MantaPay Protocol Specification
v0.4.0

Shumo Chu and Brandon H. Gomes

November 1, 2021

Abstract

MantaPay is an implementation of a *decentralized anonymous payment* scheme based on the MANTA$_{\text{DAP}}$ protocol outlined in the original MANTA whitepaper.

Contents

1 Introduction

TODO: add introductory remarks

2 Notation

The following notation is used throughout this specification:

- Type is the type of types[1].
- If $x : T$ then x is a value and T is a type, denoted $T : \mathsf{Type}$, and we say that x *has type* T.
- Bool is the type of booleans with values True and False.
- For any types $A : \mathsf{Type}$ and $B : \mathsf{Type}$ we denote the *type of functions* from A to B as $A \to B : \mathsf{Type}$.
- For any types $A : \mathsf{Type}$ and $B : \mathsf{Type}$ we denote the *product type* over A and B as $A \times B : \mathsf{Type}$ with constructor $(-,-) : T \to (S \to T \times S)$.
- For any type $T : \mathsf{Type}$, we define $\mathsf{Option}(T) : \mathsf{Type}$ as the inductive type with constructors:

$$\mathsf{None} : \mathsf{Option}(T)$$
$$\mathsf{Some} : T \to \mathsf{Option}(T)$$

- We denote the *type of finite sets* over a type $T : \mathsf{Type}$ as $\mathsf{FinSet}(T) : \mathsf{Type}$. The membership predicate for a value $x : T$ in a finite set $S : \mathsf{FinSet}(T)$ is denoted $x \in S$.
- We denote the *type of distributions* over a type $T : \mathsf{Type}$ as $\mathfrak{D}(T) : \mathsf{Type}$. A value x sampled from $\mathfrak{D}(T)$ is denoted $x \sim \mathfrak{D}(T)$ and the fact that the value x belongs to the range of $\mathfrak{D}(T)$ is denoted $x \in \mathfrak{D}(T)$. So namely, $y \in \{x \mid x \sim \mathfrak{D}(T)\} \leftrightarrow y \in \mathfrak{D}(T)$.
- Depending on the context, the notation $|\cdot|$ denotes either the absolute value of a quantity, the length of a vector, the number of characters in a string, or the cardinality of a set.

3 Concepts

3.1 Assets

The Asset is the fundamental currency object in the MantaPay protocol. An asset $a : \mathsf{Asset}$ is a tuple

$$a = (a.\mathsf{id}, a.\mathsf{value}) : \mathsf{AssetId} \times \mathsf{AssetValue}$$

where the AssetId represents the type of currency stored in a and the AssetValue represents how many units of that currency are stored in a. MantaPay is a *decentralized anonymous payment* protocol which facilitiates the private ownership and private transfer of Asset objects.

Whenever an Asset is being used in a public setting, we simply refer to it as an Asset, but when the AssetId and/or AssetValue of a particular Asset is meant to be hidden from public view, we refer to the Asset as either, *secret*, *private*, *hidden*, or *shielded*.

Assets form the basic units of *transactions* which consume Assets on input, transform them, and return Assets on output. To preserve the economic value stored in Assets, the sum of the input AssetValues must balance the sum of the output AssetValues, and all assets in a single transaction must have the same AssetId[2]. This is called a *balanced transfer*: no AssetValue is created or destroyed in the process. The MantaPay protocol uses a distributed algorithm called Transfer to perform balanced transfers and ensure that they are valid.

3.2 Addresses

In order for MantaPay participants to send and receive Assets via the Transfer protocol, they create *addresses* which represent their partipation in the protocol. MantaPay has a 3-address system consisting of a *spending key* sk, a *viewing key* vk, and a *receiving key* rk. The keys have the following uses/properties:

- Access to a receiving key rk represents the ability to send Assets to the owner of the associated sk.
- Access to a viewing key vk represents the ability to reveal shielded Asset information for Assets belonging to the owner of the associated sk.

[1] By *type of types*, we mean the type of *first-level* types in some family of type universes. Discussion of the type theory necessary to make these notions rigorous is beyond the scope of this paper.

[2] It is beyond the scope of this paper to discuss transactions with inputs and outputs that feature different AssetIds, like those that would be featured in a *decentralized anonymous exchange*.

- Access to a spending key sk represents the ability to spend Assets that were received under the associated receiving key rk.

Participants in MantaPay are represented by their addresses, but they are not unique representations, since one participant may have access to more than one triple of keys. See § 4.2 for more information on how these keys are constructed and used for spending, viewing, and receiving Assets.

