Haven Protocol technical notes

Cypher Stack*

September 25, 2021

This document describes technical information relevant to Haven Protocol. It reflects the intention of the protocol designers, but does not reflect any particular implementation and has not been independently reviewed. This document does not reflect a comprehensive protocol specification, and should not be used as such. The author asserts no warranty and disclaims liability for its use. The author further expresses no endorsement of Haven Protocol or its associated entities.

1 Introduction

Haven Protocol aims to permit signer-ambiguous confidential transactions using a modification of the Monero transaction protocol and its associated confidential transaction model [6]. Assets in the protocol are of multiple independent types, and have a hidden value represented by a commitment. Transactions consume one or more assets of the same type, and produce new assets of arbitrary type. While individual asset values remain concealed in transactions, total input and output values for each type are revealed. An externally-provided public exchange rate is used to regulate balance validity, where a transaction is valid if (but not only if) input and output asset values balance according to this rate.

In this technical note, we provide a description of the relevant cryptography used to achieve these goals, discuss practical security, and suggest improvements and best practices to improve the protocol. However, we do not provide a formal security model or proofs for the protocol, and this note should not be interpreted as doing so.

2 Constructions

The Haven Protocol transaction model, like that of [6], integrates several cryptographic constructions. We briefly describe relevant constructions here, but refer the reader elsewhere for further details.

*https://cypherstack.com

2.1 Pedersen commitment

Amounts in assets are hidden using Pedersen commitments. Let the public parameters for a Pedersen commitment scheme be $pp_{\text{com}} = (\mathbb{G}, \mathbb{F}, G, H)$, where $\mathbb{G}$ is a prime-order group where the discrete logarithm problem is hard, $\mathbb{F}$ is its scalar field, and $G, H \in \mathbb{G}$ are randomly-sampled generators. The commitment scheme is a function $\mathsf{Com} : \mathbb{F} \times \mathbb{F} \to \mathbb{G}$ defined by $\mathsf{Com}(v, r) = vG + rH$, where we refer to v as the *value* and r as the *mask*. Pedersen commitments are perfectly hiding, computationally binding, and additively homomorphic, such that

$$\mathsf{Com}(v, r) + \mathsf{Com}(v', r') = \mathsf{Com}(v + v', r + r')$$

for all $v, v', r, r' \in \mathbb{F}$.

2.2 Discrete logarithm proofs

While not used specifically in the existing protocol, we will rely in our later recommendations on a zero-knowledge discrete logarithm proving system that asserts the prover knows the discrete logarithm of a given group element with respect to a given generator. Let the public parameters for a discrete logarithm proving system by $pp_{\text{dl}} = (\mathbb{G}, \mathbb{F})$, where $\mathbb{G}$ is a prime-order group where the discrete logarithm problem is hard, and $\mathbb{F}$ is its scalar field. The proving system consists of functions $\mathsf{DLProve}$ and $\mathsf{DLVerify}$ for the following relation:

$$\{pp_{\text{dl}}, G, Y \in \mathbb{G}; y \in \mathbb{F} : Y = yG\}$$

The simple and well-known Schnorr proving system can be used for this purpose.

2.3 Range proofs

Because balance relies on sums and differences of committed values and $\mathbb{G}$ is a prime-order group, it is necessary to assert that committed amounts are unlikely to effectively bind to negative values. To do so, we require the use of a zero-knowledge range proving system that asserts a Pedersen commitment binds to a value within a specified range. Let the public parameters for a range proving scheme be $pp_{\text{rp}} = (pp_{\text{com}}, n)$, where pp_{com} is a set of public parameters for a Pedersen commitment scheme and the desired value range is $[0, 2^n)$. The range proving scheme consists of functions $\mathsf{RangeProve}$ and $\mathsf{RangeVerify}$ for the following relation:

$$\{pp_{\text{rp}}, C \in \mathbb{G}; v, r \in \mathbb{F} : v \in [0, 2^n), C = \mathsf{Com}(v, r)\}$$

It is also possible in some instantiations to permit aggregation of independent value assertions within the same proof, which allows for improved efficiency. In this case, the range proving system is generalized to functions

AggregateRangeProve and AggregateRangeVerify for the following relation:

$$\{pp_{\text{rp}}, \{C_j\}_{j=0}^{t-1} \subset \mathbb{G}; \{(v_j, r_j)\}_{j=0}^{t-1} \subset \mathbb{F}^2 : \\ v_j \in [0, 2^n) \forall j \in [0, t), C_j = \mathsf{Com}(v_j, r_j) \forall j \in [0, t)\}$$

In practice, constructions like Bulletproofs [1] or Bulletproofs+ [2] are efficient instantiations of range proving schemes that permit value aggregation. Haven Protocol selects Bulletproofs for this purpose.

