The Discreet Ledger

Formal Specification

Revision 1.0

Published June 2023

Authored by Brandon Koerner & Frederik Markor

The Discreet Ledger Formal Specification

Brandon Koerner
brandon@discreet.net

Frederik Markor
frederik@discreet.net

June 29, 2023

Abstract

This paper introduces a scalable, permissionless, fully confidential asset remittance protocol called *Discreet*. We present a concise construction for a transactional ledger-based digital currency with stronger privacy guarantees than currently exists, achieved by leveraging several recent advancements in cryptography. Utilization of mixed transactions allows for anonymous and confidential outputs to be deshielded and spent in a public manner for the purposes of compliance and regulation. Discreet is governed and secured by a novel Proof-of-Stake consensus algorithm, Aurem, which obfuscates stakes and block validators, while guaranteeing fairness without the need for a trusted dealer. It also ensures fast finality, and scalability to accommodate for global usage of the network. The native asset of the protocol, *$DIST* is used for transacting value across the network.

Keywords— privacy, cryptocurrency, zero-knowledge, DAG, consensus

Contents

1 Introduction

Discreet is a novel privacy-oriented decentralized value transfer protocol. As focus in the industry shifts towards using decentralized currencies as means of exchange, the foundational layer of any value transfer system must adhere to strict privacy standards to mimic the behaviour of fiat currency.

Building on recent work within cryptography, the protocol introduces three necessary components for an effective value transfer protocol that are essential for a currency to be a viable candidate for remittance.

1. Confidential transfers: all transactions performed on the Discreet network are confidential by default, with information only being discernible by the sender and receiver. Funds can optionally be transferred as non-confidential transactions for purposes such as compliance.
2. Scalability: Various cryptographic schemes are used to reduce overhead and ensure a light-weight network. All transactions are propagated anonymously via an *anonymity graph* in the peer-to-peer network, shielding the IP address of the origin and finalized at the consensus-layer.
3. Finality guarantees: All transactions are settled and finalized within 15 seconds. The settlement is secured by our novel confidential proof-of-stake consensus, called Aurem, a permissionless consensus which blinds the staking amounts of committee participants and removes the need for a trusted dealer, instead utilizing a randomness beacon.

This document provides definition and specification to the underlying value transfer system for the Discreet distributed ledger. Further work will specify and outline the full technology stack and core features in detail.

2 Preliminaries and Definitions

Before presenting the distributed ledger and coin model, it is necessary to define some terms. Unless stated otherwise, these definitions can be considered consistent in all presented sections in this document.

2.1 Public Parameters

Define $\mathbb{G}$ to be a cyclic group of order $l > 3$, such that the discrete logarithm problem (DLP) is hard, and the decisional and inverse decisional Diffie-Hellman assumptions hold. Define the scalar field of $\mathbb{G}$ as $\mathbb{F}_l$. Let G, H and U be generators of $\mathbb{G}$ with unknown discrete logarithm relationship; assume all subscripted or indexed G and H are independent generators of $\mathbb{G}$ as well, with unknown DLP relationship. Define two hash functions modeled as random oracles: $\mathcal{H}^p : \{0,1\}^* \to \mathbb{G}$, $\mathcal{H}^l : \{0,1\}^* \to \mathbb{F}_l$; in lieu of explicit domain separation, a subscript is added to $\mathcal{H}^l$ and $\mathcal{H}^p$ as necessary. Consider, for all hash functions, the hardness of both preimage and second preimage attacks. Additional public parameters may be presented, and will be explicitly stated to be so; consider these to also be consistent in definition. These additional definitions will be in their respective sections for the sake of context.

2.2 Blockchain Definitions

The following sections are for the definition of concepts relating to distributed ledgers, blockchain, and cryptocurrency for Discreet.

2.2.1 Distributed Ledger

A distributed ledger represents a distributed data structure consisting of a state and a sequence of state transitions with total ordering. For cryptocurrencies, the state transitions are transactions in a value transfer system, with the state being the outputs of transactions. The specific structure of the state, along with the rules governing the validity of transactions, vary between cryptocurrencies and their implementations.

Often times, it is useful to define a distributed ledger's accounting system, which can be defined as the method for storing values in each address/account. There are two main methods of accounting for distributed ledgers: the account-based model and the unspent transaction output (UTXO) model. In an account-based model, each element in the state represents an account, i.e. an addressable unit that carries value. For instance, on Ethereum's distributed ledger, each account consists of an address, a strictly increasing nonce, a value, and a hash of an autonomous script which governs the account.

The benefit of an account-based model is the ease of implementation for complex governance and autonomy, as can be seen with Ethereum's EVM and smart contracts. However, such a model depends on a shared mutable state, thus individual transactions may be manipulated to produce more desirable outcomes by miners and users. Additionally, the semantics of contract code become more complex, and authors of said code are required to understand these complexities or risk adding vulnerabilities and security issues.

The other accounting model for distributed ledgers, the UTXO model, does not rely on a shared mutable state, and thus the semantics for state transitions become simpler, at the cost of expressiveness. In this model, the state consists of a set of unspent transaction outputs, or UTXOs, which abstract the concept of valued and addressed outputs in a transaction for a value transfer ledger. Each of these UTXOs typically stores an address derived from a user's public key and an amount. The state transitions in this model are a set of input UTXOs, called transaction inputs, and new UTXOs. The rule for validity for a transaction in the UTXO model is (1) transaction inputs must be present in the set of UTXOs; (2) the sum of values in the inputs must equal the sum of values in the outputs; and (3) all transaction inputs and outputs must be unique and can only ever be spent once.

As mentioned previously, the UTXO model is not on its own as expressive as the account-based model. However, recent developments have created variants of the UTXO model which, qualitatively, have similar expressiveness as account-based models, with proof of their validity. We will go more into detail in the programmability section on this.

Discreet will implement a distributed ledger equipped with a variant of the UTXO model which allows for both private (i.e. anonymous and confidential) transaction outputs and standard transaction outputs (i.e. those seen in transparent distributed ledgers). This implementation was chosen for two reasons. First, digital cash should emulate the privacy of fiat cash, where transactions between users do not provide transparency of said user's balance or spending history. Secondly, support for a transparent currency model allows for compliance with exchanges. It should be stated that supporting both types of transfers will not sacrifice the utility of either; privacy is preserved for private transactions and UTXOs.

2.2.2 UTXO

UTXOs are the atomic unit of the Discreet ledger's state. Since Discreet implements both a transparent and private currency model, UTXOs will either be private or transparent (which will be used interchangeably with the term public). Public UTXOs are referenced by the hash of the transaction which created it and an offset representing the index of the UTXO in the transaction's output array. This is enough to guarantee

that public UTXOs will have a unique reference for a hash function under the random oracle model. Private UTXOs are referenced by a unique number assigned in order of appearance in the ledger. This suffices due to the nature of the privacy and spending rules for private transactions, and guaranteed to be unique and consistent across the distributed ledger under the performance of the chosen consensus mechanism. Please note that the terms UTXO, transaction output, and e-note may be used interchangeably with one another in this and all following sections.

As defined previously, public UTXOs contain a reference to their source transaction, an address representing the owner of the UTXO, and a value associated with the UTXO. Private UTXOs contain a reference to their source transaction, a public key representing the destination address, a potentially encrypted amount, and a commitment to the value. The exact role the destination address public key, encrypted amount, and commitment play in the coin protocol will be explained in the base transaction model section.

2.2.3 Wallets

Cryptocurrencies make use of public key cryptography in a variety of ways. Mainly, a key pair is generated representing an account, with the private key used for authorization of payment, and the public key used to derive an address associated with the account. A wallet is a collection of accounts derived from a singular source of entropy and randomness. Each wallet is generated in such a way that they can be recovered from a seed phrase, generated from the wallet's entropy. Thus, funds are recoverable if the seed phrase is backed up in the event of corruption or loss of data.

Public accounts are accounts which possess public UTXOs and can create transparent transactions. They consist of a key pair, where the public key is hashed using RIPEMD160 and the resulting hash is used to construct the account's address. This address can be shared with others to receive funds. Funds in the form of public UTXOs can be spent by authorizing their use as an input in a transaction via a cryptographic signature. The signature signs data unique to both the transaction and the output using the private key associated with the public key the address is derived from.

Private accounts are used to spend and receive private UTXOs, and can create private transactions. They consist of two sets of key pairs, called the spend keys and view keys. Spend keys are used to authorize payment, and view keys are used to decrypt values stored in private UTXOs. Their full use in authorizing payment and receiving funds will be explained in a following section.

2.2.4 Transactions

Transactions represent a state transition in the distributed ledger. Mainly, they are an object which transfers funds via the spending of UTXOs and the creation of new ones. In the context of more general distributed ledgers, they both represent state transitions governed by contract-specific code and value transfers, as seen in Ethereum.

Discreet provides three constructions for transactions in the base value transfer system: private, transparent, and mixed transactions. Private transactions spend and create private UTXOs only; transparent transactions spend and create exclusively public UTXOs. Mixed transactions bridge the two types of UTXOs and spend one type (i.e. either all private or all public inputs), and can create UTXOs of either type.

2.2.5 Blocks

Blocks are an abstraction useful in distributed ledgers, and aggregate transactions into a single unit. In the timestamp server model of distributed ledgers, they assign a timestamp to the transactions and a total ordering over a set of transactions in the block. They reference a previous block or multiple blocks, thus forming either a

blockchain or directed acyclic graph (DAG). Under some form of consensus, a chain or DAG of blocks may reach a consistent state in a distributed network.

2.3 Nodes

Nodes are instances of the Discreet distributed ledger software and directly validate all structures in the distributed ledger. Nodes also participate in a consensus mechanism to achieve consistency in the distributed ledger's state through the process of minting (i.e. creating) new blocks and linking them in the blockchain or DAG. Not all nodes directly participate in consensus, and different types of nodes may exist, such as archival nodes which store a non-pruned distributed ledger.