3.3 Ledger

Preserving the economic value of Assets requires more than just balanced transfers. It also requires that Assets are owned by exactly one address at a time, namely, that the ability to spend an Asset can be proved before a transfer and revoked after a transfer. It is not simply the *information-content* of an Asset that should be transfered, but the *ability to spend the asset in the future*, which should be transfered. Enforcing this second invariant can be solved by using a public ledger[3] that keeps track of the movement of Assets from one participant to another. Unfortunately, using a public ledger alone does not allow participants to remain anonymous, so MantaPay extends the public ledger by adding a special account called the *shielded asset pool* which is responsible for keeping track of the Assets which have been anonymized by the protocol. We denote the three ledger types in the protocol as follows: the public ledger as PublicLedger, the shielded asset pool as ShieldedAssetPool, and the combined ledger we denote Ledger.

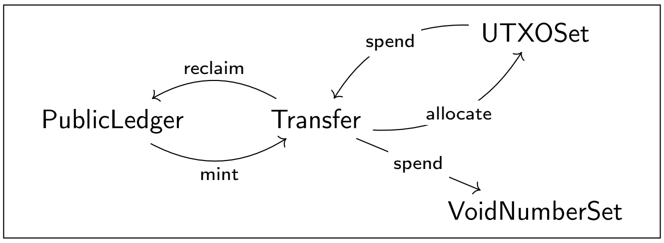

Figure 1: Lifecycle of an Asset.

The ShieldedAssetPool is made up of four parts which serve to enforce the balanced transfer of Assets among anonymous participants:

1. ShieldedAssetPool Balance: The Ledger contains a collection of Assets which represent the combined economic value of the ShieldedAssetPool and the PublicLedger. The ShieldedAssetPool balance is the subset of this total collection that has been anonymized by the MantaPay protocol. This balance is represented by a finite set of non-zero Assets.
2. § 3.3.1 UTXOSet: The UTXOSet is a collection of ownership claims to subsets of the ShieldedAssetPool (called UTXOs), each one refering to an allocated Asset transfered to a participant of the protocol.
3. § 3.3.2 EncryptedNotes: For every UTXO there is a matching EncryptedNote which contains information necessary to spend the Asset, which can be used to *provably reconstruct* the UTXO convincing the Ledger of unique ownership. The EncryptedNote can only be decrypted by the recipient of the Asset, specifically, the correct viewing key vk. See § 3.2 for more.
4. § 3.3.3 VoidNumberSet: The VoidNumberSet is a collection of commitments, like UTXOs, but which track the *spent state* of an Asset and are used to prove to the Ledger that an Asset is spent *exactly one time.*

The operation of these different parts of the ShieldedAssetPool is elaborated in the following subsections.

3.3.1 UTXOs and the UTXOSet

An *unspent transaction output*, or UTXO for short, represents a claim to the output of a balanced transfer which has otherwise *not yet been spent.* Every balanced transfer produces *public outputs*, just publicly visible Assets, and *private outputs*, represented by UTXOs, and these UTXOs are stored in the UTXOSet of the ShieldedAssetPool. A UTXO can only be claimed by the participant who owns the underlying Asset, where ownership means *knowledge of the correct spending key* and the Transfer protocol requires that all inputs to a balanced transfer *prove* that they own a UTXO which the ShieldedAssetPool has already seen in the past. The UTXOSet is *append-only* since it represents the past state of *unspent* Assets. UTXOs can only be added to the UTXOSet as outputs in the execution of a Transfer which the Ledger checks for correctness.

3.3.2 EncryptedNotes

In order to find out what Asset a UTXO is connected to, every UTXO comes with an associated EncryptedNote which stores two pieces of information, the underlying Asset, and a key diversifier, a value which allows the new

[3]A public (or private) ledger is not enough to solve the *provable-ownership problem* or the *double-spending problem.* A *consensus mechanism* is also required to ensure that all participants agree on the current state of the ledger. The *consensus mechanism* that secures the MantaPay ledger is beyond the scope of this paper.

owner of the Asset to reconstruct the UTXO. Being able to *provably reconstruct* a correct UTXO is a prerequisite to ownership and the ability to spend the Asset in the future. Once a participant spends an Asset that they can decrypt, they build a new EncryptedNote for the next participant that they sent their Assets to, so that they can then spend it, and so on. This is called the *in-band secret distribution*.

3.3.3 VoidNumbers and the VoidNumberSet

Once the ability to spend an Asset is extracted from a (UTXO, EncryptedNote) pair, the ShieldedAssetPool requires another commitment in order to spend the Asset, transfering it to another participant. This commitment, called the VoidNumber, represents the revocation of the right to spend the Asset in the future, and ensures that the same Asset cannot be spent twice. Like the UTXOSet, the VoidNumberSet is *append-only* since it represents the past state of *spent* Assets. VoidNumbers can only be added to the VoidNumberSet as inputs in the execution of a Transfer which the Ledger checks for correctness.

