ensure continuity and performance of the network. The addresses of the core nodes will be hard-coded into each release of the Stegos blockchain software.

Nodes keep a list of addresses of the nodes they know about (peers) and add new peers to the list as they become aware of them. A brand new node will connect to one of the core nodes to fetch a list of bootstrap nodes and download a copy of the blockchain.

Each full node quickly re-broadcasts received transactions to a subset of its peers, after 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, e.g. by blocking them from further participation in the network.

The Stegos blockchain uses a gossip protocol to spread information without depending on a fixed network topology. 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. Each node knows about just a few peers and information can safely propagate through the network as long as most nodes know of at least two peers.

IV. Randomness

Bias-resistant distributed randomness is a critical component of 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 verifiably unbiased random secret-sharing groups, which repeatedly produce random output at predefined intervals.

We use RandHound++ for multiple purposes:

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

See Appendix I for more details of our randomness implementation.

V. 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.

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 multi-signature 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.

Please see the Appendix V for more details on how we use CoSi to sign blocks.

B. Leader Election

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

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

A validator elected to become the leader of the current epoch is rewarded by collecting transaction fees from the blocks they form, as well as collecting the block reward.

Leader election is done using a stake-weighted lottery. The larger the validator's stake, the greater the chances of becoming the leader of the current consensus round (epoch). After serving as the leader of one epoch, the leader relinquishes his right to be re-elected until a number of rounds later.

Please see Appendix II for the exact details of our consensus protocol.

C. Blocks

Briefly describe our key blocks and microblocks

We need to be able to refer to key blocks and microblocks in FastData

VI. Snowball

A. A problem of establishing untraceability

Even though user identities are cloaked in the blockchain, the spending history of every UTXO can be established by tracing blockchain transactions backwards, up to the genesis block.

Despite having 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 — a malicious node may join our network right after mainnet release, to collect transaction history in order to analyze and trace UTXOs.

To counteract this, we want to implement a protocol that completely hides the relationship between inputs and outputs of each transaction. This is difficult because UTXOs must have a unique ID. Without such a unique ID there's no way to validate transactions or prove ownership of a UTXO. Unique UTXO ID establish a trail which we want to obscure by only showing that UTXO output could have come from one of many different and unrelated inputs. We want to hide the source of each UTXO but in a manner that allows public validation.

B. Possible solutions

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

1) Mixers: require that users trust the third-party which provides mixing services, something that's unacceptable in a truly private and confidential blockchain.

2) Ring Signatures: collect a large but random number of existing UTXOs and add those into the list of inputs actually being spent. A signature is formed on all of the inputs, enabling proof that the inputs are properly spent as a group, without revealing the exact inputs being spent.

Ring signatures prevent blockchain compaction since it's impossible to know if a UTXO becomes spent and prune it from the blockchain. All UTXOs *that ever existed* must be retained which does not 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. Anyone with access to the secret randomness used to generate the reference string is able to create false proofs that will look valid to the verifier. For a cryptocurrency that uses zk-SNARKs, e.g. Zcash, this means creating counterfeit coins.

To prevent double-spending in a zk-SNARK-based cryptocurrency, a cryptographical accumulator has to be maintained by all nodes and contain the serial numbers of all spent coins. This accumulator always grows and cannot be trimmed, which scales poorly.

4) CoinJoin: is a protocol for joining several Bitcoin transactions together before submitting them to miners to include in the block. The protocol was originally proposed by Greg Maxwell in 2013 and 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."[11] CoinJoin implementations are based on the usage of trusted servers that provided the service of mixing multiple transactions together.

In 2014, CoinJoin inspired researchers from Saarland University to develop a completely decentralized protocol they called *CoinShuffle*[17]. It also allows users to mix their coins with those of other interested users and uses the accountable anonymous group communication protocol *Dissent* to ensure anonymity and robustness against DoS attacks.

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

The same researchers presented an enhanced version of the protocol in 2016, which they called *CoinShuffle*++[18]. The key innovation of CoinShuffle++ is replacing mix-nets with Dining Cryptographers Networks (DC-nets)[20], a more efficient anonymity mechanism.