2.4 Consensus

In order for a distributed ledger's state to achieve consistency in a potentially asynchronous network, a method of reaching consensus among nodes is necessary to define. A consensus mechanism is a protocol operating on transactions and/or blocks to create a total ordering on the state transitions, thus achieving a consistent state. This is often achieved via the process of minting or proposing new blocks, propagating them and other nodes either accepting them or rejecting them based on some kind of vote or other set of governance rules.

Discreet will make use of Aurem, a novel combination of confidential proof-of-stake and any asynchronous byzantine fault tolerant system. Aurem builds on the work of Orlandi, et. al [1] by allowing zero knowledge proofs of candidacy weighted proportionally by a stake to choose validators for participation in a given unit of consensus. The implementation of this is independent of the transaction/coin protocol and sits on top of it, but below the finality layer. The finality layer makes use of a variant of AlephBFT. More details can be found in the Consensus section.

3 Base Transaction Model

In this section we will define the coin protocol specific to the privacy aspect of Discreet. This can be taken as the private transaction model used to construct Discreet's value transfer system.

3.1 Cryptographic Definitions

Discreet's public key cryptography will be implemented as elliptic curve cryptography under the Ed25519 curve, and the hash function to be used is SHA256. The advantage of these choices is that curve scalars, compressed points, and hashes will all be 32 bytes in size. Ed25519 is not a prime order group, and has order $8 * l$ (where l is a prime $\approx 2^{252}$); however, all curve points will be restricted to use the subgroup of size l.

3.2 Constructions

3.2.1 Addresses

We define a private account's view key pair to be (k^v_*, K^v_*) and spend key pair to be (k^s_*, K^s_*), where $*$ will indicate ownership, k indicates private key and K indicates public key. The address corresponding to a private account with this structure is called a *stealth address*. A stealth address is constructed from public keys (K^v_*, K^s_*) via a Base58 encoding in the following way:

$$A_* = \text{Base58}(\text{0x1} \,||\, \text{ToBytes}(K^s_*) \,||\, \text{ToBytes}(K^v_*) \,||\, \text{chk}) \tag{1}$$

Where 0x1 is the version byte for Discreet, currently set to 1, and chk is a checksum calculated using a hash function $\mathcal{H}$:

$$\text{chk} = \mathcal{H}(\text{0x1} \mid\mid \text{ToBytes}(K^s_*) \mid\mid \text{ToBytes}(K^v_*))[0:4] \tag{2}$$

Given the concatenation operator $||$ and the bytestring slice operator $[* : *]$, which returns the byte substring beginning at the first index, inclusive, and ending at the last index, exclusive.

3.2.2 DKSAP

Stealth addresses are used in a protocol for building *one-time destination addresses*, which guarantee that an observer could not recover any information about the receiving party's address in a transaction. This is known as the dual key stealth address protocol, or DKSAP.

For a spender Alice who has a receiver Bob's address (K^v_B, K^s_B), Alice can perform the following steps:

- Alice generates a random scalar $r \in \mathbb{F}_l$ and constructs a key pair (r, R) where $R = rG$.
- Alice then computes a Diffie-Hellman shared secret using Bob's view key and reduces it to a scalar $c = \mathcal{H}^l(K^v_B r)$.
- Alice finally computes the one-time destination address' public key $T = cG + K^s_B$ and publishes (R, T) as part of the transaction.

Bob with private view key k^v_B and private spend key k^s_B can easily recover his private key to his one-time destination address Alice created. He does this by first recovering the Diffie-Hellman shared secret $c = \mathcal{H}^l(k^s_B R)$ and constructing the one-time destination address' private key $t = c + k^s_B$. Bob can verify he is the receiver by checking if $tG = T$. A visualization of this protocol can be found in Figure 1.

The key R in this protocol is referred to as the transaction public key, and can be reused for multiple receivers in the same transacction by domain separating the hash function used to calculate the Diffie-Hellman shared secret. This is done in practice by including the index of the private output containing the one-time destination address in the calculation.

3.2.3 Commitments

A commitment to a value $\text{Com}(v)$ can be defined as a one-way function operating on a value which does not reveal the value itself. The commitments used for Discreet are *Pedersen commitments*, which are commitments to values that are additively homomorphic; i.e. $\text{Com}(v_a) + \text{Com}(v_b) = \text{Com}(v_a + v_b)$. Pedersen commitments are constructed using the group generators G and H:

$$\text{Com}(a, r) = aG + rH \tag{3}$$

For a value $a \in \mathbb{F}_l$ and a randomness $r \in \mathbb{F}_l$. We require the commitment scheme to be perfectly hiding, which is to say, for the generator $H = \gamma G$, where γ is unknown, an attacker cannot find any a' and r' where $a' + r'\gamma = a + r\gamma$. We also require the commitment scheme to be computationally binding; that is to say, an attacker cannot find a combination $a' + r'\gamma$ without solving the DLP for γ. From our definition of H and $\text{Com}(a, r)$ we can guarantee these properties. It is also useful to note that amount commitments use a as a blinding factor (also known as a mask), which is a random value; and r as the committed value, the reverse of what has been presented.

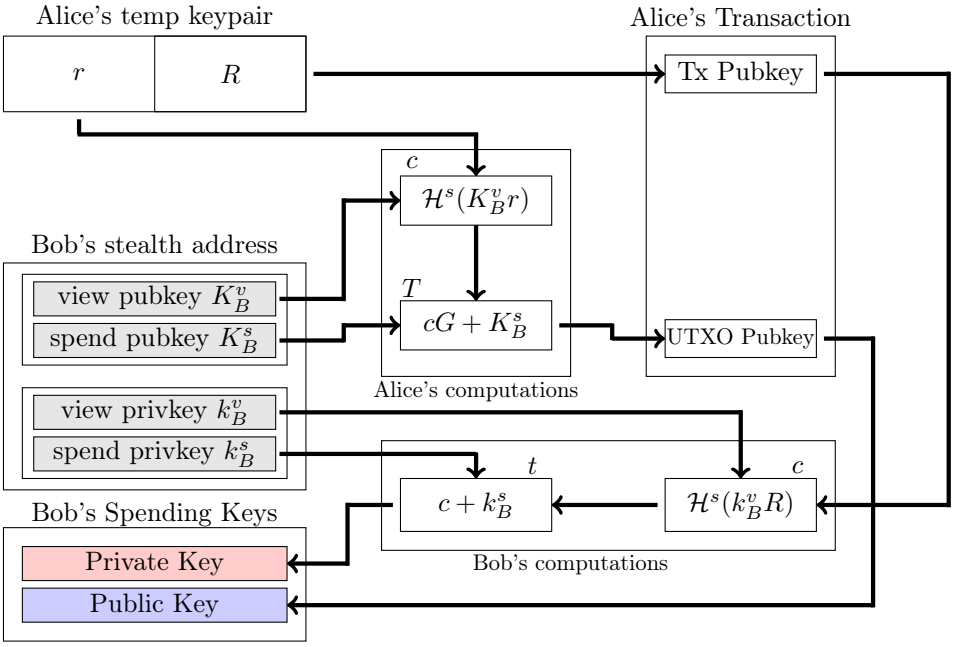

Figure 1: Dual key stealth address protocol used in private transactions

In the following sections, it is necessary to extend the definition of a commitment Com to matrices of form $A = \{a_{j,i}\}$; for $\{a_{j,i}\}, r \in \mathbb{F}_l$:

$$\text{Com}(A, r) = rH + \sum_{j,i} a_{j,i} G_{j,i} \tag{4}$$

Given our definition of indexed G in the public parameters section. We also define a commitment scheme on two matrices $A = \{a_{j,i}\}$, $B = \{b_{j,i}\}$; for $\{a_{j,i}\}, \{b_{j,i}\}, r \in \mathbb{F}_l$:

$$\text{Com}(A, B, r) = rH + \sum_{j,i} a_{j,i} G_{0,j,i} + \sum_{j,i} b_{j,i} G_{1,j,i} \tag{5}$$

Formally, a commitment in this scheme is to a value a and some randomness r, such that a two-phase cryptographic protocol is constructed. The first phase, /textitCommit, chooses randomness r and the value a and constructs a binding commitment; the second phase, /textitReveal, opens the commitment; that is, the value and randomness are revealed. A commitment to zero is one where the value $a = 0$. Proofs of knowledge to an opening to a commitment to zero are useful in the construction of privacy protocols, and will be shown in the following sections.

Amount commitments in Discreet are used as the basis for balance proofs in transactions, and are used to hide the values used as amounts in private transactions. Additionally in this section, we define equations used to construct a private output in a private transaction. First, we include a commitment to the output amount $C_* = \text{Com}(y_*, b_*)$ for a receiver to the output receiving b_* coins and masking value $y_* = \mathcal{H}^l(\text{"}commitment\_mask\text{"} \mid\mid c)$, using the definition for the Diffie-Hellman shared secret c in the previous section and "$commitment\_mask$" used for domain separation. We also define a symmetrically encrypted amount $g_* = b_* \oplus_8 \mathcal{H}(\text{"}amount\text{"} \mid\mid c)$, where $\oplus_8$ is the truncated XOR operation on the first 8 bytes of the hash. Note that a receiver simply applies this operation on g_* after recovering c to get the amount b_*,

and that domain separation is still used in the calculation of c using the index of the private output.

From this, we can define a private output Q_* as a tuple (T_*, C_*, g_*), which is 72 bytes in size. The coin protocol for balance proofs and their construction will be presented in another section.

3.2.4 Range Proofs

Due to commitments to values operating on a cyclic prime-order subgroup, one could construct a commitment to a value b_* which results in overflow, or construct two outputs with commitments to values $b_1 > 0$, $b_2 = -b_1$ to artificially create arbitrarily many spendable coins b_1. To prevent this, a proof must be constructed that all b_* lie in a valid range. This proof is known as a *range proof*.