4 Abstract Protocol

4.1 Abstract Cryptographic Schemes

In the following section, we outline the formal specifications for all of the *cryptographic primitives* used in the MantaPay protocol.

Definition 4.1.1. A *commitment scheme* COM is defined by the schema:

$$\begin{aligned} \mathsf{Trapdoor} &: \mathsf{Type} \\ \mathsf{Input} &: \mathsf{Type} \\ \mathsf{Output} &: \mathsf{Type} \\ \mathsf{TrapdoorDistribution} &: \mathfrak{D}(\mathsf{Trapdoor}) \\ \mathsf{commit} &: \mathsf{Trapdoor} \times \mathsf{Input} \to \mathsf{Output} \end{aligned}$$

with the following properties:

- **Binding**: It is infeasible to find an x, y : Input and r, s : Trapdoor such that $x \neq y$ and $\mathsf{commit}(r, x) = \mathsf{commit}(s, y)$.
- **Hiding**: For all x, y : Input, the distributions $\{\mathsf{commit}(r, x) \mid r \sim \mathsf{TrapdoorDistribution}\}$ and $\{\mathsf{commit}(r, y) \mid r \sim \mathsf{TrapdoorDistribution}\}$ are *computationally indistinguishable*.

Notation: For convenience we refer to $\mathsf{COM.commit}(r, x)$ by $\mathsf{COM}_r(x)$.

Definition 4.1.2. A *hash function* CRH is defined by the schema:

$$\begin{aligned} \mathsf{Input} &: \mathsf{Type} \\ \mathsf{Output} &: \mathsf{Type} \\ \mathsf{hash} &: \mathsf{Input} \to \mathsf{Output} \end{aligned}$$

with the following properties:

- **Pre-Image Resistance**: For a given y : Output, it is infeasible to find x : Input such that $\mathsf{hash}(x) = y$.
- **Collision Resistance**: It is infeasible to find an x_1, x_2 : Input such that $x_1 \neq x_2$ and $\mathsf{hash}(x_1) = \mathsf{hash}(x_2)$.

Notation: For convenience we refer to $\mathsf{CRH.hash}(x)$ by $\mathsf{CRH}(x)$.

Definition 4.1.3. A *symmetric-key encryption scheme* SYM is defined by the schema:

$$\begin{aligned} \mathsf{Key} &: \mathsf{Type} \\ \mathsf{Plaintext} &: \mathsf{Type} \\ \mathsf{Ciphertext} &: \mathsf{Type} \\ \mathsf{encrypt} &: \mathsf{Key} \times \mathsf{Plaintext} \to \mathsf{Ciphertext} \\ \mathsf{decrypt} &: \mathsf{Key} \times \mathsf{Ciphertext} \to \mathsf{Option}(\mathsf{Plaintext}) \end{aligned}$$

with the following properties:

- **Validity**: For all keys k : Key and plaintexts p : Plaintext, we have that

$$\mathsf{decrypt}(k, \mathsf{encrypt}(k, p)) = \mathsf{Some}(p)$$

- **TODO**: hiding, one-time encryption security?

Definition 4.1.4. A *key-agreement scheme* KA is defined by the schema:

$$\begin{aligned}
\mathsf{PublicKey} &: \mathsf{Type}\\
\mathsf{SecretKey} &: \mathsf{Type}\\
\mathsf{SharedSecret} &: \mathsf{Type}\\
\mathsf{derive} &: \mathsf{SecretKey} \to \mathsf{PublicKey}\\
\mathsf{agree} &: \mathsf{SecretKey} \times \mathsf{PublicKey} \to \mathsf{SharedSecret}
\end{aligned}$$

with the following properties:

- **Agreement**: For all $\mathsf{sk}_1, \mathsf{sk}_2 : \mathsf{SecretKey}$, $\mathsf{agree}(\mathsf{sk}_1, \mathsf{derive}(\mathsf{sk}_2)) = \mathsf{agree}(\mathsf{sk}_2, \mathsf{derive}(\mathsf{sk}_1))$
- **TODO**: security properties

Definition 4.1.5. A *key-derivation function* KDF defined over a symmetric-key encryption scheme SYM and a key-agreement scheme KA is a function of type:

$$\mathsf{KDF} : \mathsf{KA.SharedSecret} \to \mathsf{SYM.Key}$$

Definition 4.1.6. An *integrated encryption scheme* IES is a hybrid encryption scheme made of up a symmetric-key encryption scheme SYM, a key-agreement scheme KA, and a KDF to convert from $\mathsf{KA.SharedSecret}$ to $\mathsf{SYM.Key}$. We can define the following encryption/decryption algorithms:

- Encryption: Given a secret key $\mathsf{sk} : \mathsf{KA.SecretKey}$, a public key $\mathsf{pk} : \mathsf{KA.PublicKey}$, and plaintext $p : \mathsf{SYM.Plaintext}$, we produce the pair

  $$m := (\mathsf{KA.derive}(\mathsf{sk}), \mathsf{SYM.encrypt}(\mathsf{KDF}(\mathsf{KA.agree}(\mathsf{sk}, \mathsf{pk})), p)) : \mathsf{KA.PublicKey} \times \mathsf{SYM.Ciphertext}$$

- Decryption: Given a secret key $\mathsf{sk} : \mathsf{KA.SecretKey}$, and an encrypted message, as above, $m := (\mathsf{pk}, c) : \mathsf{KA.PublicKey} \times \mathsf{SYM.Ciphertext}$, we can decrypt m, producing the plaintext,

  $$p := \mathsf{SYM.decrypt}(\mathsf{KDF}(\mathsf{KA.agree}(\mathsf{sk}, \mathsf{pk})), c) : \mathsf{Option}(\mathsf{SYM.Plaintext})$$

  which should decrypt successfully if the $\mathsf{KA.PublicKey}$ that m was encrypted with is the derived key of $\mathsf{sk} : \mathsf{KA.SecretKey}$.

Notation: We denote the above *encrypted message* type as $\mathsf{Message} := \mathsf{KA.PublicKey} \times \mathsf{SYM.Ciphertext}$, and the above two algorithms by

$$\begin{aligned}
\mathsf{encrypt} &: \mathsf{KA.SecretKey} \times \mathsf{KA.PublicKey} \times \mathsf{SYM.Plaintext} \to \mathsf{Message}\\
\mathsf{decrypt} &: \mathsf{KA.SecretKey} \times \mathsf{SYM.Ciphertext} \to \mathsf{Option}(\mathsf{SYM.Plaintext})
\end{aligned}$$

TODO: security properties, combine with SYM and KA properties, like the fact that some of these keys should be ephemeral, etc.
TODO: add explicit message authentication

Definition 4.1.7. A *key-diversification scheme* KDIV over a key-agreement scheme KA is defined by the schema:

$$\begin{aligned}
\mathsf{public} &: \mathsf{KA.SecretKey} \times \mathsf{KA.PublicKey} \times \mathsf{KA.PublicKey} \to \mathsf{KA.PublicKey}\\
\mathsf{secret} &: \mathsf{KA.PublicKey} \times \mathsf{KA.SecretKey} \times \mathsf{KA.SecretKey} \to \mathsf{KA.SecretKey}
\end{aligned}$$

Notation: We refer to the first argument to a KDIV function as the *diversifier* and we write it as a subscript

$$\mathsf{public}_d(x, y) := \mathsf{public}(d, x, y) \text{ and } \mathsf{secret}_d(x, y) := \mathsf{secret}(d, x, y)$$

For convenience we also write KDIV_d to mean $\mathsf{KDIV.public}_d$ or $\mathsf{KDIV.secret}_d$, when the context is clear.

Every KDIV also has the following properties:

- **Derivation Invariance**: For any diversifier $d : \mathsf{KA.SecretKey}$ and pair of secret keys $(\mathsf{sk}_1, \mathsf{sk}_2)$ we have

  $$\mathsf{KDIV}_d(\mathsf{KA.derive}(\mathsf{sk}_1), \mathsf{KA.derive}(\mathsf{sk}_2)) = \mathsf{KA.derive}(\mathsf{KDIV}_{\mathsf{KA.derive}(d)}(\mathsf{sk}_1, \mathsf{sk}_2))$$

- **TODO**: security properties?

Definition 4.1.8. A *dynamic cryptographic accumulator* DCA is defined by the schema:

$$\begin{aligned}
\mathsf{Item} &: \mathsf{Type} \\
\mathsf{State} &: \mathsf{Type} \\
\mathsf{Checkpoint} &: \mathsf{Type} \\
\mathsf{Proof} &: \mathsf{Type} \\
\mathsf{checkpoint} &: \mathsf{State} \to \mathsf{Checkpoint} \\
\mathsf{update} &: \mathsf{Item} \times \mathsf{State} \to \mathsf{State} \\
\mathsf{contains} &: \mathsf{Item} \times \mathsf{State} \to \mathsf{Option}(\mathsf{Checkpoint} \times \mathsf{Proof}) \\
\mathsf{verify} &: \mathsf{Item} \times \mathsf{Checkpoint} \times \mathsf{Proof} \to \mathsf{Bool}
\end{aligned}$$

with the following properties:

- **TODO**: add checkpoint, update, contains, and verify properties
- **TODO**: security properties

Definition 4.1.9. A *non-interactive zero-knowledge proving system* ZKPS is defined by the schema:

$$\begin{aligned}
\mathsf{Statement} &: \mathsf{Type} \\
\mathsf{ProvingKey} &: \mathsf{Type} \\
\mathsf{VerifyingKey} &: \mathsf{Type} \\
\mathsf{PublicInput} &: \mathsf{Type} \\
\mathsf{SecretInput} &: \mathsf{Type} \\
\mathsf{Proof} &: \mathsf{Type} \\
\mathsf{keys} &: \mathsf{Statement} \to \mathfrak{D}(\mathsf{ProvingKey} \times \mathsf{VerifyingKey}) \\
\mathsf{prove} &: \mathsf{Statement} \times \mathsf{ProvingKey} \times \mathsf{PublicInput} \times \mathsf{SecretInput} \to \mathfrak{D}(\mathsf{Option}(\mathsf{Proof})) \\
\mathsf{verify} &: \mathsf{VerifyingKey} \times \mathsf{PublicInput} \times \mathsf{Proof} \to \mathsf{Bool}
\end{aligned}$$

Notation: We use the following notation for a ZKPS:

- We write the $\mathsf{Statement}$ and $\mathsf{ProvingKey}$ arguments of prove in the superscript and subscript respectively,

$$\mathsf{prove}^{P}_{\mathsf{pk}}(x, w) := \mathsf{prove}(P, \mathsf{pk}, x, w)$$

- We write the $\mathsf{VerifyingKey}$ argument of verify in the subscript,

$$\mathsf{verify}_{\mathsf{vk}}(x, \pi) := \mathsf{verify}(\mathsf{vk}, x, \pi)$$

- We say that $(x, w) : \mathsf{PublicInput} \times \mathsf{SecretInput}$ has the property of being a $\mathsf{satisfying}$ input whenever

$$\mathsf{satisfying}(x, w) := \exists \pi : \mathsf{Proof},\ \mathsf{Some}(\pi) \in \mathsf{prove}^{P}_{\mathsf{pk}}(x, w)$$

Every ZKPS has the following properties for a fixed statement $P : \mathsf{Statement}$ and keys $(\mathsf{pk}, \mathsf{vk}) \sim \mathsf{keys}(P)$:

- **Completeness**: For all $(x, w) : \mathsf{PublicInput} \times \mathsf{SecretInput}$, if there exists a proof $\pi : \mathsf{Proof}$, such that $\mathsf{Some}(\pi) \in \mathsf{prove}^{P}_{\mathsf{pk}}(x, w)$, then $\mathsf{verify}_{\mathsf{vk}}(x, \pi) = \mathsf{True}$.
- **Knowledge Soundness**: For any polynomial-size adversary $\mathcal{A}$,

$$\mathcal{A} : \mathsf{ProvingKey} \times \mathsf{VerifyingKey} \to \mathfrak{D}(\mathsf{PublicInput} \times \mathsf{Proof})$$

  there exists a polynomial-size extractor $\mathcal{E}_{\mathcal{A}}$

$$\mathcal{E}_{\mathcal{A}} : \mathsf{ProvingKey} \times \mathsf{VerifyingKey} \to \mathfrak{D}(\mathsf{SecretInput})$$

  such that the following probability is negligible:

$$\Pr\left[\begin{array}{c} \mathsf{satisfying}(x, w) = \mathsf{False} \\ \mathsf{verify}_{\mathsf{vk}}(x, w) = \mathsf{True} \end{array} \,\middle|\, \begin{array}{l} (\mathsf{pk}, \mathsf{vk}) \sim \mathsf{keys}(P) \\ (x, \pi) \sim \mathcal{A}(\mathsf{pk}, \mathsf{vk}) \\ w \sim \mathcal{E}_{\mathcal{A}}(\mathsf{pk}, \mathsf{vk}) \end{array}\right]$$