A mix-net requires sequential processing so the number of communication rounds in the original CoinShuffle protocol grows linearly with the number of users. By using DC-nets, CoinShuffle++ allows for mixing to proceed in parallel, requiring a constant number of communication rounds regardless of the number of users.

C. ValueShuffle

We improve on *ValueShuffle*[19], an extension of CoinShuffle++ that is compatible with confidential transactions. ValueShuffle ensures the anonymity of mixing participants as well as the confidentiality of their payment values, even against malicious mixing participants. The ValueShuffle paper is missing some key details, though. For example, the paper provides neither details on forming a pool of senders who wish to mix their transactions, nor the details on how to form a signature on the resulting pooled transaction.

We implement a protocol where a *facilitator* elected among the validators provides the services of the *Bulletin Board* (as defined in the ValueShuffle protocol). We also implement collective Schnorr signatures[22] on the resulting transaction. Details of these protocols, as well as a brief explanation of our implementation of ValueShuffle, are given in Appendix IV.

VII. BLOCKCRUNCH

As described in Appendix V-B, a Stegos block is composed of a header and a body, where the header contains root hashes of two Merkle trees, that comprise the body of the block. Those two trees are a tree of all inputs (TXIN Merkle Tree) and a tree of all outputs (TXOUT Merkle Tree).

When a new block is verified and signed by CoSi witnesses, the signatures are computed over just the block header. The body of the block is not signed because it is secured against modification by the nature of the Merkle tree. It will be impossible to modify the body of the block without invalidating the root hashes contained in the signed header.

After verifying and signing a new block, the *leader* broadcasts it to the network. All nodes must validate the collective block signature and process the new block using the following steps:

- A node must retrieve all leaves of TXIN Merkle Tree from the block body, and for each leaf (UTXO ID) do the following:
    1) Find the block that contains the UTXO with ID
    2) Find a leaf in the TXOUT Merkle Tree of that block that contains the UTXO

3) Mark the leaf as *SPENT* without modifying the value of the hash that belongs to that leaf
4) If a sibling leaf is also marked as *SPENT* then delete *both leaves and their corresponding hashes*. Then mark the parent node of these two pruned leaves as *SPENT* as well and restart from step 4. Repeat until there are no *SPENT* siblings left.

- If the process above applies to all UTXOs in the TXIN Merkle Tree of the block, delete that tree from the block body and leave just the TXIN Root Hash in the block header. Append the block to the local copy of the blockchain.

Stegos blockchain compaction is a continuous process where each incoming and properly signed block triggers the pruning. As a result of this continuous pruning, the Stegos blockchain becomes a database of unspent coins (UTXO), with no transaction history kept.

VIII. FastData

Stegos introduces data transactions in addition to regular payment transactions. Both types of transactions enjoy the same encryption and privacy protection. In fact, there's no way to tell payments apart from data messages.

Unlike payments that need to spend existing coins, data has no inputs so here's no need to obfuscate which inputs resulted in which output. This means that data does not need to incur the delay of going through SnowballVI and can be sent much faster.

Data messages are meant to be handled at the application layer and the payload is expected to include a sequence number. Unlike payments, data cannot be double-spent and there's no need to wait for data messages to become irreversible. The message sequence number should help applications order messages and check for messages that are missing.

Stegos data messaging is a lot like UDP/IP and serves as a message bus that all kinds of applications can use to communicate securely and privately.

Data messages are automatically recycled and removed from the blockchain using BlockCrunchVII.

IX. Token economics

Describe our token economics

Stegos is powered by the STG token, available once the mainnet is launched. When posted as a performance bond by nodes wishing to support the Stegos platform, the STG token gives validators the right to earn fees and block rewards.

A. Monetary policy

STG token emission will start at 14% during the first 4 years and halve every 4 years thereafter.

B. Block rewards and transaction fees

Describe block rewards and transaction fees

C. Jackpot

Describe the Jackpot

- A portion of block rewards goes to the Jackpot. - Calculated as 1/3 of the inflation rate. - Stakes forfeited by cheating validators go into the pot. - Users pay coins to buy lottery tickets. - The payment for each ticket is added to the pot. - Each ticket has 7 numbers specified by the user. - The range of the first 6 numbers is 1-33 and the last number 1-16. - A 7-number winning sequence is generated for each lottery run. - A tier of prizes for guessing the numbers right - 1 prize. Matching all the 7 numbers. - 2 prize. Guessing the first 6 numbers correctly. - 3 prize. Guessing any 5 numbers, plus the 7th. - 4 prize. Any 5 numbers, or 4 numbers plus the 7th. - 5 prize. Any 4 numbers, or 3 numbers plus the 7th. - 6 prize. Any 2 numbers + 7th, Any 1 number + 7th, just 7th. The price of the ticket back. - If the pot is not won then it keeps growing. - The jackpot runs at a random time. - Only tickets bought 1 week or more in advance participate in the draw.