We define the relation for a range proof as follows:

$$\Bigg\{(\{G_j\}_{j=0}^{mn-1}, \{H_j\}_{j=0}^{mn-1} \subset \mathbb{G}, G, H \in \mathbb{G}, \mathbf{V} = \{V_j\} \in \mathbb{G}^m; \mathbf{v} = \{v_j\}, \boldsymbol{\gamma} = \{\gamma_j\} \in \mathbb{F}_l^m) : \\ V_j = v_j G + \gamma_j H \wedge v_j \in [0, 2^n - 1] \text{ for } j \in [0, m)\Bigg\}$$

This gives a relation for an aggregated range proof on m output commitments $\mathbf{V}$ with masking values $\boldsymbol{\gamma}$ and values $\mathbf{v}$ within n bits in size.

Discreet makes use of Bulletproofs+[2], a succinct proving system which has a mechanism for satisfying the above relation on aggregated commitments and values. We assume a function $\text{BPProve}(\mathbf{V}, \mathbf{v}, \boldsymbol{\gamma}) \to \mathcal{B}$ implements Bulletproofs+ and satisfies the above relation. We also assume a function $\text{BPVerify}(\mathcal{B}) \to \{0, 1\}$ outputs 1 for a valid Bulletproof+ $\mathcal{B}$ and 0 otherwise. Values in Discreet will be 64 bits in size, thus $n = 64$. We choose not to present the full definition of the Bulletproofs+ protocol here for brevity and also due to their construction not being fully relevant to the techniques Discreet builds upon. Relevant details can be found in [2].

3.3 Ring Signatures

Ring signatures provide a method for obscuring the specific output being spent in a private transaction, providing stronger anonymity than using DKSAP alone. They operate as an aggregate relation demonstrating membership of a public key M_l with private key r in a set of public keys (often called the ring) $\{M_j\}_{j=0}^{N-1} \subset \mathbb{G}$, $\{M_j\}[l] = M_l$; knowledge of the private key r where $M_l = rG$; and weak linkability. Weak linkability means that no valid ring signature has been constructed using r and M_l before. This guarantees a private output cannot be spent twice. The relation, $\mathcal{R}_{\text{sig}}$, is given by:

$$\mathcal{R}_{\text{sig}} = \Bigg\{\{M_j\}_{j=0}^{N-1} \subset \mathbb{G}; r \in \mathbb{F}_l; G, J, U \in \mathbb{G}; l \in \mathbb{Z}_{\geq 0}; M_l \in \mathbb{G} : \\ 0 \leq l < N \wedge M_l \in \{M_j\} \wedge M_l = rG \wedge U = rJ\Bigg\} \tag{6}$$

The parameter J is defined to be the linking tag. Linking tags are the image of a verifiable pseudorandom function operating on the signing key r and must be constructed properly in order for the proving system under the relation $\mathcal{R}_{\text{sig}}$ to be sound. It is also used in performing weak linkability; if a valid ring signature with linking tag J' is produced using the same r, $J = J'$. Thus, part of the relation would also need to demonstrate for all valid ring signatures, all J are unique.

Typically, the proof for a ring signature relies on the prover demonstrating knowledge of an opening to a commitment to zero in a set of commitments. It is useful to note that a public key $R = aG = aG + 0H$ can be viewed as a commitment to zero

$(r = 0)$ with value a. This is beneficial to the construction of Discreet's linkable ring signature scheme, and many others.

Provided here is a formal definition for the four algorithms which comprise a linkable ring signature scheme:

- KeyGen$(r) \rightarrow (x, X)$: Generates a secret key x and public key X, optionally using randomness r if specified.
- Sign$(x, M, R) \rightarrow \sigma$: Builds signature σ on message M using a ring $R = \{X_j\}_{j=0}^{n-1}$, with some $X_i \in R$ where $xG = X_i$.
- Verify$(\sigma, M, R) \rightarrow \{0, 1\}$: Verifies signature σ on message M with respect to the ring R, outputting 0 on rejection and 1 on acceptance.
- Link$(\sigma, \sigma') \rightarrow \{0, 1\}$: Outputs 1 if σ and σ' were signatures using the same private key; and 0 otherwise.

3.3.1 Sigma Protocols

Sigma protocols were introduced in a paper by Jens Groth [3] as a means of describing common proving systems and their similarities, and as a generalization of cryptographic prover-verifier protocols. A sigma protocol is a three move interactive protocol allowing a prover to convince a verifier that a given statement is true. The prover sends an initial message to the verifier, and the verifier responds with a random challenge, to which the prover sends a response. The verifier looks at the transcript of the interaction and decides whether or not to accept or reject the proof of the statement. Sigma protocols are restricted to protocols in this manner with the following properties:

- **Completeness:** If the prover has knowledge of a witness w for the statement u, then the prover should be able to convince the verifier of said statement.
- **Special soundness:** Given that a prover does not have knowledge of a witness w for the statement, the prover cannot convince the verifier; formally, if the prover answers multiple challenges correctly from the verifier, then it is possible for a witness w to be extracted for the statement.
- **Special honest verifier zero-knowledge:** The sigma protocol does not reveal information about the prover's witness w. Formally, protocol transcripts can be simulated given any verifier challenge; i.e., for any statement and verifier challenge, a transcript can be simulated that is accepted by an honest verifier without knowledge of a corresponding witness.

The benefit of formalizing a cryptographic protocol as a sigma protocol is twofold. First, it generalizes many existing cryptographic protocols, thus allowing for connections between instances of them to be made. Secondly, they are easy to make non-interactive through the Fiat-Shamir heuristic, given a cryptographic hash function modeled as a truly random function (the random oracle model). Thus, they may describe many constructions, such as digital signature schemes and encryption schemes.

3.3.2 One-Out-of-Many Proofs

A one-out-of-many proof is a statement on a ring of commitments to which the prover knows of exactly one opening to a commitment to zero. A one-out-of-many proof can also be a ring signature, due to our relation between public-key cryptography and commitments in the previous sections. First introduced by Groth, generalized by Bootle, and modified by Sarang and Surae Noether [4], it a is a sigma protocol for the following relationships:

$$\mathcal{R}_{\text{bits}} = \left\{ (l \in \mathbb{Z}_{\geq 0}) : 0 \leq l < N \right\} \tag{7}$$

$$\mathcal{R}_{\text{link}} = \left\{ \{M_j\}_{j=0}^{N-1} \subset \mathbb{G}; r \in \mathbb{F}_l; G, J, U \in \mathbb{G}; l \in \mathbb{Z}_{\geq 0} : U = rJ \wedge M_l = rG \right\} \quad (8)$$

Given a set of $N = n^m$ commitments. A one-out-of-many proof like this has the additional property of linkability, although a one-out-of-many proof does not necessarily need to have this property.

The relation is split into two components, the first of which is the "bits" proof. $\mathcal{R}_{\text{bits}}$ is a binary proof on the signing index, which is decomposed into an n by m matrix $\{\sigma_{j,i}\}_{i=0,j=0}^{n-1,m-1}$ where $\sigma_{j,i} \equiv \delta(l_j, i)$ where δ is the Kronecker delta and l_j is defined as the jth digit of i in base n. The sigma protocol for $\mathcal{R}_{\text{bits}}$ is defined in Figure 2 and Figure 3.

$\underline{\mathcal{P}_{\text{bits}}(l):}$

- Select random $r_A \in \mathbb{F}_l$ and $\{a_{j,i}\}_{i=1,j=0}^{n-1,m-1} \subset \mathbb{F}_l$. Set

$$\{a_{j,0}\}_{j=0}^{m-1} \equiv -\sum_{i=1}^{n-1} a_{j,i}$$

- Define $\{\sigma_{j,i}\}_{i,j=0}^{n-1,m-1} \subset \mathbb{F}_l$ such that $\sigma_{j,i} \equiv \delta(l_j, i)$ and select random $r_B \in \mathbb{F}_l$.
- Define $A \equiv \text{Com}(a, -a^2, r_A)$ and $B \equiv \text{Com}(b, a(1-2b), r_B)$.

$\underline{\mathcal{P} \to \mathcal{V}:}$
A, B

$\underline{\mathcal{V} \to \mathcal{P}:}$
$\xi \in \{0,1\}^*$

$\underline{\mathcal{P}(\xi):}$

- Define $\{f_{j,i}\}_{i=1,j=0}^{n-1,m-1}$ such that $f_{j,i} \equiv a_{j,i} + \xi\sigma_{j,i}$.
- Define $z_A \equiv r_A + \xi r_B$.

$\underline{\mathcal{P} \to \mathcal{V}:}$
$\{f_{j,i}\}_{i=1,j=0}^{n-1,m-1}, z_A$

Figure 2: Sigma protocol for $\mathcal{R}_{\text{bits}}$

$\underline{\mathcal{V}_{\text{bits}}(A, B, f, z_A):}$

- For $0 \leq j < m$, let $f_{j,0} \equiv \xi - \sum_{i=1}^{n-1} f_{j,i}$.
- Accept if and only if:

$$A + \xi B = \text{Com}(f, f(\xi - f), z_A) \quad (9)$$

Figure 3: Sigma protocol for $\mathcal{R}_{\text{bits}}$, verify component

$\underline{\mathcal{P}_{\text{link}}(\{M_i\}_{i=0}^{N-1}, J, l, r):}$

- Construct $\{a_{j,i}\}_{i=0,j=0}^{n-1,m-1}$ and $\{\sigma_{j,i}\}_{i=0,j=0}^{n-1,m-1}$ in the same manner described in the first two items of $\mathcal{P}_{\text{bits}}$.
- Define coefficients $\{p_{k,j}\}_{k=0,j=0}^{N-1,m-1}$ such that

$$p_k(x) \equiv \prod_{j=0}^{m-1} (a_{j,k_j} + x\sigma_{j,k_j}) = \delta(l,k)x^m + \sum_{j=0}^{m-1} p_{k,j}x^j$$

  where k_j is taken to be the the jth digit of k in base n.
- Choose random $\{\rho_j\}_{j=0}^{m-1} \subset \mathbb{F}_l$.
- Construct $\{X_j\}_{j=0}^{m-1}$ where

$$X_j \equiv \sum_{k=0}^{N-1} p_{k,j}M_k + \rho_j G$$

- Define $\{Y_j\}_{j=0}^{m-1}$ where $Y_j \equiv \rho_j J$.