- **Statistical Zero-Knowledge**: There exists a stateful simulator $\mathcal{S}$, such that for all stateful distinguishers $\mathcal{D}$, the difference between the following two probabilities is negligible:

$$\Pr\left[\begin{array}{c}\mathsf{satisfying}(x,w)=\mathsf{True}\\ \mathcal{D}(\pi)=\mathsf{True}\end{array}\;\middle|\;\begin{array}{l}(\mathsf{pk},\mathsf{vk})\sim\mathsf{keys}(P)\\ (x,w)\sim\mathcal{D}(\mathsf{pk},\mathsf{vk})\\ \mathsf{Some}(\pi)\sim\mathsf{prove}^P_{\mathsf{pk}}(x,w)\end{array}\right] \text{ and } \Pr\left[\begin{array}{c}\mathsf{satisfying}(x,w)=\mathsf{True}\\ \mathcal{D}(\pi)=\mathsf{True}\end{array}\;\middle|\;\begin{array}{l}(\mathsf{pk},\mathsf{vk})\sim\mathcal{S}(P)\\ (x,w)\sim\mathcal{D}(\mathsf{pk},\mathsf{vk})\\ \pi\sim\mathcal{S}(x)\end{array}\right]$$

- **Succinctness**: For all $(x,w) : \mathsf{PublicInput}\times\mathsf{SecretInput}$, if $\mathsf{prove}(P,\mathsf{pk},x,w)=\mathsf{Some}(\pi)$, then $|\pi|=\mathcal{O}(1)$, and $\mathsf{verify}(\mathsf{vk},x,\pi)$ runs in time $\mathcal{O}(|x|)$.

4.2 Addresses and Key Components

Given a choice of IES for the Transfer protocol we have the following definitions:

Definition 4.2.1. A SpendingKey is the following pair of keys:

$$\begin{aligned}\mathsf{view} &: \mathsf{IES.KA.SecretKey}\\ \mathsf{spend} &: \mathsf{IES.KA.SecretKey}\end{aligned}$$

The first secret key, view, is called the ViewingKey.

Definition 4.2.2. A ReceivingKey is the following pair of keys:

$$\begin{aligned}\mathsf{view} &: \mathsf{IES.KA.PublicKey}\\ \mathsf{spend} &: \mathsf{IES.KA.PublicKey}\end{aligned}$$

which is derived from a spending key $\mathsf{sk} : \mathsf{SpendingKey}$ by deriving each component:

$$\begin{aligned}\mathsf{rk.view} &:= \mathsf{KA.derive}(\mathsf{sk.view})\\ \mathsf{rk.spend} &:= \mathsf{KA.derive}(\mathsf{sk.spend})\end{aligned}$$

4.3 Transfer Protocol

Definition 4.3.1. A Sender is the following tuple:

$$\begin{aligned}\mathsf{sk} &: \mathsf{SpendingKey}\\ \tilde{d} &: \mathsf{IES.KA.PublicKey}\\ \mathsf{trapdoor} &: \mathsf{IES.KA.PublicKey}\\ \mathsf{asset} &: \mathsf{Asset}\\ \mathsf{cm} &: \mathsf{UTXO}\\ \mathsf{cm}_c &: \mathsf{UTXOSet.Checkpoint}\\ \mathsf{cm}_\pi &: \mathsf{UTXOSet.Proof}\\ \mathsf{vn} &: \mathsf{VoidNumber}\end{aligned}$$

A Sender, S, is constructed from a spending key $\mathsf{sk} : \mathsf{SpendingKey}$ and an encrypted message $\mathsf{note} : \mathsf{EncryptedNote}$ with the following algorithm:

$$\begin{aligned}S.\mathsf{sk} &:= \mathsf{sk}\\ \tilde{d}, c &:= \mathsf{note}\\ \mathsf{Some}(\mathsf{asset}) &:= \mathsf{IES.decrypt}(S.\mathsf{sk.view}, c)\\ S.\mathsf{asset} &:= \mathsf{asset}\\ S.\tilde{d} &:= \tilde{d}\\ S.\mathsf{trapdoor} &:= \mathsf{KA.derive}(\mathsf{KDIV}_{S.\tilde{d}}(S.\mathsf{sk.view}, S.\mathsf{sk.spend}))\\ S.\mathsf{cm} &:= \mathsf{COM}_{S.\mathsf{trapdoor}}(S.\mathsf{asset})\\ \mathsf{Some}(\mathsf{cm}_c, \mathsf{cm}_\pi) &:= \mathsf{UTXOSet.contains}(S.\mathsf{cm}, \mathsf{Ledger.utxos}())\\ S.\mathsf{cm}_c &:= \mathsf{cm}_c\\ S.\mathsf{cm}_\pi &:= \mathsf{cm}_\pi\\ S.\mathsf{vn} &:= \mathsf{COM}_{S.\tilde{d}}(S.\mathsf{sk.view} \,||\, S.\mathsf{sk.spend})\end{aligned}$$

Definition 4.3.2. A $\mathsf{SenderPost}$ is the following tuple extracted from a Sender:

$$\begin{aligned} \mathsf{cm}_c &: \mathsf{UTXOSet.Checkpoint} \\ \mathsf{vn} &: \mathsf{VoidNumber} \end{aligned}$$