Electing a leader of the consensus and choosing a winner in the Jackpot lottery uses 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.

D. Red packets

Describe the red packets

- A lot like token airdrops. - The red envelope is funded with a number of tokens. - A QR code is generated by the Stegos app and can be scanned by anyone. - Scanning the QR code opens the red envelope. - Each attempt to open the envelope will result in either some tokens or no tokens at all. - Opening the envelope results in tokens then these tokens are deposited into the wallet of the user who opened the envelope. - The envelope can be opened until there are no more tokens to give out.

A great promotion and viral tool!

X. Privacy applications platform

The Stegos Privacy Platform builds on top of our fast message bus VIII and makes developing mobile privacy apps a breeze. The Stegos mobile app is the primary window to the platform. It integrates the Trusted Application Container (TAC) with one-on-one and group chat, Privacy App Store and private marketplaces.

A. Trusted Application Container

The Trusted Application Container (TAC) is a sandbox for running plug-in apps written using HTML, CSS and

JavaScript. The architecture is very similar to that of WeChat mini-programs [12].

The TAC prevents apps from wreaking havoc on the host mobile phone. It also completely abstracts the blockchain from running applications, providing instead an interface (API) to send messages and access the wallet. The TAC tightly controls access to the outside world and ensures that spending tokens requires confirmation from the user.

Expand on the TAC?

Stegos will provide the software development kit (SDK) and documentation for building privacy apps. The release date for the software development tools will be announced at a later date.

B. Identities

Describe the need for identity and associated privacy tradeoffs

A Stegos wallet has a public key (address) associated with it. It's absolutely safe to post a Stegos public key on a website or plaster it on a street billboard. Stegos uses *stealth addresses* and payments to a public key are cloaked with two random values. This makes it impossible to find payments to someone by analyzing the blockchain. Only the sender and the receiver are privy to any exchange of information.

Stegos uses the UTXO model and each UTXO is best seen as a coin. There's no notion of identity in Stegos, although the wallet address can be used as one for the purposes of building a reputation or keeping a social score. Identities come into play as we discuss chat, the Privacy App Store and private marketplaces.

Stegos users can export their wallet address as a QR code.

C. Privacy App Store

Describe the App Store

- App fingerprints stored on the blockchain. - Apps themselves can be stored elsewhere with download URLs in the app manifest on the blockchain. - App bundles are signed with the developer's public key. - TAC verifies that the signature of the app bundle and the hash (fingerprint) match the information in the manifest. - TAC installs the app bundle locally and makes the app available in the Stegos mobile app.

D. Chat

Describe one-on-one and group chat

- Just your basic chat, except completely private. - Your public key serves as your user id and identity. - You can send the QR code of your wallet address to anyone and initiate an absolutely private chat. - You can create chat groups and invite people using QR codes.

E. Marketplaces

Describe marketplaces

[12]https://walkthechat.com/wechat-mini-programs-simple-introduction/

XI. Future work

Describe the future roadmap

1) Mobile staking: Talk about the advantages of mobile staking

2) Bitcoin-style smart contracts:

Describe programmable coins

XII. Next Steps

In this paper we laid out a design for Stegos - a private, confidential, and scalable blockchain that is eco-friendly, upon which applications for sending monetary values and/or arbitrary data can be built.

The Stegos team is working on implementation of the blockchain and the development progress can be tracked online at `https://github.com/orgs/stegos/projects/1`. Our source code is and always be 100% open.

As our implementation progresses, we will publish separate, more detailed papers focused on our most essential contributions to the blockchain ecosystem, like improved distributed randomness generation (*RandHound++*), our flavor of collective signing protocol and consensus based on it (*BLS CoSi/pBFT*), and our approach for transactions and data sharding.