$\underline{\mathcal{P} \to \mathcal{V}:}$
$\{X_j\}, \{Y_j\}$

$\underline{\mathcal{V} \to \mathcal{P}:}$
$\xi \in \{0,1\}^*$

$\underline{\mathcal{P}(\xi):}$

- Define $\{f_{j,i}\}_{i=1,j=0}^{n-1,m-1}$ such that $f_{j,i} = a_{j,i} + \xi\sigma_{j,i}$
- Let $z \equiv r\xi^m - \sum_{j=0}^{m-1} \rho_j \xi^j$.

$\underline{\mathcal{P} \to \mathcal{V}:}$
$\{f_{j,i}\}_{i=1,j=0}^{n-1,m-1}, z$

Figure 4: Sigma protocol for $\mathcal{R}_{\text{link}}$

$\mathcal{V}_{\text{link}}(\{X_j\}, \{Y_j\}, f, z)$:

- For $0 \leq j < m$, let $f_{j,0} \equiv \xi - \sum_{i=1}^{n-1} f_{j,i}$.
- Take k_j to be the jth digit of k in base n.
- Accept if and only if:

$$\sum_{k=0}^{N-1} M_k \left(\prod_{j=0}^{m-1} f_{j,k_j} \right) - \sum_{j=0}^{m-1} \xi^j X_j - zG = 0 \tag{10}$$

$$U \sum_{k=0}^{N-1} \left(\prod_{j=0}^{m-1} f_{j,k_j} \right) - \sum_{j=0}^{m-1} \xi^j Y_j - zJ = 0 \tag{11}$$

Figure 5: Sigma protocol for $\mathcal{R}_{\text{link}}$, verify component

The construction of the sigma protocol $\mathcal{P}_{bits}, \mathcal{V}_{bits}$ is sufficient for proving the relation $\mathcal{R}_{\text{bits}}$. First, the relation can be proven as a binary proof on the deconstruction of l into σ:

$$\forall i, j : \sigma_{j,i} \in \{0, 1\} \qquad \text{and} \qquad \forall j : \sum_{i=0}^{n-1} \sigma_{j,i} = 1$$

This first constraint is satisfied when checking Equation 9, where both f is checked to be well-formed and the coefficient on ξ^2 in $f(\xi - f)$ is shown to be zero. That is, $\sigma_{j,i}(1 - \sigma_{j,i}) = 0$ if the commitments are equal, thus ensuring all $\sigma_{i,j}$ are either 0 or 1. Note that the fact that $\sigma_{j,i}^2 = \sigma_{j,i}$ is used in this equation's verification. The second constraint is satisfied if f is well-formed, which is checked entirely in the equation. Thus, all the constraints are verified by Equation 9.

The sigma protocol for $\mathcal{R}_{\text{link}}$ is used to prove membership $M_l \in \{M_j\}_{j=0}^{N-1}$, knowledge to the opening r of commitment $M_l \equiv \text{Com}(0, r)$, and linkability where $U = rJ$. The definitions for $\{f_{j,i}\}$, $\{a_{j,i}\}_{i,j=0}^{n-1,m-1}$, and σ can be reused for both $\mathcal{R}_{\text{bits}}$ and $\mathcal{R}_{\text{link}}$, which is the case for efficient implementations of one-out-of-many constructions. The validity and proof of the construction for the sigma protocol for $\mathcal{R}_{\text{link}}$ is left out for brevity, but can be checked in [4]. The definitions used in this paper are the same, save for the separation of the bits sigma protocol and the linkable one-out-of-many sigma protocol. It can be taken that both constructions are complete, special sound, and special honest verifier zero-knowledge.

3.3.3 Parallel One-Out-of-Many Construction

It is necessary to extend the one-out-of-many construction to prove knowledge of multiple openings to commitments to zero at the same index in $d > 1$ separate sets, but retaining linkability for the first set only. This is a d-linkable one-out-of-many sigma protocol which demonstrates the following relation:

$$\mathcal{R}_{\text{plink}} = \Bigg\{ \{M_{i,\alpha}\}_{i,\alpha=0}^{N-1,d-1} \subset \mathbb{G}^d, J \in \mathbb{G}, l \in \mathbb{Z}_{\geq 0}, \{r_\alpha\}_{\alpha=0}^{d-1} \subset \mathbb{F}_l : \{M_{l,\alpha} = r_\alpha G\}_{\alpha=0}^{d-1} \wedge U = r_0 J \Bigg\}$$

Only minor changes are necessary to $\mathcal{R}_{\text{link}}$ to produce a valid sigma protocol for the above relation. No changes are necessary to $\mathcal{R}_{\text{bits}}$, and as such it is not reproduced.

Note that the use of μ_α, when used in future sections, will be referred to as the μ_α trick. The additions of and modifications to protocol elements are produced solely, for the sake of brevity.

$\underline{\mathcal{P}_{\text{plink}}(\{M_{i,\alpha}\}, J, l, \{r_\alpha\}) :}$

- Let $K_\alpha \equiv r_\alpha J$ for all $\alpha \in (0, d)$.
- Define $\mu_\alpha \equiv \mathcal{H}(\alpha, \{M_{i,\alpha}\}, J, \{K_\alpha\})$ for all $\alpha \in (0, d)$.
- Construct $\{X_j\}_{j=0}^{m-1}$ where

$$X_j \equiv \sum_{k=0}^{N-1} p_{k,j} \left(M_{k,0} + \sum_{\alpha=1}^{d-1} \mu_\alpha M_{k,\alpha} \right) + \rho_j G$$

$\underline{\mathcal{P} \rightarrow \mathcal{V} :}$
$\{K_\alpha\}, \{X_j\}$

$\underline{\mathcal{V} \rightarrow \mathcal{P} :}$
$\xi \in \{0, 1\}^*$

$\underline{\mathcal{P}(\xi) :}$

- Let $z \equiv \left(r_0 + \sum_{\alpha=1}^{d-1} \mu_\alpha r_\alpha \right) \xi^m - \sum_{j=0}^{m-1} \rho_j \xi^j$.

$\underline{\mathcal{P} \rightarrow \mathcal{V} :}$
z $\underline{\mathcal{V}_{\text{plink}}(\{M_{i,\alpha}\}, J) :}$

- Define $\mu_\alpha \equiv \mathcal{H}(\alpha, \{M_{i,\alpha}\}, J, \{K_\alpha\})$ for all $\alpha \in (0, d)$.
- Accept if and only if:

$$\sum_{k=0}^{N-1} \left(M_{k,0} + \sum_{\alpha=1}^{d-1} \mu_\alpha M_{k,\alpha} \right) \left(\prod_{j=0}^{m-1} f_{j,k_j} \right) - \sum_{j=0}^{m-1} \xi^j X_j - zG = 0 \quad (12)$$

$$\left(U + \sum_{\alpha=1}^{d-1} \mu_\alpha K_\alpha \right) \sum_{k=0}^{N-1} \left(\prod_{j=0}^{m-1} f_{j,k_j} \right) - \sum_{j=0}^{m-1} \xi^j Y_j - zJ = 0 \quad (13)$$

Figure 6: Sigma protocol for $\mathcal{R}_{\text{plink}}$

3.3.4 Triptych

Triptych is the family of linkable ring signatures which prove the relations presented in the previous sections in a linkable ring signature protocol. The full sigma protocol employed by Triptych is presented in figure 7 and only differs from the original protocol presented in [4] slightly. The main modification is in $\mathcal{P}_{\text{bits}}$, where originally the isolated proof size was four group elements and two field elements (ignoring f); we instead use the modification made in [5], which halves the isolated proof size. The entirety of Triptych is reproduced in a single sigma protocol for completeness, even if it is

redundant.

3.4 Private Transaction Model

The private transaction model for Discreet is based on the Triptych linkable ring signature protocol, with $d = 2$. Before providing the transaction construction protocol, a few things must be considered. First, the Discreet private distributed ledger $L \equiv (\{P_k\}, \{\mathcal{T}_i\}, \{J_k\})$ is composed of transactions $\mathcal{T}_i$ which are previously validated, creating all P_k outputs; it is unknown to the ledger which outputs are spent due to the privacy model, but it is guaranteed that an output is only spent once in the ledger. All J_k correspond to the P_k such that an unknown one-to-one map $M : P \to J$ could be constructed if the DLP is broken. These J_k are the linking tags used in the Triptych proofs for each input in a transaction $\mathcal{T}$, and are kept in their own set for transaction verification. They, too, form a set.

Each output P, whether spent or unspent, is a tuple as defined previously in section 3.2.3. The tuple contains 2 commitments and a (potentially) encrypted amount field used in information recovery in the manner described in section 3.2.3. The two commitments are one public key T, computed via DKSAP, and a value commitment C. It is also useful to define some reference to the creating transaction $\mathcal{T}_i$ which all outputs contain.

A transaction $\mathcal{T}$ is defined as a tuple $\mathcal{T} \equiv (u_f, R, \{\mathcal{I}_i\}_{i=0}^{W-1}, \{\mathcal{O}_o\}_{o=0}^{O-1}, \{\sigma_i\}_{i=0}^{W-1}, \mathcal{B})$ where u_f is the fee, R is the transaction public key, $\mathcal{I}_i$ are the transaction input data, $\mathcal{O}_o$ are the transaction output data, σ_i are the Triptych proofs/signatures, and $\mathcal{B}$ is the Bulletproof+ range proof on the outputs. W is the number of inputs, and O is the number of outputs; both of these are constant within the transaction but can be different in another transaction. The transaction outputs are tuples where $\mathcal{O}_o \equiv (T_o, Q_o, g_o)$ with one-time destination address public key (hereon referred to as the output public key) T_o, output commitment Q_o and encrypted amount g_o. The transaction inputs are tuples as well, where $\mathcal{I}_i \equiv (\{M_{i;k,\alpha}\}_{k,\alpha=0}^{N-1,1}, P_i', J_i)$ with $\{M_{i;k,\alpha}\}_{k,\alpha=0}^{N-1,1}$ being the ring of outputs (excluding the encrypted amounts; thus $M_{i;k,0}$ is the output public key and $M_{i;k,1}$ is the output commitment). P_i' is known as the pseudo-output, and is constructed in a particular manner described in the transaction creation protocol. J_i is the linking tag corresponding to some output public key $M_{i;l,0} = r_{i;l,0}G$, $J_i \equiv r_{i;l,0}^{-1}U$.