Definition 4.3.3. A $\mathsf{Receiver}$ is the following tuple:

$$\begin{aligned} \mathsf{rk} &: \mathsf{ReceivingKey} \\ d &: \mathsf{IES.KA.SecretKey} \\ \mathsf{trapdoor} &: \mathsf{IES.KA.PublicKey} \\ \mathsf{asset} &: \mathsf{Asset} \\ \mathsf{cm} &: \mathsf{UTXO} \\ \mathsf{note} &: \mathsf{EncryptedNote} \end{aligned}$$

A $\mathsf{Receiver}$, R, is constructed from a receving key $\mathsf{rk} : \mathsf{ReceivingKey}$, an asset $\mathsf{asset} : \mathsf{Asset}$, and a chosen diversifier $d : \mathsf{IES.KA.SecretKey}$ with the following algorithm:

$$\begin{aligned} R.\mathsf{rk} &:= \mathsf{rk} \\ R.d &:= d \\ R.\mathsf{trapdoor} &:= \mathsf{KDIV}_{R.d}(R.\mathsf{rk.view}, R.\mathsf{rk.spend}) \\ R.\mathsf{asset} &:= \mathsf{asset} \\ R.\mathsf{cm} &:= \mathsf{COM}_{R.\mathsf{trapdoor}}(R.\mathsf{asset}) \\ R.\mathsf{note} &:= \mathsf{IES.encrypt}(R.d, R.\mathsf{rk.view}, \mathsf{asset}) \end{aligned}$$

Definition 4.3.4. A $\mathsf{ReceiverPost}$ is the following tuple extracted from a $\mathsf{Receiver}$:

$$\begin{aligned} \mathsf{cm} &: \mathsf{UTXO} \\ \mathsf{note} &: \mathsf{EncryptedNote} \end{aligned}$$

Definition 4.3.5. A Source (or a Sink) is an Asset representing a public input (or output) of a $\mathsf{Transfer}$.

Definition 4.3.6. A $\mathsf{Transfer}$ is the following tuple:

$$\begin{aligned} \mathsf{sources} &: \mathsf{FinSet}(\mathsf{Asset}) \\ \mathsf{senders} &: \mathsf{FinSet}(\mathsf{Sender}) \\ \mathsf{receivers} &: \mathsf{FinSet}(\mathsf{Receiver}) \\ \mathsf{sinks} &: \mathsf{FinSet}(\mathsf{Asset}) \end{aligned}$$

The *shape* of a $\mathsf{Transfer}$ is the following 4-tuple of cardinalities of those sets

$$\big(|T.\mathsf{sources}|, |T.\mathsf{senders}|, |T.\mathsf{receivers}|, |T.\mathsf{sinks}|\big)$$

In order for a $\mathsf{Transfer}$ to be considered *valid*, it must adhere to the following constraints:

- **Same Id**: All the $\mathsf{AssetId}$s in the $\mathsf{Transfer}$ must be equal.
- **Balanced**: The sum of input $\mathsf{AssetValue}$s must be equal to the sum of output $\mathsf{AssetValue}$s.
- **Well-formed Senders**: All of the Senders in the $\mathsf{Transfer}$ must be constructed according to the above Sender definition.
- **Well-formed Receivers**: All of the $\mathsf{Receiver}$s in the $\mathsf{Transfer}$ must be constructed according to the above $\mathsf{Receiver}$ definition.

In order to prove that these constraints are satisfied for a given $\mathsf{Transfer}$, we build a zero-knowledge proof which will witness that the $\mathsf{Transfer}$ is valid and should be accepted by the ledger. It is not necessary to prove that the encryption of $\mathsf{Receiver.note}$ and the decryption of a note from the ledger are valid. Deviation from the protocol in encryption or decryption does not reduce the security of the protocol for honest participants.

Definition 4.3.7. ($\mathsf{Transfer}$ Validity Statement) A transfer $T : \mathsf{Transfer}$ is considered *valid* if and only if

1. All the $\mathsf{AssetIds}$ in T are equal:

$$\left|\left(\bigcup_{a\in T.\mathsf{sources}} a.\mathsf{id}\right) \cup \left(\bigcup_{S\in T.\mathsf{senders}} S.\mathsf{asset.id}\right) \cup \left(\bigcup_{R\in T.\mathsf{receivers}} R.\mathsf{asset.id}\right) \cup \left(\bigcup_{a\in T.\mathsf{sinks}} a.\mathsf{id}\right)\right| = 1$$