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.

[21] Whitfield Diffie, , Martin E. Hellman. "New Directions in Cryptography," in IEEE Transactions on Information Theory, volume 22 (6), pages 644–654.

[22] C.P. Schnorr. "Efficient Identification and Signatures for Smart Cards," in Advances in Cryptology — CRYPTO' 89, 1990, Springer, pages 239–252.

Appendix I
Randomness

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 do not 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. There is 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 the 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 validator group leader, among the other purposes.

After every election, a validator 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 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 will publish a separate paper describing in more detail our improvements on the original RandHound protocol in the nearest future.

Appendix II
Consensus

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 comes in 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 have a better chance of winning.

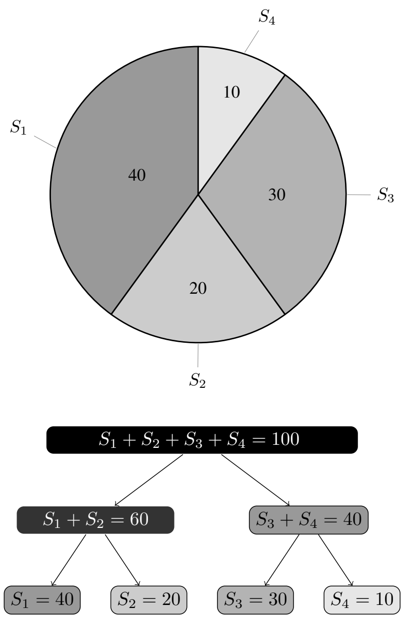

Fig. 1. Imagining a spin dial as a tree. Segments are sized in proportion to the posted stake.

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. This is done using the random value as a probe of the interval described by the partial sum at each node. The division between left and right are based on the relative weights of its two subtrees. Descent continues until meeting a leaf node, and declaring that node the winner of the leader election.

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

We can see that this works by noting that every interior node of the tree serves both as a denominator to nodes below, and as a numerator to connections above, hence cancelling out completely. The final probability of any participant winning becomes the size of their stake divided by the sum of all participating stakes.

1) Handling leader election 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 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 the 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.

A. Key Blocks

Stegos blockchain is able to accommodate multiple kinds of blocks. The kind of the block is defined by the *Block Type*

ID slot in the block header. In this paper we define two types of blocks: $0(KeyBlock)$ and $1(MonetaryBlock)$.

Monetary blocks, the blocks that carry all cryptocurrency transactions data and coins, will be described in detail in Section **??**. We will be referring to them simply as *blocks* throughout this paper.

The purpose of Key blocks is to deliver the administrative information to blockchain participants. Any node, searching through key blocks, can see who is current or past epoch *leader*, and who are witnesses (validators) for the current or past epoch. Nodes wishing to submit a transaction should consult with the latest key block to find who is the current pooled transactions *facilitator*.

The key block header is constructed by the current *leader* filling in the following slots:

- Block Type ID, 0
- Version Number
- Epoch Number
- Previous block hash, computed as the hash of the block header of the current blockchain tip block
- *Leader* public key
- Ordered list of witness public keys; the leader node is also considered a witness
- Pooled Transactions *facilitator* public key
- BLS Multi-signature, to be filled in as a result of the consensus on the block.
- Bitmap of signers in the multi-signature

There is no body in the key block.

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.

The key block verification and sealing should be done using the same consensus protocol as for the monetary blocks. We will describe this protocol in detail later on in this paper, in Section **??**.

Appendix III
Basic Transactions

In previous sections, we explained the high-level concepts of Stegos, including networking, randomness, and consensus. Now it is 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.

1) Pedersen commitment and Bulletproof: To form a Pedersen commitment, Alice multiplies the amount by a generator, A for the Elliptic curve group, E_r, 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:

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

$$x, \gamma \in Z_r$$

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.

2) Destination address cloaking: Alice then cloaks Bob's public key with a factor, $\delta \in Z_r$, so that instead of Bob's original public key P_B, she will put into UTXO $P_{B,\delta} = P_B + \delta\,G$.