Here, we present the manner which a user can construct a valid transaction $\mathcal{T}$ spending their W outputs $P_i = (T_i, C_i, g_i, \mathcal{H}(\mathcal{T}))$, where $\mathcal{H}(\mathcal{T})$ is a reference to the creating transaction, and creating O fresh outputs O_o to destinations $\{(K_{*;o}^v, K_{*;o}^s)\}_{o=0}^{O-1}$. It is assumed that if needed, a change output is included implicitly. Note the reference to the creating transaction can be used to look up information from that transaction. The user's own address is (K_u^v, K_u^s) with corresponding private keys (k_u^v, k_u^s).

TxRecover$(\{P_i\}_{i=0}^{W-1}) \to (\{b_i\}, \{y_i\}, \{t_i\})$:

- For all outputs to be spent $P_i = (T_i, C_i, g_i, \mathcal{H}(\mathcal{T}))$, recover each of their source transaction's public key $R_{t;i}$ using the source transaction reference $\mathcal{H}(\mathcal{T})$ in each output and some oracle, which can be implemented as a local key-value database.
- First calculate the shared secret $c_i \equiv \mathcal{H}^l(k_u^r R_{t;i})$, then recover decrypted amounts $b_i \equiv g_i \oplus_8 \mathcal{H}(\text{"}amount\text{"} \mid\mid c_i)\ \forall i \in [0, W)$.
- Verify that $C_i = \text{Com}(\mathcal{H}(\text{"}commitment\_mask\text{"} \mid\mid c_i), b_i)\ \forall i \in [0, W)$.
- Recover output private keys $t_i \equiv c_i + k_u^s$ for all outputs to be spent; make sure to check $T_i = t_i G$.

TxCreate$(\{P_i\}_{i=0}^{W-1}, \{b_i\}) \to \mathcal{U}_{tx}$:

$\underline{\mathcal{P}_{\text{triptych}}(\{M_{i,\alpha}\}, J, l, \{r_\alpha\}) :}$

- Select random $r_A \in \mathbb{F}_l$ and $\{a_{j,i}\}_{i=1,j=0}^{n-1,m-1} \subset \mathbb{F}_l$. Set

$$\{a_{j,0}\}_{j=0}^{m-1} \equiv -\sum_{i=1}^{n-1} a_{j,i}$$

- Define $\{\sigma_{j,i}\}_{i,j=0}^{n-1,m-1} \subset \mathbb{F}_l$ such that $\sigma_{j,i} \equiv \delta(l_j, i)$ and select random $r_B \in \mathbb{F}_l$.
- Define $A \equiv \text{Com}(a, -a^2, r_A)$ and $B \equiv \text{Com}(b, a(1-2b), r_B)$.
- Let $K_\alpha \equiv r_\alpha J$ for all $\alpha \in (0, d)$.
- Define $\mu_\alpha \equiv \mathcal{H}(\alpha, \{M_{i,\alpha}\}, J, \{K_\alpha\})$ for all $\alpha \in (0, d)$.
- Define coefficients $\{p_{k,j}\}_{k=0,j=0}^{N-1,m-1}$ such that

$$p_k(x) \equiv \prod_{j=0}^{m-1} (a_{j,k_j} + x\sigma_{j,k_j}) = \delta(l,k)x^m + \sum_{j=0}^{m-1} p_{k,j}x^j$$

  where k_j is taken to be the the jth digit of k in base n.

- Choose random $\{\rho_j\}_{j=0}^{m-1} \subset \mathbb{F}_l$.
- Construct $\{X_j\}_{j=0}^{m-1}$ where

$$X_j \equiv \sum_{k=0}^{N-1} p_{k,j} \left(M_{k,0} + \sum_{\alpha=1}^{d-1} \mu_\alpha M_{k,\alpha} \right) + \rho_j G$$

- Define $\{Y_j\}_{j=0}^{m-1}$ where $Y_j \equiv \rho_j J$.

$\underline{\mathcal{P} \to \mathcal{V} :}$
$\{K_\alpha\}, A, B, \{X_j\}, \{Y_j\}$

$\underline{\mathcal{V} \to \mathcal{P} :}$
$\xi \in \{0,1\}^*$

$\underline{\mathcal{P}(\xi) :}$

- Define $\{f_{j,i}\}_{i=1,j=0}^{n-1,m-1}$ such that $f_{j,i} \equiv a_{j,i} + \xi\sigma_{j,i}$.
- Define $z_A \equiv r_A + \xi r_B$.
- Let $z \equiv \left(r_0 + \sum_{\alpha=1}^{d-1} \mu_\alpha r_\alpha\right) \xi^m - \sum_{j=0}^{m-1} \rho_j \xi^j$.

$\underline{\mathcal{P} \to \mathcal{V} :}$
$\{f_{j,i}\}_{i=1,j=0}^{n-1,m-1}, z_A, z$

Figure 7: Sigma protocol for Triptych

$\mathcal{V}_{\text{triptych}}(\{M_{i,\alpha}\}, J)$:

- For $0 \leq j < m$, let $f_{j,0} \equiv \xi - \sum_{i=1}^{n-1} f_{j,i}$.
- Define $\mu_\alpha \equiv \mathcal{H}(\alpha, \{M_{i,\alpha}\}, J, \{K_\alpha\})$ for all $\alpha \in (0, d)$.
- Accept if and only if:

$$A + \xi B = \text{Com}(f, f(\xi - f), z_A) \tag{14}$$

$$\sum_{k=0}^{N-1} \left(M_{k,0} + \sum_{\alpha=1}^{d-1} \mu_\alpha M_{k,\alpha} \right) \left(\prod_{j=0}^{m-1} f_{j,k_j} \right) - \sum_{j=0}^{m-1} \xi^j X_j - zG = 0 \tag{15}$$

$$\left(U + \sum_{\alpha=1}^{d-1} \mu_\alpha K_\alpha \right) \sum_{k=0}^{N-1} \left(\prod_{j=0}^{m-1} f_{j,k_j} \right) - \sum_{j=0}^{m-1} \xi^j Y_j - zJ = 0 \tag{16}$$

Figure 8: Sigma protocol for Triptych, verify component

KeyGen(r) :

- If unspecified, select randomly $r \in \mathbb{F}_l$ and return $(x, X) = (r, rG)$.

Sign(x, M, R) :

- Let $R = \{X_{j,\alpha}\}_{j,\alpha=0}^{N-1,d-1}$, $x = \{x_\alpha\}_{\alpha=0}^{d-1}$ such that $X_{l,\alpha} = x_\alpha G$ for all $\alpha \in [0, d)$.
- Compute $J \equiv x_{l,0}^{-1} U$.
- Run $\mathcal{P}_{\text{triptych}}(R, J, l, x_{l,0}) \to a$ (up to verifier challenge).
- Set $\xi \equiv \mathcal{H}(M, R, a)$.
- Run $\mathcal{P}_{\text{triptych}}(\xi) \to z$ (after verifier challenge).
- Return $\sigma = (a, z, J)$.

Verify(σ, M, R) :

- Let $R = \{X_{j,\alpha}\}_{j,\alpha=0}^{N-1,d-1}$.
- Let $\sigma = (a, z, J)$ and set $\xi \equiv \mathcal{H}(M, R, a)$.
- Return $\mathcal{V}_{\text{triptych}}(R, J, a, z)$.

Link(σ, σ') :

- Assuming σ and σ' have been previously verified, extract J from σ and J' from σ'.
- Return 1 if $J = J'$ and 0 otherwise.

Figure 9: Linkable ring signature for Triptych

- Construct a transaction key pair $\text{KeyGen}(*) \rightarrow (r_t, R_t)$ where R_t is the transaction public key.
- Calculate the transaction fee u_f using some fee calculation function, assumed to be consistent and based on known information up to this point.
- Choose, up to the user, O amounts b_o $\forall o \in [0, O)$ such that the sum of these amounts plus the transaction fee u_f equals the sum of decrypted amounts b_i in the inputs.
- Run DKSAP on each address $(K^v_{*;o}, K^s_{*;o})$ $\forall o \in [0, O)$ to obtain corresponding output public keys T_o; extract the Diffie-Hellman shared secrets c_o from each DKSAP run.
- Calculate output masking values $y_o \equiv \mathcal{H}^l(\text{"}commitment\_mask\text{"} \mid\mid c_o)$ from each c_o, and compute the encrypted output amounts $g_o \equiv b_o \oplus_8 \mathcal{H}(\text{"}amount\text{"} \mid\mid c_i)$.
- Compute output commitments $C_o \equiv \text{Com}(y_o, b_o)$. Define $O_o \equiv (T_o, C_o, g_o)$.
- Choose random masks $y'_i \in \mathbb{F}_l$ $\forall i \in [1, W)$, and compute y'_0 as:

$$y'_0 \equiv \sum_{o=0}^{O-1} y_o - \sum_{i=1}^{W-1} y'_i$$

- Compute pseudo-outputs $P'_i \equiv \text{Com}(y'_i, b_i)$ $\forall i \in [0, W)$.
- Create a Bulletproof+ range proof $\mathcal{B} \equiv \text{BPProve}(\{C_o\}, \{b_o\}, \{y_o\})$.
- Finally, for each $i \in [0, W)$, select $N-1$ outputs $\{P_{i;k}\}_{k=0}^{N-2}$ from the ledger L according to some algorithm Π_{DSA}. Insert P_i at some randomly chosen index l_i. Extract their output public keys $\{T_{i;k}\}_{k=0}^{N-1}$ and commitments $\{C_{i;k}\}_{k=0}^{N-1}$. Construct $\{M_{i;k,\alpha}\}_{k,\alpha=0}^{N-1}$ where $M_{i;k,0} \equiv T_{i;k}$ and $M_{i;k,1} \equiv C_{i;k}$.
- Construct the unsigned transaction $\mathcal{U}_{tx} \equiv (u_f, R_t, \{M_{i;k,\alpha}\}, \{P'_i\}, \{O_o\}, \mathcal{B})$.
- Return $\mathcal{U}_{tx}$.