2.4 Linkable ring signature

Signer ambiguity in the context of Haven Protocol requires the use of a 2-linkable ring signature (2-LRS) scheme. In such a construction, the signer provides a list of group element pairs, and shows (in at least a witness-indistinguishable manner) that at least one such pair consists of independent Pedersen commitments to zero for which it knows the corresponding openings. Further, a linkability property asserts that attempts to generate multiple signatures with the same key can be publicly detected; in the context of the transaction protocol, this would constitute a double-spend attempt and be rejected by consensus rules. There are several security models reflecting general ring signature and linkable ring signature constructions, but we consider the definitions in [4] for this work.

Let $pp_{\text{lrs}} = (pp_{\text{com}}, N)$ be the public parameters for a 2-LRS scheme, where pp_{com} is a set of public parameters for a Pedersen commitment scheme and N is the size of the input list. The 2-LRS consists of functions LRSSign, LRSVerify, and LRSLink that respectively produce signatures, verify signatures, and determine if signatures are linked by a common signer.

Constructions like MLSAG [6], CLSAG [4], and Triptych [5] can be used for this purpose, with different tradeoffs in efficiency and applications. Haven Protocol selects CLSAG for this purpose.

Remark. *We note that the selection of range proof and 2-LRS schemes may necessitate additional requirements on the group $\mathbb{G}$ chosen. Fortunately, common elliptic curve groups satisfy typical cryptographic hardness assumptions. Haven Protocol selects the prime-order subgroup of the `ed25519` elliptic curve group for this purpose.*

3 Protocol description

We describe how Haven Protocol defines assets, as well as the proofs and signatures required to authorize their transfer.

3.1 Assets

Assets are represented by outputs, which are structures generically containing (among other data) the following:

- a public key $P \in \mathbb{G}$, the private key $p \in \mathbb{F}$ of which is required to authorize a transaction consuming the output
- a value commitment $C \in \mathbb{G}$, the value of which is encrypted to the recipient
- a type flag T identifying the asset type

3.2 Transfer transactions

Transfer transactions consume outputs of a single type T, and generate outputs of the same type. These transactions operate effectively the same as in [6]. Suppose such a transaction consumes $w \geq 1$ outputs, and generates $t > 1$ outputs. We consider the method by which the sender produces output public keys $\{P_j\}_{j=0}^{t-1}$ and other auxiliary transaction data outside the scope of this analysis, so we focus instead on value-based operations.

The sender does the following (among other steps):

1. Gathers an input cover set $\{P_i, C_i\}_{i=0}^{N-1}$ of outputs of type T such that for all $u = [0, w)$, the value l_u represents the index of a consumed output within this set. That is, we define this index such that $P_{l_u} = \mathsf{Com}(0, p_u)$ for output private key p_u, and $C_{l_u} = \mathsf{Com}(v_u, x_u)$ for known value and mask v_u and x_u.
2. Selects a fee $f \in \mathbb{F}$.
3. For each $j \in [0, t)$, produces a new generated output:
   (a) Generates the output public key $\overline{P}_j$ according to protocol-specific rules.
   (b) Chooses a value $\overline{v}_j \in \mathbb{F}$ for the output, selects a mask $\overline{x}_j \in \mathbb{F}$ uniformly at random, and computes the output value commitment $\overline{C}_j = \mathsf{Com}(\overline{v}_j, \overline{x}_j)$.
4. For each $u \in [0, w-1)$, selects a mask $x'_u \in \mathbb{F}$ uniformly at random, and computes a consumed output value commitment offset $C'_u = \mathsf{Com}(v_u, x'_u)$ to the same value as C_{l_u}.
5. Computes the final mask
$$x'_{w-1} = \sum_{j=0}^{t-1} \overline{x}_j - \sum_{u=0}^{w-2} x'_u$$
and consumed output value commitment offset $C'_{w-1} = \mathsf{Com}(v_{w-1}, x'_{w-1})$.
6. For each $u \in [0, w-1)$, generates an authorizing signature σ_u using $\mathsf{LRSSign}(pp_{\text{lrs}})$ showing that, among the set $\{P_i, C_i - C'_u\}_{i=0}^{N-1}$, the signer knows the commitment openings to both P_{l_u} and $C_{l_u} - C'_u$.
7. Generates one or more range proofs on generated outputs:

(a) If the range proving system supports aggregation, produces a single range proof on all generated outputs:

$$\Pi_{\text{rp}} = \mathsf{AggregateRangeProve}(pp_{\text{rp}}, \{\overline{C}_j\}_{j=0}^{t-1}; \{(\overline{v}_j, \overline{x}_j)\}_{j=0}^{t-1})$$