3) Encrypted payload: Alice must pass values for the amount x, and cloaking factors, γ and δ, to Bob. She can do it by including this information into the *encrypted payload* of the UTXO. Besides the values mentioned above, the encrypted payload may also contain arbitrary data, that Alice wants to share with Bob. Detailed explanation of storing arbitrary data on Stegos blockchain will be given in Section **??**.

For generating the encrypted payload, Alice must choose random values $\alpha, k \in Z_r$ that will be used in creating and cloaking a symmetric data encryption key. The actual symmetric key for the encrypted data object will be $H(k\,G)$. In order to pass it safely via the blockchain to Bob, Alice needs to cloak it with α and store it in the UTXO as the following tuple:

$$Key_\alpha = (\alpha\,P_B + k\,G, \alpha\,G)$$

Alice encrypts her payload with AES-128 using $H(k\,G)$ key and places the encrypted payload into the UTXO as $E_B(x,\gamma,\delta)$.

When Bob receives UTXO later on, he will take out Key_α from it, multiply the second element of the tuple by his secret key s_B, getting $s_B\,\alpha\,G = \alpha\,P_B$, then subtract that from the first element to find $k\,G$. Afterwards he produces $H(k\,G)$ thus computing the symmetric key and decrypts the payload that Alice sent to him.

4) TTL and Data Size: Alice sets TTL (time-to-live) and $Size_{data}$ slots of the UTXO to zero, indicating that this is a *monetary UTXO*. Non-zero values of the mentioned slots will indicate that this is *data UTXO*, handling of which will be described in details later in this document, it Section **??**.

5) UTXO ID: Alice forms a UTXO ID by hashing the the whole UTXO object sans the ID slot, which contains the cloaked version of Bob's public key, $P_{B,\delta}$, the Pedersen commitment and Bulletproof, TTL, $Size_{data}$, 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.

6) UTXO structure: Thus the resulting structure of the UTXO object will be the following:

$$\begin{aligned} UTXO = (&ID, P_{B,\delta}, Bp, TTL, Size_{data}, \\ &Key_\alpha, E_B(x,\gamma,\delta)) \end{aligned}$$

where

$$\begin{aligned}
ID &= H(P_{B,\delta}, Bp, TTL, Size_{data}, \\
&\quad Key_\alpha, E_B(x,\gamma,\delta)) \in Z_r \\
H(arg_1, arg_2, ...) &= \textit{hash mapping of concat args} \\
P_{B,\delta} &= P_B + \delta\, G \\
G &= \textit{known generator for group } E_r \\
Bp &= \textit{Bulletproof and Pedersen} \\
&\quad \textit{commitment on amount}, x \\
\\
TTL &= \textit{Time-to-Live}, 0 \\
Side_{data} &= \textit{Data payload size}, 0 \\
Key_\alpha &= (\alpha\, P_B + k\, G, \alpha\, G) \\
E_B(x,\gamma,\delta) &= \textit{AES-128 encrypted payload}
\end{aligned}$$

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 need not 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 IDs 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 fees. 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 IDs, 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, $P_B = s_B\, G$. Then his effective secret key for the signature becomes:

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

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

$$K = k\, G$$

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

$$Sig(s_{eff}, T) = (u, K)$$

so that validators can see that:

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

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

where T represents the entire transaction record, sans signature. The function $H_r(x)$ represents the hash $H(x)$ mapped onto the field Z_r.

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 his 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 he 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:

$$ID' = H(P_{S,\delta_2}, Bp', TTL, Size_{data}, Key_{\alpha_2}, E_S(x - Fee, \gamma_2, \delta_2))$$

where P_{C,δ_2} is Charile's cloaked public key for this transaction, TTL and $Size_{data}$ are zero, and Key_{α_2} is a cloaked symmetric key.

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:

$$\text{TXOUT} = (ID', P_{S,\delta_2}, Bp', TTL, Size_{data}, Key_{\alpha_2}, E_S(x - Fee, \gamma_2, \delta_2))$$

To wrap up, Bob publishes the transaction:

$$\begin{aligned} \text{T} = \{&\text{TXIN} : \{ID\}, \\ &\text{TXOUT} : \{(ID', P_{S,\delta_2}, Bp', TTL, Size_{data}, Key_{\alpha_2}, E_S(x - Fee, \gamma_2, \delta_2))\}, \\ &\text{FEE} : \mathit{Fee}, \\ &\text{GAMMA} : \gamma_{adj} = \sum_{i \in \text{ins}} \gamma_i - \sum_{j \in \text{outs}} \gamma_j \\ &\text{SIG} : \mathit{Sig}(s_B, T)\} \end{aligned}$$

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. And this term will be added to a block sum when the transaction is absorbed into the blockchain, to show that entire blocks, which may contain many UTXOs, continue to show a zero balance.