$\underline{\text{TxSign}(\mathcal{U}_{tx}, \{y_i\}, \{t_i\}, \{y'_i\}, \{b_i\}, \{b_o\}) \rightarrow \mathcal{T}:}$

- Unpack $\mathcal{U}_{tx} = (u_f, R_t, \{M_{i;k,\alpha}\}, \{P'_i\}, \{O_o\}, \mathcal{B})$.
- Unpack $\{O_o\} = \{(T_o, C_o, g_o)\}$.
- Compute linking tags $J_i \equiv t_i^{-1} U$.
- Compute message $m \equiv \mathcal{H}(\{R_t, M_{i;k,\alpha}\}, \{J_i\}, \{g_i\}, \{g_o\}, \{T_o\})$. (In practice this may have additional data to ensure no malleability.)
- Define, for all $i \in [0, W)$, a set $\{R_{i;k,\alpha}\}_{i,\alpha=0}^{N-1,1}$ such that $R_{i;k,0} \equiv M_{i;k,0}$ and $R_{i;k,1} \equiv M_{i;k,1} - P'_i$.
- Define, for all $i \in [0, W)$, a set $\{r_{i;\alpha}\}_{\alpha=0}^{1}$ such that $r_{i;0} \equiv t_i$ and $r_{i;1} \equiv y_i - y'_i$.
- Calculate Triptych linkable ring signatures $\sigma_i \equiv \text{Sign}(\{r_{i;\alpha}\}_{\alpha=0}^{1}, m, \{R_{i;k,\alpha}\}_{i,\alpha=0}^{N-1,1})$ for all $i \in [0, W)$.
- Construct transaction inputs $\mathcal{I}_i \equiv (\{M_{i;k,\alpha}\}, P'_i, J_i)$ for all $i \in [0, W)$.
- Build the transaction $\mathcal{T} \equiv (u_f, R_t, \{\mathcal{I}_i\}, \{O_o\}, \{\sigma_i\}, \mathcal{B})$.
- Return $\mathcal{T}$.

$\underline{\text{TxVerify}(\mathcal{T}) \rightarrow \{0, 1\}:}$

- If any items in the transaction are malformed, return 0.

- If the equation
$$\sum_{i=0}^{W-1} P'_i = \text{Com}(0, u_f) + \sum_{o=0}^{O-1} C_o$$
does not hold, return 0.
- Recompute the message m as defined in TxCreate. Additionally, recompute $\{R_{i;k,\alpha}\}$ as defined in TxCreate for all $i \in [0, W)$.
- For all $i \in [0, W)$, run Verify$(\sigma_i, m, \{R_{i;k,\alpha}\})$. If any return 0, return 0.
- Run BPVerify$(\mathcal{B})$ and return 0 if it returns 0.
- For all previous transactions in the ledger L, extract all Triptych signatures in each transaction in the ledger to form the set $\{\sigma_k\}$. For all $i \in [0, W)$ and all σ_k, if Link$(\sigma_i, \sigma_k) = 1$, return 0. Note that this step can be efficiently implemented using a computationally efficient table of all linking tags J_k in the ledger and checking if the linking tags J_i from each input are already present.
- Return 1.

The algorithm which decides how the ring is constructed for each ring signature is known as the decoy selection algorithm, and is presented in these algorithms as Π_{DSA}. It can be constructed using statistical information about the nature of how outputs are spent, and must be built in a way where the probability of an attacker guessing the position of the output being spent in the ring is $1/|R| + \eta(\lambda)$ with $|R|$ being the size of the ring and $\eta(\lambda)$ being a negligible error term. Essentially, the construction of the ring must appear completely random in its choice of decoys such that the attacker cannot gain any advantage from the algorithm alone. Constructions of Π_{DSA} are an active topic and very specific to ledger implementations, and as such no specific one is provided in this section. Further reading can be found in [6]. As of the time of writing, proper construction of DSA has become even more important. We hope to discuss more about this in a planned paper detailing potential upgrades to the ledger in the future.

Note that TxRecover, TxCreate and TxSign are designed to be run together as they depend on information generated in each algorithm. They've been semantically separated in their presentation only for ease of reading.

4 Transparency and Discreet

Discreet provides both a private and public distributed ledger. This section defines the transparent coin protocol of Discreet, and the bridge between the two coin protocols in the form of mixed transactions.

4.1 Transparent Transactions

Transparent transactions, or public transactions, are value transfers on Discreet where the amount field, as well as the destination address, are not obscured by the coin protocol itself. This is required due to the regulatory restrictions exchanges and other entities require for cryptocurrency transactions on their platforms. However, this protocol is separate from the private coin protocol, and thus privacy guarantees are not weakened simply from allowing both coin protocols.

4.1.1 Definitions

The transparent coin protocol's public key cryptography uses Ed25519 for the elliptic curve and EdDSA as the signature algorithm. The hash function is SHA256. We will assume for the sake of abstraction EdDSA is equipped with definitions Sign$(k, K, M) \rightarrow$

σ and $\text{Verify}(\sigma, M) \rightarrow \{0,1\}$ for a message M, key pair (k, K) and signature σ. σ is also assumed to store the public key K as well.

The protocol also defines a public address as RIPEMD160(SHA256(K)). Addresses are encoded using Base58 and may have additional data, such as version bytes and checksums; these are not included in the protocol specification for clarity.

In the actual protocol, inputs to a transparent transaction are encoded by taking the ID of the transaction the output being spent was created in (in the form of a SHA256 hash of the transaction) and appending a byte representing the offset of said output in the transaction's outputs. This is to save on space in the ledger, and it is assumed the transparent ledger can easily retrieve the full output data from this input.

4.1.2 Protocol Specification

A public transaction $\mathcal{T}$ with W inputs and O outputs is defined as a tuple $\mathcal{T} \equiv (u_f, \{\mathcal{I}_i\}_{i=0}^{W-1}, \{\mathcal{O}_o\}_{o=0}^{O-1}, \{\sigma_{\text{EdDSA};i}\}_{i=0}^{W-1})$ with u_f being the fee and $\sigma_{\text{EdDSA};i}$ input signatures. Prior to signing, a hash of the inputs and outputs, as well as the fee and any transaction header information, is made, and is known as the inner hash H_{inner}. Signatures are performed on $\text{SHA256}(H_{\text{inner}}||\text{SHA256}(\mathcal{I}_i))$, a hash concatenation of the inner hash and the hash of the input corresponding to the owner performing their authorization of spend via signature. This construct is to ensure uniqueness in the signing hash for each input and to prevent known malleability-based attacks.

A transparent transaction's individual inputs are represented by the tuple $\mathcal{I}_i \equiv (H_{\text{TxSrc}}, A_i, u_{\text{amt}})$, with H_{TxSrc} being the SHA256 hash of the transaction which created the output, A_i being the address which owns the output being spent, and u_{amt} being the denomination, i.e. the amount of droplets the output contains. Similarly, an output $\mathcal{O}_o \equiv (A_o, u_{\text{amt}})$ with the members of the tuple having the same definitions but respective to the output. The protocol for transparent transactions, then, is as follows:

$\underline{\text{TxCreate}(\{\mathcal{I}_i\}_{i=0}^{W-1}, \{A_o\}_{o=0}^{O-1}, \{u_o\}_{o=0}^{O-1}) \rightarrow \mathcal{U}_{tx}:}$

- Recover the W individual inputs $\mathcal{I}_i$ as needed, as well as their associated keypairs (k_i, K_i) and construct their set.
- Calculate the transaction fee u_f using some fee calculation function, assumed to be consistent and based on known information up to this point.
- Choose, up to the user, O addresses and amounts to send funds to (represented in the inputs as $\{A_o\}$ and $\{u_o\}$) such that the sum of the funds to send plus the fee calculated in the step above equals the total number of coins in the inputs being spent.
- Construct the outputs $\mathcal{O}_o \equiv (A_o, u_o)$ for all $o \in [0, O)$.
- Construct the unsigned transaction $\mathcal{U}_{tx} \equiv (u_f, \{\mathcal{I}_i\}, \{\mathcal{O}_o\})$.
- Return $\mathcal{U}_{tx}$.

$\underline{\text{TxSign}(\mathcal{U}_{tx}, \{(k_i, K_i)\}_{i=0}^{W-1}) \rightarrow \mathcal{T}:}$

- Calculate the inner hash $H_{\text{inner}} \equiv \text{SHA256}(\mathcal{U}_{tx})$ (note: inner hash in practice may be calculated with header information, but for clarity this information is not included).
- Unpack $\mathcal{U}_{tx} = (u_f, \{\mathcal{I}_i\}, \{\mathcal{O}_o\})$.
- Calculate the signing hashes $H_{\text{sign};i} \equiv \text{SHA256}(H_{\text{inner}}||\text{SHA256}(\mathcal{I}_i))$ for all $i \in [0, W)$.
- Calculate the signatures $\sigma_{\text{EdDSA};i} \equiv \text{Sign}(k_i, K_i, H_{\text{sign};i})$ for all $i \in [0, W)$.

- Construct the transaction $\mathcal{T} \equiv (u_f, \{\mathcal{I}_i\}, \{\mathcal{O}_o\}, \{\sigma_{\text{EdDSA};i}\})$.
- Return $\mathcal{T}$.

