

# 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 MANTADAP protocol outlined in the original [MANTA whitepaper](#).

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Notation</b>	<b>2</b>
<b>3</b>	<b>Concepts</b>	<b>2</b>
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
<b>4</b>	<b>Abstract Protocol</b>	<b>3</b>
4.1	Abstract Cryptographic Schemes	3
4.2	Addresses and Key Components	6
4.3	Transfer Protocol	6
<b>5</b>	<b>Concrete Protocol</b>	<b>7</b>
5.1	Constants	7
5.2	Concrete Cryptographic Schemes	7
5.2.1	Commitments	7
5.2.2	Hash Functions	7
5.2.3	Encryption	7
5.2.4	Zero-Knowledge Proving Systems	7
5.2.4.1	Groth16	7
5.2.4.2	PLONK	7
<b>6</b>	<b>Differences from MANTADAP</b>	<b>7</b>
6.1	Reusable Addresses	7
6.2	Transfer Circuit Unification	7
<b>7</b>	<b>Acknowledgements</b>	<b>8</b>
<b>8</b>	<b>References</b>	<b>8</b>

## 1 Introduction

**TODO:** add introductory remarks

## 2 Notation

The following notation is used throughout this specification:

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

$$\begin{aligned} \text{None} &: \text{Option}(T) \\ \text{Some} &: T \rightarrow \text{Option}(T) \end{aligned}$$

- We denote the *type of distributions* over a type  $T : \text{Type}$  as  $\mathfrak{D}(T) : \text{Type}$ . 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  $a : \text{Asset}$  is a tuple

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

The MantaPay protocol is a *decentralized anonymous payment* scheme which facilitates the private ownership and private transfer of **Asset** objects. The **AssetId** field encodes the type of currency being used, and the **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 **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**<sup>2</sup>.

### 3.2 Addresses

In order for participants in the MantaPay protocol to send and receive Assets, they must create secret and public *addresses* according to an *address scheme*. For 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.

<sup>1</sup>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.

<sup>2</sup>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*.

### 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<sup>3</sup> 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.

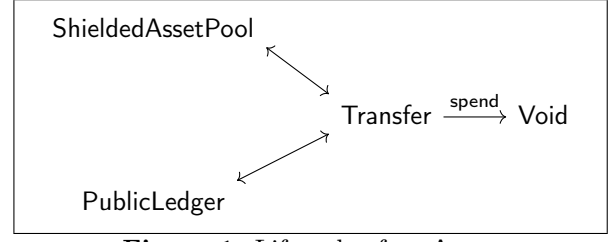


Figure 1: Lifecycle of an Asset.

**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 committed 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:

$$\begin{aligned}
 &\text{Trapdoor} : \text{Type} \\
 &\quad \text{Input} : \text{Type} \\
 &\quad \text{Output} : \text{Type} \\
 &\text{TrapdoorDistribution} : \mathcal{D}(\text{Trapdoor}) \\
 &\quad \text{commit} : \text{Trapdoor} \times \text{Input} \rightarrow \text{Output}
 \end{aligned}$$

with the following properties:

- **Binding**: It is infeasible to find an  $x, y : \text{Input}$  and  $r, s : \text{Trapdoor}$  such that  $x \neq y$  and  $\text{commit}(r, x) = \text{commit}(s, y)$ .

<sup>3</sup>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  $x, y : \text{Input}$ , the distributions  $\{\text{commit}(r, x) \mid r \sim \text{TrapdoorDistribution}\}$  and  $\{\text{commit}(r, y) \mid r \sim \text{TrapdoorDistribution}\}$  are *computationally indistinguishable*.

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

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

Input : Type  
Output : Type  
hash : Input  $\rightarrow$  Output

with the following properties:

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

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

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

Key : Type  
Plaintext : Type  
Ciphertext : Type  
encrypt : Key  $\times$  Plaintext  $\rightarrow$  Ciphertext  
decrypt : Key  $\times$  Ciphertext  $\rightarrow$  Option(Plaintext)

with the following properties:

- **Validity:** For all keys  $k : \text{Key}$  and plaintexts  $p : \text{Plaintext}$ , we have that  $\text{decrypt}(k, \text{encrypt}(k, p)) = \text{Some}(p)$ .
- **TODO:** hiding, one-time encryption security?

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

PublicKey : Type  
SecretKey : Type  
SharedSecret : Type  
derive : SecretKey  $\rightarrow$  PublicKey  
agree : SecretKey  $\times$  PublicKey  $\rightarrow$  SharedSecret

with the following properties:

- **Agreement:** For all  $s_1, s_2 : \text{SecretKey}$ ,  $\text{agree}(s_1, \text{derive}(s_2)) = \text{agree}(s_2, \text{derive}(s_1))$
- **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  $\text{KA.SharedSecret} = \text{SYM.Key}$ . We can define the following encryption/decryption algorithms:

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

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

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

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

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

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

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

**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:

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

**Notation:** We use the following notation for a ZKPS:

- We write the Statement and ProvingKey arguments of prove in the superscript and subscript respectively,

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

- We write the VerifyingKey argument of verify in the subscript,

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

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

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

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

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

$$\mathcal{A} : \text{ProvingKey} \times \text{VerifyingKey} \rightarrow \mathcal{D}(\text{PublicInput} \times \text{Proof})$$