The final line is Bob's signature asserting ownership of the entire transaction which is based on the hash of all the stated TXINs, TXOUTs, Fee, and γ_{adj} term. This final signature also 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.

Appendix IV
Snowball

A. Forming the pool of senders

Each epoch, besides selecting new witnesses, a leader, and a winner of the Jackpot, all validators select a node among them which must serve as a *Bulletin Board* in ValueShuffle protocol and include the public key of this node into the sealed keyblock for the new epoch. We call that Bulletin Board node *a facilitator*.

Each node intending to submit a transaction should broadcast a *Transaction Intent* message with a fresh ephemeral public key along with the signature proof of validity.

Facilitators should listen to transaction intent messages from the nodes and pool public keys from those messages into collections of K keys each. K defines a cardinality of the anonymity set by defining how many participants should form a transaction mixing pool. This is a tunable parameter which can be set for an each epoch of the blockchain.

Upon collection of K keys or on the timeout of T seconds, *the facilitator* broadcasts *Transactions Pool* message (signed with its private key), containing all collected ephemeral public keys and corresponding signatures.

Each node that recognizes its key in *Transactions Pool* messages should sort the collection of keys from it in ascending numerical order, form the hash of the collection, and use that hash as a random seed for the *Pooled Transactions Session*.

A pool leader is elected by participating nodes by forming the XOR of the hash value defined above with the public key of the each pool participant and, if the resulting value is the minimum in the list, then that node should be chosen as a *pool leader*.

A pool leader is responsible for publication of a final pooled transaction, that we call a *super-transaction*.

B. Establishing the pooled transaction session

All communications, including broadcasting, within the pooled transactions session should be contained within the group of the pool participants. All nodes of the pool will broadcast their lists of TXINs along with signatures validating ownership of their TXINs. Every participant can validate the lists from other participants by verifying the signature attesting to TXIN ownership. Participants should check that the public key referenced by the TXIN ID is from one of the group members.

1) Collecting inputs: A valid signature on the TXINs from one participant is formed by creating a Schnorr signature on the sum of the cloaked public keys shown in the UTXOs referenced in the TXINs:

$$\begin{aligned} TXIN &= \{ID_1, ID_2, ..., ID_N\} \\ \mathit{Sig} &= (u, K) \\ K &= k\,G \\ s_{cmp} &= Ns + \sum_i \delta_i \\ P_{cmp} &= NP + (\sum_i \delta_i)\,G = s_{cmp}\,G \\ u &= k + H_r(K, P_{cmp}, H(ID_1, ID_2, ..., ID_N))\,s_{cmp}, \end{aligned}$$

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

Verification of a signature is done by seeing that:

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

where the P_i are the cloaked public keys shown in the UTXOs corresponding to each ID_i. This signature proves ownership of the stated UTXOs referenced in TXINs.

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 TXOUTs, 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 TXINs again.

Once contributions have been received from each participant, or a timeout occurs, the resulting pool of TXINs is known to all participants. Since these are merely UTXO IDs 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.

2) Establishing pair-wise shared keys: All participants will establish pair-wise shared secret keys between themselves and every other participant of the pooled transactions session. 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. 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 a Diffie-Hellman secure key exchange[21]. With two participants, A and B, this can be established by having A send to B

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

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

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

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.

3) Producing DiceMix arrays for outputs: Next, each participant computes their TXOUTs with *fresh randomness*, chooses random k-factors for an eventual collective Schnorr signature, and produces both a DiceMix array containing the fragmented TXOUTs, 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 TXOUTs 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 TXINs and TXOUTs.

4) Forming the super-transaction: 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 participants' TXOUTs. These TXOUTs are reassembled, and a *super-transaction* is formed containing all TXINs, 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 TXOUTs Bulletproofs. They must also find their own contributions in the lists of TXINs and TXOUTs. 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.