TxVerify($\mathcal{T}$) $\rightarrow \{0, 1\}$:

- If any items in the transaction are malformed, return 0.
- If any input in the transaction has already been spent, return 0.
- Unpack the input amounts $u_i \forall i \in [0, W)$ and output amounts $u_o \forall o \in [0, O)$, as well as the transaction fee u_f.
- If the equation
$$\sum_{i=0}^{W-1} u_i = u_f + \sum_{o=0}^{O-1} u_o$$
does not hold, return 0.
- Recompute the inner hash H_{inner} and all $H_{\text{sign};i} \forall i \in [0, W)$ as specified in TxSign.
- For all $i \in [0, W)$, run Verify($\sigma_{\text{EdDSA};i}, H_{\text{sign};i}$). If any return 0, return 0.
- Unpack the destination addresses A_i from the inputs and all K_i from the signatures.
- Calculate $A'_i \equiv \text{RIPEMD160}(\text{SHA256}(K_i))$.
- For all $i \in [0, W)$, if any equality $A_i = A'_i$ does not hold, return 0.
- Return 1.

4.2 Mixed Transactions

Discreet utilizes a novel transaction framework allowing native transactions between transparent and private addresses without the need for a bridge. Thus, the protocol is equipped with a special type of transaction, known as a mixed transaction, which removes the need for a service provider to exist within or on top of the protocol.

This transaction can be defined as:

$$\mathcal{T} \equiv (u_f, \{\mathcal{I}_{t;i}\}_{i=0}^{W_t-1}, \{\mathcal{O}_{t;o}\}_{o=0}^{O_t-1}, \{\sigma_{\text{EdDSA};i}\}_{i=0}^{W_t-1}, R, \{\mathcal{I}_{p;i}\}_{i=0}^{W_p-1}, \{\mathcal{O}_{p;o}\}_{o=0}^{O_p-1}, \{\sigma_i\}_{i=0}^{W_p-1}, \mathcal{B})$$

Note that this simply reads as a combination of the fields in a private and transparent transaction. As such, a mixed transaction can be thought of as, when separating the fee parameter u_f from the underlying transaction:

$$\mathcal{T} \equiv (\mathcal{T}_p, \mathcal{T}_t, u_f)$$

The verification logic, as well as the creation logic, are the main points of difference when constructing a mixed transaction. First, when a mixed transaction contains transparent inputs, the amounts in the private outputs must be transparent as well. This can be accomplished by treating those outputs the same as coinbase outputs; i.e. the mask for their commitment would be 1 instead of a calculated y. This is to ensure no funds are created out of thin air through careful manipulation of transaction parameters, and also to ensure correctness. Note that by utilizing a separate mixin pool for these types of outputs anonymity is preserved long-term; thus, implementations of the mixed transaction protocol should pay attention to such solutions. Additionally, one can form a commitment with a zero mask (as is done with the fee parameter in private transactions) for both the sum of inputs and sum of outputs for transparent members during validation. These are the main changes which must be made for mixed transactions to work. The full logic is not reproduced for these transactions in this section for the sake of brevity; it should be easy enough to see their logic through combination of the protocols specified above and with the changes stated in this paragraph.

5 Consensus

Discreet provides a means of achieving consensus by a novel combination of confidential proof-of-stake and Asynchronous Byzantine Fault Tolerant (ABFT) systems called Aurem. The confidential proof-of-stake protocol allows a subset of validators to be picked to serve on a committee responsible for achieving finality, for a given epoch. The actual mechanism for achieving consensus is performed with an asynchronous byzantine fault tolerant mechanism, which guarantees liveness, total ordering, and agreement, with appropriate resilience to censorship. The ABFT mechanism currently chosen is based on AlephBFT [7], since it achieves optimal latency and communication complexity while fitting to our throughput and transaction latency requirements.

5.1 Consensus Definitions

While ABFT consensus has many desirable properties, in order to achieve optimal scalability there are restrictions on how many users can operate during a given epoch as members. Additionally, since the set of consensus-level validators is dynamic (i.e., consensus validators which are viable candidates may join and leave the network), having a dynamically-chosen subset of validators is necessary in our protocol. This leads to two design choices.

First, a given committee at the consensus layer is static and responsible for running the finality mechanism for a given number of rounds, known as an epoch. During this time, new blocks are minted and transactions are settled on the network via the consensus mechanism. Necessarily, this committee is chosen prior to the epoch they will run for.

Second, a fair means of selecting committee members from a pool of candidate validators must be specified. In Aurem, this consists of a validator submitting a proof of candidacy, weighted fairly by a stake in $DIST and in such a way that the staked amount is confidential.

In Bitcoin, these two points are one and the same: any node can serve as a validator and a consensus node simply by minting a new valid block at the next height. However, the consensus mechanism cannot achieve finality instantly; instead, consensus is asymptotically achieved, and a total ordering is eventually valid. In practice many users of the Bitcoin network set a threshold number of blocks which must be minted after the one containing a transaction to consider said transaction as "settled". We want to have a mechanism which provides *instant finality*, which is to say, after a given constant number of consensus rounds, we can consider a transaction as fully settled.

Formally, consensus is divided into *epochs* and *rounds*. A round is a single duration of nodes reliably minting and broadcasting blocks at the same height. It is synonymous with block height, but refers to the topological height within the DAG. An epoch is a set of rounds such that epoch $\geq c$ for c being a sufficiently sized parameter. In consensus, prior to the next epoch, a set of nodes submit a *bid for candidacy* marked by a proof of validity, which for Aurem is a confidential proof-of-stake demonstrating ownership of a sufficient amount of $DIST in a staked output. The proof of validity also contains extra information verified in zero-knowledge necessary for fair selection.

5.2 Choosing Committee Members

In order to choose committee members, a fair means of selection which can occur in a decentralized network must exist. The heart of this method for Discreet is in the confidential proof-of-stake mechanism.

First, the staking layer must have a structure to guarantee that stakes are anonymous in the transaction layer, as well as confidential in all layers of the network pro-

tocol. This can be achieved through the use of a dense merkle tree of stakes. A node creates a special type of transaction called a *Stake Transaction*, which commits a value to the stake merkle tree, but does not reveal the location in the tree or the amount staked via a zero-knowledge proof of correctness for the calculations and a commitment to the value. Later on, a node can submit an *Unstake Transaction*, which produces a nullifier (a kind of opening to the committed value in the stake merkle tree), which is stored on a sparse merkle tree. This protocol forms the basis of confidential proof-of-stake.

In order for committee members to be selected, an additional transaction called *StakeVote* is used. This is a zero-knowledge proof where the creator attests to a stake it created by proving (in zero-knowledge) a valid opening exists owned by itself, as well as proof that the stake was computed correctly. Additional checks for the specific implementation of proof-of-stake can be included additionally in the zero-knowlege proof. *StakeVote* must also contain a method for producing a manipulation-free *score* weighted only by the staked amount. The score is used to determine if a node will be present in the next epoch as a committee member, and is computed such that (1) no node can influence their score outside of the staked amount; (2) the staked amount proportionally increases the probability of a score leading to selection. Such a zero-knowledge proof is possible via succinct noninteractive arguments of knowlede, or zkSNARKs. These can use any proving system which fits the standards of Discreet and as this specific area of research is active, no commitments to any specific proving system have been made, while many have been shown to work in private testing.

When implemented, *StakeVote* will make use of additional parameters to ensure sufficient randomness in the probability distribution. Additional details on the specifics of the stake and unstake transactions, as well as full descriptions of the proofs, will be shared as the details are finalized. These will be published in a separate document and made available through our website.

5.3 ABFT

As mentioned previously, Discreet's finality layer can use any ABFT protocol. A variant of AlephBFT was chosen due to the optimal communication complexity and the guarantees from its use of reliable broadcast. It also provides sufficient finalization times and throughput, and from tests has seen 89600 transactions per second[8] with latency less than a second. Note that these numbers are seen as a "best case" as in practice network usage isn't constant and uniform, and over time the likelihood of byzantine behavior will cause several potential issues. Nonetheless it is more than enough to meet the needs of a global payment system.

The following sections will outline the core features of the protocol. For a full overview, we recommend reading the source paper.

5.3.1 Broadcast

The protocol consists of a committee of validators which form the consensus network, and operate by aggregating communication on the transaction layer into a total ordering of units on the consensus layer. These units are added by each node into a local copy of the communication history, in an object called the communication history DAG (ch-DAG). Such a structure stores all received communication (i.e. blocks, or units) and guarantees that (1) for all honest nodes, the set of units in the ch-DAG form a valid chain; (2) all units at round r has at least $2f+1$ parents at round $r-1$, for all rounds; and (3) all units have parents created by different nodes (i.e., each parent unit does not share the same creator). A node creates a new unit for round r when it collects enough at least $2f+1$ units from round $r-1$ from other nodes. These, as well as the units which maximize the round they were created in for all other nodes not in

the previous set, are the parents of said node's unit. Unit-specific data is added from the transaction layer, as well as a signature and an index, and the unit is propagated via reliable broadcast.

Reliable broadcast is an asynchronous communication protocol where (1) incorrect units cannot be broadcasted successfully and (2) every unit broadcasted by an honest node is eventually received. This is enough to guarantee the ch-DAG is reliable, ever-expanding and fork-free. In practice, this protocol is coupled with an alert system which can quickly tell the network if an invalid fork is created; this helps to prevent fork bomb attacks, improve security and speed up the communication. Additionally, reliable broadcast guarantees the ch-DAGs of all nodes can be kept in sync. Reliable broadcast will also define a set of communication keys for each node k $\{pk_{k\to i}\}_{i=0}^{N}$ which allow for primitives $Enc_{k\to i}(\{0,1\}^*) \to$ cipher and $Dec_{k\to i}(\text{cipher}) \to \{0,1\}^*$ to be constructed for encrypting a plaintext (represented as a binary string, implicitly) to a ciphertext to be exchanged from node k to node i, and vice-versa.

