MantaPay Protocol Specification
v0.4.0

Shumo Chu and Brandon H. Gomes

October 31, 2021

Abstract

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

Contents

1 Introduction **1**

2 Notation **2**

3 Concepts **2**
3.1 Assets . . . 2
3.2 Addresses . . . 2
3.3 Ledger . . . 3
3.3.1 UTXO Set . . . 3
3.3.2 Encrypted Notes . . . 3
3.3.3 VoidNumber Set . . . 3

4 Abstract Protocol **3**
4.1 Abstract Cryptographic Schemes . . . 3
4.2 Addresses and Key Components . . . 6
4.3 Transfer Protocol . . . 6
4.3.1 Senders . . . 6
4.3.2 Receivers . . . 6
4.3.3 Transfers . . . 6
4.3.4 TransferPosts . . . 6

5 Concrete Protocol **6**
5.1 Conventions . . . 6
5.2 Constants . . . 6
5.3 Concrete Cryptographic Schemes . . . 6
5.3.1 Commitments . . . 6
5.3.2 Hash Functions . . . 6
5.3.3 Encryption . . . 6
5.3.4 Zero-Knowledge Proving Systems . . . 6
5.3.4.1 Groth16 . . . 6
5.3.4.2 PLONK . . . 6

6 Differences from $\text{MANTA}_{\text{DAP}}$ **6**
6.1 Reusable Addresses . . . 6
6.2 Transfer Circuit Unification . . . 6

7 Acknowledgements **6**

8 References **6**

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:

$$\begin{aligned} \mathsf{None} &: \mathsf{Option}(T) \\ \mathsf{Some} &: T \to \mathsf{Option}(T) \end{aligned}$$

- 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, $y \in \left\{x \mid x \sim \mathfrak{D}(T)\right\} \leftrightarrow y \in \mathfrak{D}(T)$.

3 Concepts

3.1 Assets

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

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

The $\mathsf{MantaPay}$ protocol is a *decentralized anonymous payment* scheme which facilitiates the private ownership and private transfer of Asset objects. The $\mathsf{AssetId}$ field encodes the type of currency being used, and the $\mathsf{AssetValue}$ encodes how many units of that currency are being used, in the standard base unit of that currency.

Whenever an Asset is being used in a public setting, we simply refer to it as an Asset, but when the $\mathsf{AssetId}$ and/or $\mathsf{AssetValue}$ 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 $\mathsf{AssetValues}$ must balance the sum of the output $\mathsf{AssetValues}$, and all assets in a single transaction must have the same $\mathsf{AssetId}$[2].

3.2 Addresses

In order for participants in the $\mathsf{MantaPay}$ protocol to send and receive Assets, they must create secret and public *addresses* according to an *address scheme*. For $\mathsf{MantaPay}$, the address scheme consists of a *spending key* sk, a *viewing key* vk, and a *public key* pk. The keys have the following uses/properties:

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

See § 4.2 for more information on how these keys are constructed and used for spending, viewing, and receiving Assets.

[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 $\mathsf{AssetIds}$, like those that would be featured in a *decentralized anonymous exchange*.

3.3 Ledger

Ensuring that Assets maintain their economic value is not only dependent on transactions preserving inputs and outputs, but also that Assets are not *double-spent*. The *double-spending problem* can be solved by using a public ledger[3] that keeps track of the flow of Assets from one participant to the other. 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 ShieldedAssetPool. The ShieldedAssetPool is responsible for keeping track of the Assets which have been anonymized by the protocol.

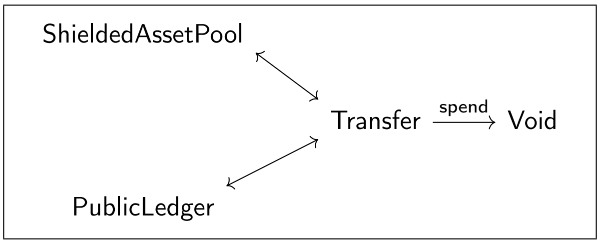

Figure 1: Lifecycle of an Asset.

Assets can be in one of three states, public (tracked by the PublicLedger), allocated (spendable subset of the ShieldedAssetPool), or spent (voided Assets). By way of the § 4.3 Transfer Protocol, Assets can be sent to and from the PublicLedger and the ShieldedAssetPool.

The ShieldedAssetPool is made up of four parts:

1. ShieldedAssetPool Balance: The MantaPay 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 value that has been anonymized by the MantaPay protocol.
2. § 3.3.1 UTXO Set: A collection of claims to subsets of the ShieldedAssetPool, each owned by participants of the MantaPay protocol.
3. § 3.3.2 Encrypted Notes: For each UTXO there is a matching encrypted note which contains information necessary to spend the Asset, which is commited in the UTXO, but 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 VoidNumber Set: A collection of commitments keeping track of those UTXOs which have participated in exactly one instance of the Transfer Protocol.

An Asset is in the public state if it belongs to the PublicLedger. An Asset is in the allocated state if a UTXO for the Asset is a member of the UTXO Set, but its matching VoidNumber is **not** in the VoidNumber Set. An Asset is in the spent state if it was allocated in the past, but its matching VoidNumber is now in the VoidNumber Set.

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

3.3.1 UTXO Set

3.3.2 Encrypted Notes

3.3.3 VoidNumber Set

4 Abstract Protocol

4.1 Abstract Cryptographic Schemes

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 $x, y : \mathsf{Input}$ and $r, s : \mathsf{Trapdoor}$ such that $x \neq y$ and $\mathsf{commit}(r, x) = \mathsf{commit}(s, y)$.

[3] A public (or private) ledger is not enough to solve 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.

- **Hiding**: For all $x, y : \mathsf{Input}$, 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 $y : \mathsf{Output}$, it is infeasible to find $x : \mathsf{Input}$ such that $\mathsf{hash}(x) = y$.
- **Collision Resistance**: It is infeasible to find an $x_1, x_2 : \mathsf{Input}$ 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 $k : \mathsf{Key}$ and plaintexts $p : \mathsf{Plaintext}$, 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 $s_1, s_2 : \mathsf{SecretKey}$, $\mathsf{agree}(s_1, \mathsf{derive}(s_2)) = \mathsf{agree}(s_2, \mathsf{derive}(s_1))$
- **TODO**: security properties

Definition 4.1.5. An *integrated encryption scheme* IES is a hybrid encryption scheme made of up a symmetric-key encryption scheme SYM and a key-agreement scheme KA, where $\mathsf{KA.SharedSecret} = \mathsf{SYM.Key}$. 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

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

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

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

  which should decrypt successfully if the 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

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

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

Definition 4.1.6. 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 Statement and 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 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 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

4.3 Transfer Protocol

4.3.1 Senders

4.3.2 Receivers

4.3.3 Transfers

4.3.4 TransferPosts

5 Concrete Protocol

5.1 Conventions

5.2 Constants

5.3 Concrete Cryptographic Schemes

5.3.1 Commitments

5.3.2 Hash Functions

5.3.3 Encryption

5.3.4 Zero-Knowledge Proving Systems

5.3.4.1 Groth16

5.3.4.2 PLONK

6 Differences from MANTA$_{\mathtt{DAP}}$

6.1 Reusable Addresses

6.2 Transfer Circuit Unification

7 Acknowledgements

8 References