5) Forming the collective Schnorr 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:

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

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.

6) Publishing the super-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 TXINs and TXOUTs will have been broken. All that anyone can see is that all TXINs 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 session leader* then sends the super-transaction into the network using gossip protocol for validation and inclusion into the block.

7) The blame cycle: 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 TXINs and TXOUTs. Any participant which cannot or will not do this is blamed for the fault, and the protocol restarts after taking note of the TXINs 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.

APPENDIX V
VALIDATION OF BLOCKS AND TRANSACTIONS

A. Validation of Transactions

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

- For each TXIN, a 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 does not point to an existing UTXO, then the entire transaction is invalid.
- No two TXINs 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 TXOUTs 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:

$$\begin{aligned}
\text{TXINs} &= \{ID_1, ID_2, ..., ID_N\} \\
ID_i &\to P_{i,\delta_i} = P_i \\
ID_i &\to C(x_i, \gamma_i) = C_i \\
P_{eff} &= \sum_{i \in \text{ins}} P_i + \sum_{i \in \text{ins}} C_i - \sum_{j \in \text{outs}} C_j - Fee\, A \\
Sig(s,T) &= (u, K) \\
T &= \text{Entire transaction sans signature}
\end{aligned}$$

  Now check that

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

  If this equation holds true, then the transaction body has not been mutated, the sender has proven ownership of all UTXOs referenced in the TXINs, 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 TXINs and their UTXOs must be marked as pending extinction, so that they will be pruned later on according to the procedure explained in Section **??**.

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

B. Formation and Validation of Monetary Blocks

1) Block formation: The leader 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 TXINs must refer to UTXOs that are either already in the blockchain, or else the transactions containing the UTXOs 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 itself and to the Jackpot. 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 Merkle trees, and planting those two trees into the proposed block's body. TXINs only need to record the UTXO IDs that they reference. Hence the TXIN Merkle tree is just a tree of hashes. The TXOUT tree is a Merkle tree with UTXOs stored in each leaf node.
  To support eventual trimming of matching TXINs and UTXOs from the blockchain, the TXOUT Merkle 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 Merkle 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 the following slots:
  - Block Type ID, 1
  - Version number
  - Epoch number
  - Previous block hash, computed as the hash of the block header of the current blockchain tip block
  - $\gamma_{blk} = \sum \gamma_{adj}$, the sum of all γ_{adj} terms found in the block transactions, which includes the γ_{adj} from the leader's fee distribution transaction.
  - Root hash of TXIN Merkle tree
  - Root hash of TXOUT Merkle tree
  - BLS Multi-signature, to be filled in as a result of the consensus on the block.[13]
  - 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 *Block Proposal* message which contains:
  - The proposed block
  - The fee distribution transaction
  - An ordered list of transaction IDs that went into the block. Transaction IDs 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.

[13]We are 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.

2) Block validation: Once constructed, the witness leader must then seek consensus for its proposed block from all the other witnesses according to the protocol described at a high-level in the Section **??**.

In this section we are going to clarify the details of the Stegos consensus protocol, which occurs in two passes among the witness group.

- **PrePrepare Phase**: *The leader* broadcasts the contents of the signed *Block Proposal* 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*, 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 Merkle trees must be reconstructed as the transaction list is examined; at the end of transaction examination, the two Merkle 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 root hashes for the TXIN and TXOUT Merkle 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* 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* 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 timeout 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 root hashes of the Merkle trees. By signing only the block header hash, we leave allowance for future block pruning actions, without disturbing the validity of the block. Pruned Merkle trees always continue to show the same root 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* accumulates valid *Prepare* responses from the witnesses:

  - Every response is checked to see if there 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 by the *leader*, 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*, 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 multi-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 multi-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 multi-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.

APPENDIX VI
FASTDATA

We already described the structure of the payment UTXO in Appendix III-A. Here we explain the structure of the data UTXO and show how transactions can be composed using a mix of payments and data.