Nodes then perform the consensus protocol by deciding whether a sufficient number of units are visible in the ch-DAG, then deciding on a common vote. The common vote is randomized through a distributed key generation (DKG) scheme called *Randomness Beacon*, which provides a trustless means of creating a common source of randomness. Prior to the vote, a random permutation of units at the given round is also created to provide a means of choosing the head unit in the ch-DAG. This is required due to the non-triviality of such a problem; in an asynchronous setting, there is no simple method of guaranteeing consistency across all ch-DAGs that the head unit is the same for all honest nodes. The details on why this is so correspond to the FLP impossibility.

5.3.2 Randomness Beacon

Randomness beacon is a means of constructing a trustless source of randomness in ABFT. It is used to resolve calls to the primitive SecretBits(i, r) necessary for both choosing a head unit for the topological layer within the ch-DAG and generating a permutation of units. The protocol itself is inspired by threshold signatures, which make use of the properties of pairing-based cryptography (PBC).

In a PBC scheme using elliptic curves, an underlying finite field $\mathbb{Z}_p$ of prime p elements defines an elliptic curve G_1 of prime r elements. An embedding degree k is chosen such that $p^k - 1$ is a multiple of r, and as such $\mathbb{Z}_{p^k}$ can be represented as a field of the same characteristic as $\mathbb{Z}_p$. A second elliptic curve G_2 over this field can be generated such that one can define an isomorphism between G_1 and G_2 (this is due to the fact that the r-th roots of unity are contained in $\mathbb{Z}_{p^k}$ and due to the r-torsion group is contained in G_2) [9]. Additionally a "pairing" between the curves $e : G_1 \times G_2 \to G_T$ can be constructed which is:

- **Non-degenerate:** $e(P, Q) \neq 1$,
- **Bilinear:** $\forall a, b \in \mathbb{Z}_p^*, P \in G_1, Q \in G_2, : e(aP, bQ) = e(P, Q)^{ab}$,
- **Computable:** An efficient algorithm can be constructed to compute e.

G_T is another group of order r over $\mathbb{Z}_{p^k}$ related between the first curves with e. In practice, G_2 can be efficiently reduced via a twist with degree d dividing k if constructed properly, and this is done in Barreto-Naehrig curves using a sectic twist $E'(\mathbb{Z}_p^2)$ for $\xi \in \mathbb{Z}_{p^2}$, $W^6 - \xi$ irreducible over $\mathbb{Z}_{p^2}[W]$ for $p \equiv 1(\text{mod}6)$. More details can be found in [9]. Assume g_1 is a chosen, agreed-upon generator for G_1, g_2 for G_2, and $g_t = e(g_1, g_2)$.

Regarding randomness beacons, nodes serve as key dealers as in traditional threshold signature schemes, without picking a specific node to serve as the trusted dealer. This is achieved using key boxes, and are constructed for a node k as follows.

- Sample a random polynomial of degree f

$$A_k(x) = \sum_{j=0}^{f} a_{k,j} x^j \in \mathbb{Z}_p[x]$$

- Compute a commitment to A_k

$$C_k = (g_1^{a_{k,0}}, ..., g_1^{a_{k,f}})$$

- Define tossing keys TK_k and verification keys VK_k

$$TK_k \equiv (tk_{k,1}, ..., tk_{k,N}) = A_k(i) \forall i \in 1..N$$

$$VK_k \equiv (vk_{k,1}, ..., vk_{k,N}) = g_1^(tk_{k,i}) \forall i \in 1..N$$

  (Note that the verification key can be reconstructed by $vk_{k,i} = \prod j = 0^f C_{k,j}^{ij}$).

- Encrypt tossing keys using dedicated public keys from reliable broadcast $pk_{k \to i}$:

$$e_{k,i} \equiv \text{Enc}_{k \to i}(tk_{k,i})$$

  Let $E_k \equiv (e_{k,1}, ..., e_{k,N}$.

- The key box for k is $KB_k = (C_k, E_k)$.

The key set for k, KS_k, is (VK_k, TK_k). VK_k can be reconstructed from KB_k, and individual tossing keys for each node can be decrypted from E_k, but only for that specific node. Nodes vote on correctness of the key box by emitting 1 if the decrypted $tk_{k,i}$ satisfies $g_1^{tk_{k,i}} = vk_{k,i}$, and $\text{Dec}_{k \to i}(e_{k,i})$ otherwise; thus nodes cannot falsify these if node k is an honest dealer. This vote occurs at round 3 and is performed for all KB_k in its ch-DAG. Nodes also include their shares $m^{tk_{k,i}}$ for all valid key boxes received in their units as these KB_k become available through communication, for a chosen nonce m, for shares $\S_{m,k,i} \equiv m^{A_k(i)} = m^{tk_{k,i}}$, for m being mappable to an element in G_2. These shares are reconstructible into a random secret $\sigma_{m,k} \equiv m^{A_k(0)}$ through lagrange interpolation.

At round 6, each node includes in their unit a set $T_i \subseteq [N]$ where each element is part of the set only if (1) $U[k;0] \leq V$ (that is, the topological height of the unit from node k at round 0 is less than this of the current node's new unit); (2) for all $j \in [N]$ such that $U[j;3] \leq V$ the votes for KB_k from j are 1.

Nodes can now combine tosses at round $r \geq 9$ and for nodes that have $U[i;6]$ in their ch-DAG. First, the threshold signature produced is defined by:

$$\tau_{m,i} \equiv \prod k \in T_i \sigma_{m,k} = m^{\Sigma_{k \in T_i} A_k(0)} \in G_2.$$

This can be hashed and used to produce an i-th source of randomness called MultiCoin_i. Nodes produce a nonce $m \equiv "i||r"$ and request from other nodes to include their shares for the nonce m for each key set KS_j for $j \in T_i$ between honest nodes which voted KB_j was correct in round 3. At round $r+1$ nodes can produce the $\text{SecretBits}(i,r)$ from the units at round r in their ch-DAG and their shares to form MultiCoin_i. This is sufficient to produce a randomness beacon protocol useful for our variant of AlephBFT.

The actual implementation of this protocol will use many of the variants discussed in [7]'s appendices to gain as much speed as possible. A full detailed description of the full finality protocol is found in said source as well. Note that the protocol above was chosen due to the guarantee of asynchronous liveness and due to the minimal communication complexity for both setup and query. Any protocol which fits our desired target values for throughput and latency while also guaranteeing consistency, total ordering, resistance to censorship and liveness in an asynchronous setting can be used.

6 Future Work

This represents the first of many publications specifying details about the Discreet ledger and its capabilities. For each of the main topics set forth in this paper, a subsequent paper will be released, giving insight into how these will be implemented with full explanations of their components. Two of these we plan to touch on further are programmability and native multi-asset; initially planned to be part of this specification, we came to the conclusion these topics would be better served in publications of their own in the near future. The token economy for $DIST will be touched on in a separate publication to be available shortly. This will detail tail emissions, coinbase amount, schedules for minting, and token emissions. Additionally, regarding the confidential proof-of-stake proofs *Stake*, *Unstake*, and *StakeVote*, a document containing full diagrams and specifications for their circuits and logic will be made available on our site once finalized. We also want to detail the ongoing research done within the privacy community regarding privacy upgrades to the core coin protocol in an "anticipated upgrades" publication.

7 Acknowledgements

We would like to thank Adam Gągol for his insight into asynchronous BFT. We would also like to thank Aaron Feickert (Sarang Noether) for his initial explanations of Triptych with our team in early 2021. Special thanks to Ukoe of MobileCoin for his insight into Seraphis, a major upgrade in privacy coin protocols to be released next year. Finally, we thank our community on various social media sites for their contribution to the project and support of our team.

References

[1] Chaya Ganesh, Claudio Orlandi, and Daniel Tschudi. *Proof-of-Stake Protocols for Privacy-Aware Blockchains*. Cryptology ePrint Archive, Paper 2018/1105. https://eprint.iacr.org/2018/1105. 2018. URL: https://eprint.iacr.org/2018/1105.

[2] Heewon Chung, Kyoohyung Han, and Chanyang Ju. *Bulletproofs+: Shorter Proofs for Privacy-Enhanced Distributed Ledger*. 2020. URL: https://eprint.iacr.org/2020/735.pdf.

[3] Jens Groth and Markulf Kohlweiss. *One-out-of-Many Proofs: Or How to Leak a Secret and Spend a Coin*. 2014. URL: https://eprint.iacr.org/2014/764.pdf.

[4] Sarang Noether and Brandon Goodell. *Triptych: logarithmic-sized linkable ring signatures with applications*. Cryptology ePrint Archive, Report 2020/018. https://eprint.iacr.org/2020/018.pdf. 2020.

[5] Muhammed F. Esgin et al. "MatRiCT: Efficient, Scalable and Post-Quantum Blockchain Confidential Transactions Protocol". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. CCS '19. London, United Kingdom: Association for Computing Machinery, 2019, 567–584. ISBN: 9781450367479. DOI: 10.1145/3319535.3354200. URL: https://doi.org/10.1145/3319535.3354200.

[6] Malte Möser et al. *An Empirical Analysis of Traceability in the Monero Blockchain*. 2018. arXiv: 1704.04299 [cs.CR].

[7] Adam Gągol, Damian Leundefinedniak, and Damian Straszak. "Aleph: Efficient Atomic Broadcast in Asynchronous Networks with Byzantine Nodes". In: *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*. AFT '19. Zurich, Switzerland: Association for Computing Machinery, 2019, 214–228. ISBN: 9781450367325. DOI: 10.1145/3318041.3355467. URL: https://doi.org/10.1145/3318041.3355467.

[8] Damian Straszak and Michal Handzlik. *Aleph Zero Foundation: Aleph Consensus*. 2021. URL: https://github.com/aleph-zero-foundation/consensus-go.

[9] Augusto Devegili, Michael Scott, and Ricardo Dahab. "Implementing Cryptographic Pairings over Barreto-Naehrig Curves". In: vol. 4575. July 2007, pp. 197–207. ISBN: 978-3-540-73488-8. DOI: 10.1007/978-3-540-73489-5_10.