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

$$\mathcal{E}_{\mathcal{A}} : \text{ProvingKey} \times \text{VerifyingKey} \rightarrow \mathcal{D}(\text{SecretInput})$$

such that the following probability is negligible:

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

- **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:

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

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

## 4.2 Addresses and Key Components

### 4.3 Transfer Protocol

**Definition 4.3.1.** A Sender is the following tuple:

$$\begin{aligned} \text{sk} &: \text{IES.KA.SecretKey} \times \text{IES.KA.SecretKey} \\ \tilde{r} &: \text{IES.KA.PublicKey} \\ \text{osk} &: \text{IES.KA.PublicKey} \\ \text{opk} &: \text{IES.KA.PublicKey} \\ \text{asset} &: \text{Asset} \\ \text{cm} &: \text{COM.Output} \\ \text{cm}_{\text{root}} &: \text{MerkleTree.Root} \\ \text{cm}_{\text{path}} &: \text{MerkleTree.Path} \\ \text{vn} &: \text{COM.Output} \end{aligned}$$

**Definition 4.3.2.** A SenderPost is the following tuple:

$$\begin{aligned} \text{cm}_{\text{root}} &: \text{MerkleTree.Root} \\ \text{vn} &: \text{COM.Output} \end{aligned}$$

**Definition 4.3.3.** A Receiver is the following tuple:

$$\begin{aligned} \text{pk} &: \text{IES.KA.PublicKey} \times \text{IES.KA.PublicKey} \\ r &: \text{IES.KA.SecretKey} \\ \text{opk} &: \text{IES.KA.PublicKey} \\ \text{asset} &: \text{Asset} \\ \text{cm} &: \text{COM.Output} \\ \text{note} &: \text{IES.Message} \end{aligned}$$

**Definition 4.3.4.** A ReceiverPost is the following tuple:

$$\begin{aligned} \text{cm} &: \text{COM.Output} \\ \text{note} &: \text{IES.Message} \end{aligned}$$

**Definition 4.3.5.** A Transfer is a 4-tuple of finite sets  $(T_{\text{source}}, T_{\text{sender}}, T_{\text{receiver}}, T_{\text{sink}})$  made up of Source, Sender, Receiver, and Sink tuples respectively. The *shape* of a Transfer is the 4-tuple of cardinalities of those sets

$$(|T_{\text{source}}|, |T_{\text{sender}}|, |T_{\text{receiver}}|, |T_{\text{sink}}|)$$

In order for a Transfer to be considered *valid*, it must adhere to the following constraints:

- **Same Id:** All the AssetIds in the Transfer must be equal.
- **Balanced:** The sum of input AssetValues must be equal to the sum of output AssetValues.
- **Well-formed Senders:** All of the Senders in the Transfer must be constructed according to the above Sender definition.
- **Well-formed Receivers:** All of the Receivers in the Transfer must be constructed according to the above Receiver definition.

In order to prove that these constraints are satisfied for a given Transfer, we build a zero-knowledge proof which will witness that the Transfer is valid and should be accepted by the ledger. It is not necessary to prove that the encryption of 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 4.3.6.** (Transfer Validity Statement) A transfer  $T : \text{Transfer}$  is considered *valid* if and only if

1. All the AssetIds in  $T$  are equal:

$$\left| \left( \bigcup_{x \in T_{\text{source}}} x.\text{asset.id} \right) \cup \left( \bigcup_{S \in T_{\text{sender}}} S.\text{asset.id} \right) \cup \left( \bigcup_{R \in T_{\text{receiver}}} R.\text{asset.id} \right) \cup \left( \bigcup_{x \in T_{\text{sink}}} x.\text{asset.id} \right) \right| = 1$$

2. The sum of input AssetValues is equal to the sum of output AssetValues:

$$\left( \sum_{x \in T_{\text{source}}} x.\text{asset.value} \right) + \left( \sum_{S \in T_{\text{sender}}} S.\text{asset.value} \right) = \left( \sum_{R \in T_{\text{receiver}}} R.\text{asset.value} \right) + \left( \sum_{x \in T_{\text{sink}}} x.\text{asset.value} \right)$$

3. For all  $S \in T_{\text{sender}}$ ,  $S$  is well-formed:

$$\begin{aligned} S.\text{osk} &= \dots \\ S.\text{cm} &= \text{COM}_{\text{KA.derive}(S.\text{osk})}(S.\text{asset}) \\ S.\text{vn} &= \text{COM}_{\dots}(S.\text{osk}) \\ \text{MerkleTree.contains}(S.\text{cm}, S.\text{cm}_{\text{root}}, S.\text{cm}_{\text{path}}) &= \text{True} \end{aligned}$$

4. For all  $(i, R) \in \text{enumerate}(T_{\text{receiver}})$   $R$  is well-formed at index  $i$  with respect to FAIR:

$$\begin{aligned} R.r &= \text{CRH}(i \parallel R.\text{pk} \parallel \text{FAIR}) \\ R.\text{opk} &= \dots \\ R.\text{cm} &= \text{COM}_{R.\text{opk}}(R.\text{asset}) \end{aligned}$$

where FAIR is the following fair randomness constant:

$$\text{FAIR} := \text{CRH}(\text{Concat}_{S \in T_{\text{sender}}}(S.\text{cm}))$$

**Definition 4.3.7.** A TransferPost is the following tuple:

## 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 MANTA<sub>DAP</sub>

### 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