2. The sum of input $\mathsf{AssetValues}$ is equal to the sum of output $\mathsf{AssetValues}$:

$$\left(\sum_{a\in T.\mathsf{sources}} a.\mathsf{value}\right) + \left(\sum_{S\in T.\mathsf{senders}} S.\mathsf{asset.value}\right) = \left(\sum_{R\in T.\mathsf{receivers}} R.\mathsf{asset.value}\right) + \left(\sum_{a\in T.\mathsf{sinks}} a.\mathsf{value}\right)$$

3. For all $S \in T.\mathsf{senders}$, the Sender S is well-formed:

$$\begin{aligned} S.\mathsf{trapdoor} &= \mathsf{KA.derive}(\mathsf{KDIV}_{S.\tilde{d}}(S.\mathsf{sk.view}, S.\mathsf{sk.spend})) \\ S.\mathsf{cm} &= \mathsf{COM}_{S.\mathsf{trapdoor}}(S.\mathsf{asset}) \\ S.\mathsf{vn} &= \mathsf{COM}_{S.\tilde{d}}(S.\mathsf{sk.view} \,||\, S.\mathsf{sk.spend}) \\ \mathsf{UTXOSet.verify}(S.\mathsf{cm}, S.\mathsf{cm}_c, S.\mathsf{cm}_\pi) &= \mathsf{True} \end{aligned}$$

4. For all $(i, R) \in \mathsf{enumerate}(T.\mathsf{receivers})$, the $\mathsf{Receiver}$ R is well-formed at index i with respect to FAIR:

$$\begin{aligned} R.d &= \mathsf{CRH}(i \,||\, R.\mathsf{rk.view} \,||\, R.\mathsf{rk.spend} \,||\, \mathsf{FAIR}) \\ R.\mathsf{trapdoor} &= \mathsf{KDIV}_{R.d}(R.\mathsf{rk.view}, R.\mathsf{rk.spend}) \\ R.\mathsf{cm} &= \mathsf{COM}_{R.\mathsf{trapdoor}}(R.\mathsf{asset}) \end{aligned}$$

   where FAIR is the following constant, using the ledger as a randomness oracle:

$$\mathsf{FAIR} := \mathsf{CRH}(\mathsf{UTXOSet.checkpoint}(\mathsf{Ledger.utxos}()) \,||\, \mathsf{Concat}_{S\in T.\mathsf{senders}}(S.\mathsf{sk.spend}))$$

Notation: This statement is denoted $\mathsf{ValidTransfer}$ and is assumed to be expressible as a $\mathsf{Statement}$ of ZKPS.

Definition 4.3.8. A $\mathsf{TransferPost}$ is the following tuple:

$$\begin{aligned} \mathsf{sources} &: \mathsf{FinSet}(\mathsf{Asset}) \\ \mathsf{senders} &: \mathsf{FinSet}(\mathsf{SenderPost}) \\ \mathsf{receivers} &: \mathsf{FinSet}(\mathsf{ReceiverPost}) \\ \mathsf{sinks} &: \mathsf{FinSet}(\mathsf{Asset}) \\ \pi &: \mathsf{ZKPS.Proof} \end{aligned}$$

A $\mathsf{TransferPost}$, P, is constructed by assembling the zero-knowledge proof of $\mathsf{Transfer}$ validity from a known proving key $\mathsf{pk} : \mathsf{ZKPS.ProvingKey}$ and a given $T : \mathsf{Transfer}$:

$$\begin{aligned} x &:= \mathsf{Transfer.public}(T) \\ w &:= \mathsf{Transfer.secret}(T) \\ \mathsf{Some}(\pi) &:= \mathsf{ZKPS.prove}_{\mathsf{pk}}^{\mathsf{ValidTransfer}}(x, w) \\ P.\mathsf{sources} &:= x.\mathsf{sources} \\ P.\mathsf{senders} &:= x.\mathsf{senders} \\ P.\mathsf{receivers} &:= x.\mathsf{receivers} \\ P.\mathsf{sinks} &:= x.\mathsf{sinks} \\ P.\pi &:= \pi \end{aligned}$$

5 Concrete Protocol

5.1 Constants

TODO: add constants for the protocol

5.2 Concrete Cryptographic Schemes

TODO: add names of cryptographic scheme implementations

5.2.1 Commitments

5.2.2 Hash Functions

5.2.3 Encryption

5.2.4 Zero-Knowledge Proving Systems

5.2.4.1 Groth16

5.2.4.2 PLONK

6 Differences from $\textsc{Manta}_{\texttt{DAP}}$

6.1 Reusable Addresses

TODO: compare old one-time address protocol to reusable addresses and why reusable is better

6.2 Transfer Circuit Unification

TODO: compare new single transfer circuit to the many old circuits

7 Acknowledgements

TODO: add acknowledgements

8 References

TODO: add references