A. Structure of the Data UTXO

Data UTXOs have the same structure as monetary UTXOs, which has been explained in Section **??**. The difference is in the values of Bp, TTL, $Size_{data}$ and the encrypted payload slots:

- Bp slot should only contain Pedersen commitment to zero, $\gamma\, G$, without Bulletproof range proof.
- TTL (Time-to-Live period) should be set to a number of blocks for which this UTXO should be kept on the blockchain since it has been added to it.
- $Size_{data}$ should be set equal to the size of encrypted data, in bytes, that will be stored on the blockchain as a part of encrypted payload.
- The encrypted payload should be enhanced with the fourth element, *data*, to accomodate the data to be stored. The first element of the payload, x, should be set to 0.

Consider an example where Alice wants to send Bob some data object. As before, P_B and s_B will stand for Bob's public and private secret key correspondingly.

Alice should compose the following data UTXO:

$$\begin{aligned} UTXO_D = (&ID, P_{B,\delta}, PC, TTL, Size_{data},\\ &Key_\alpha, E_B(0,\gamma,\delta,data)) \end{aligned}$$

where

$$\begin{aligned}
ID &= H(P_{B,\delta}, PC, TTL, Size_{data},\\
&\quad Key_\alpha, E_B(0,\gamma,\delta,data)) \in Z_r\\
P_{B,\delta} &= P_B + \delta\, G\\
PC &= \textit{Pedersen commitment to zero}, \gamma\, G\\
TTL &= \textit{Time-to-Live of the UTXO},\\
&\quad \textit{in number of blocks}\\
Size_{data} &= \textit{Data payload size, in bytes}\\
Key_\alpha &= (\alpha\, P_B + k\, G, \alpha\, G)\\
E_B(0,\gamma,\delta,data) &= \textit{AES-128 encrypted payload},\\
&\quad \textit{including the data}
\end{aligned}$$

B. Transactions involving data UTXOs

Because data UTXOs have the same structure as monetary UTXOs, and they have Pedersen commitments to zero included in their bodies, they can be easily mixed in the same transactions with monetary UTXOs, because they will not influence the monetary balance of transactions. The algorithms for validating transactions will stay the same, as shown previously in this paper.

There are a few differences of transactions with mixed UTXO types from "normal" tranactions:

- The first difference is that a value of the Fee slot in a transactions must obey the following formula:

$$\begin{aligned}
FeeUnit &= \textit{Fee for keeping 1024 bytes for 1 block}\\
SizeKB_i &= \frac{Size_{data_i}}{1024}, \textit{rounded up to the next integer}\\
Fee_i &= SizeKB_i * TTL_i * FeeUnit\\
Fee &= \sum_i Fee_i
\end{aligned}$$

  So, for each data UTXO in outputs of the transaction, the fee must be calculated proportionally to the size of the data stored in that UTXO and time-to-live for that data. If there are multiple data UTXOs in outputs of transaction, then these UTXO's fees they must be summed up in order to calculate the final trasaction fee. If there are monetary UTXOs in outputs as well, then ordinary monetary fees must be also added to the final fee of the transaction.
- The second difference is that it is possible to leave outputs in the transaction completely empty if only data UTXOs are referenced by the transaction inputs. In this case, Fee slot for such a transaction can be set to 0.

C. Pruning of used and expired data

If UTXO ID of the data UTXO was referenced in inputs of a transaction and subsequently were validated and included into a new sealed block, then all full nodes of the blockchain will process that block according to the pruning procedure described in Section VII and delete the corresponding UTXO from the blockchain. So, there is no difference in pruning spent monetary and used data UTXOs.

With that, a special mechanism should be established for deleting expired data — those data UTXOs which haven't been referenced in inputs to any registered transaction, and whose TTL timers have expired.

The duty to identify expired data UTXOs belongs to the epoch *leader*. In addition to the normal duties of the leader, now, in the presence of data UTXO in the UTXO Database, before proposing the new block to witnesses in the group, it must scan the database and check in each data UTXO a number of the block in which it has been created and its TTL. If the sum of creation block number and TTL is less than number of the new block that the leader is going to propose, then the leader must add this UTXO's ID to the inputs to special data cleaning transaction which should be added to the proposed block along with other special transactions for collecting fees.

Upon receiving block proposal from the leader, witnesses will validate that UTXOs referenced in the data cleaning transaction are indeed expired and seal the block with their signatures. Finally, the sealed block will be propagated through the network, and all full nodes will prune expired data UTXOs according to the usual pruning procedure.

TODO LIST