(b) Otherwise, produces a separate range proof for each $j \in [0, t)$:

$$(\Pi_{\text{rp}})_j = \mathsf{RangeProve}(pp_{\text{rp}}, \overline{C}_j; \overline{v}_j, \overline{x}_j)$$

We require specifically that the transaction balance, which means that

$$\sum_{u=0}^{w-1} v_u - \sum_{j=0}^{t-1} \overline{v}_j - f = 0.$$

To verify a transfer transaction, a verifier does the following (among other steps):

1. For each $u \in [0, w)$, uses $\mathsf{LRSVerify}$ to verify that σ_u is a valid signature.
2. Verifies any range proofs using either $\mathsf{RangeVerify}$ or $\mathsf{AggregateRangeVerify}$, as appropriate.
3. Confirms that the fee f is within an implementation-specific valid range.
4. Checks that
$$\sum_{u=0}^{w-1} C'_u - \sum_{j=0}^{t-1} \overline{C}_j - \mathsf{Com}(f, 0) = 0.$$

Since the commitment scheme is computationally binding, the validity of signatures $\{\sigma_u\}_{u=0}^{w-1}$ asserts that each C_{l_u} and C'_u bind to the same value except with negligible probability. Further, if the transaction balances, then verification step 4 holds; the converse holds except with negligible probability since the commitment scheme is computationally binding.

3.3 Conversion transactions

Conversion transactions consume outputs of a single type, but generate outputs of two different types: the consumed output type and a converted type. Suppose that such a transaction consumes $w \geq 1$ outputs. Let T represent the type flag of the consumed outputs, and T' represent the type flag of the converted outputs. Let $n_T \geq 1$ be the number of generated outputs of type T, and $n_{T'} \geq 1$ be the number of generated outputs of type T'. Let $t = n_T + n_{T'}$.

We assume the existence of a trusted external price oracle $\mathcal{O}_\$$ whose values are public and immutable. On inputs T and T', the oracle $\mathcal{O}_\$$ returns a nonzero value $r \in \mathbb{F}$ representing the conversion rate between asset values of these types. That is, a value conversion is correct if, for a rate $r = \mathcal{O}_\$(T, T')$, value v_T of type T, and value $v_{T'}$ of type T', the equation $v_{T'} = rv_T$ holds. We stress that

the assumptions on the oracle are broad, and that practical security is likely to depend highly on the properties of this oracle; for the purposes of this protocol description, we consider an abstract idealized oracle.

The sender does the following (among other steps):

1. Gathers an input cover set $\{P_i, C_i\}_{i=0}^{N-1}$ of outputs of type T such that for all $u = [0, w)$, the value l_u represents the index of a consumed output within this set. That is, we define this index such that $P_{l_u} = \mathsf{Com}(0, p_u)$ for output private key p_u, and $C_{l_u} = \mathsf{Com}(v_u, x_u)$ for known value and mask v_u and x_u.
2. Queries a conversion rate $r = \mathcal{O}_{\$}(T, T')$.
3. Selects a fee f in T units.
4. For each $j \in [0, n_T)$, produces a new generated output of type T:
   (a) Generates the output public key $\overline{P}_j$ according to protocol-specific rules.
   (b) Chooses a value $\overline{v}_j \in \mathbb{F}$ in T units for the output, selects a mask $\overline{x}_j \in \mathbb{F}$ uniformly at random, and computes the output value commitment $\overline{C}_j = \mathsf{Com}(\overline{v}_j, \overline{x}_j)$.
5. For each $j \in [n_T, t)$, produces a new generated output of type T':
   (a) Generates the output public key $\overline{P}_j$ according to protocol-specific rules.
   (b) Chooses a value $\overline{v}_j \in \mathbb{F}$ in T' units for the output, selects a mask $\overline{x}_j \in \mathbb{F}$ uniformly at random, and computes the output value commitment $\overline{C}_j = \mathsf{Com}(\overline{v}_j, \overline{x}_j)$.
6. For each $u \in [0, w-1)$, selects a mask $x'_u \in \mathbb{F}$ uniformly at random, and computes a consumed output value commitment offset $C'_u = \mathsf{Com}(v_u, x'_u)$ to the same value as C_{l_u}.
7. Computes the final mask
$$x'_{w-1} = \sum_{j=0}^{n_T-1} \overline{x}_j + \frac{1}{r} \sum_{j=n_T}^{t-1} \overline{x}_j - \sum_{u=0}^{w-2} x'_u$$
and consumed output value commitment offset $C'_{w-1} = \mathsf{Com}(v_{w-1}, x'_{w-1})$.
8. For each $u \in [0, w-1)$, generates an authorizing signature σ_u using $\mathsf{LRSSign}(pp_{\text{lrs}})$ showing that, among the set $\{P_i, C_i - C'_u\}_{i=0}^{N-1}$, the signer knows the commitment openings to both P_{l_u} and $C_{l_u} - C'_u$.
9. Generates one or more range proofs on generated outputs:

(a) If the range proving system supports aggregation, produces a single range proof on all generated outputs:

$$\Pi_{\text{rp}} = \mathsf{AggregateRangeProve}(pp_{\text{rp}}, \{\overline{C}_j\}_{j=0}^{t-1}; \{(\overline{v}_j, \overline{x}_j)\}_{j=0}^{t-1})$$

(b) Otherwise, produces a separate range proof for each $j \in [0, t)$:

$$(\Pi_{\text{rp}})_j = \mathsf{RangeProve}(pp_{\text{rp}}, \overline{C}_j; \overline{v}_j, \overline{x}_j)$$

10. Generates the following commitment mask sums:

$$a_T = \sum_{u=0}^{w-1} x'_u$$
$$\overline{a}_T = \sum_{j=0}^{n_T-1} \overline{x}_j$$

11. Computes the net value sums in each asset type:

$$y_T = \sum_{u=0}^{w-1} v_u - \sum_{j=0}^{n_T-1} \overline{v}_j - f$$
$$y_{T'} = \sum_{j=n_T}^{t-1} \overline{v}_j$$

In this case, transaction balance requires that

$$\sum_{u=0}^{w-1} v_u - \sum_{j=0}^{n_T-1} \overline{v}_j - \frac{1}{r} \sum_{j=n_T}^{t-1} \overline{v}_j - f = 0.$$

To verify a conversion transaction, a verifier does the following (among other steps):

1. Queries the conversion rate $r = \mathcal{O}_{\$}(T, T')$ appropriate to the transaction.
2. For each $u \in [0, w)$, uses $\mathsf{LRSVerify}$ to verify that σ_u is a valid signature.
3. Verifies any range proofs using either $\mathsf{RangeVerify}$ or $\mathsf{AggregateRangeVerify}$ as appropriate.
4. Confirms that the fee f is within an implementation-specific valid range.
5. Confirms the transaction balances by checking that

$$\sum_{u=0}^{w-1} C'_u - \sum_{j=0}^{n_T-1} \overline{C}_j - \frac{1}{r} \sum_{j=n_T}^{t-1} \overline{C}_j - \mathsf{Com}(f, 0) = 0.$$

6. Confirms the type-T amount by checking that

$$\sum_{u=0}^{w-1} C'_u - \sum_{j=0}^{n_T-1} \overline{C}_j - \mathsf{Com}(f, 0) - \mathsf{Com}(0, a_T - \overline{a}_T) - \mathsf{Com}(y_T, 0) = 0.$$

7. Confirms the plaintext type-T' amount by checking that $y_{T'} - r y_T = 0$.

If the transaction balances, then verification steps 5, 6, and 7 hold; the converses hold except with negligible probability since the commitment scheme is computationally binding.

4 Observations and recommendations

In this section, we make observations and propose changes to the protocol to improve its practical security.

4.1 Remove the plaintext type-T' amount check

While the type-T amount y_T is verified in conversion transactions by testing its relationship to the relevant commitments in verification step 6, the type-T' amount $y_{T'}$ is never tested against commitments. Rather, it only appears in the check in verification step 7, where its presence in the transaction structure and algorithms is unnecessary and offers no additional security. An adversary who can produce a transaction that passes verification steps 5 and 6, but produce type-T' commitments to an unexpected total value, would break the binding property of the commitment scheme. The value $y_{T'}$ and associated check in step 7 can therefore be safely removed, and this value can simply be computed by the verifier as needed for other purposes.

4.2 Reveal a single aggregate type-T commitment mask

In conversion transaction generation step 10, mask sums a_T and $\overline{a}_T$ are computed. However, in verification step 6, only their difference $a_T - \overline{a}_T$ is used, yielding no additional security by including both values in transaction data. Since no additional checks are required or performed on the individual values, only the difference needs to be included in the transaction data.

4.3 Replace aggregate type-T commitment masks with representation proof

In conversion transactions, commitment mask sums are uniformly distributed, but they are not independent. In particular, the mask equation

$$\sum_{u=0}^{w-1} x'_u - \sum_{j=0}^{n_T-1} \overline{x}_j - \frac{1}{r} \sum_{j=n_T}^{t-1} \overline{x}_j = 0$$

holds. When a user produces a conversion transaction that reveals mask terms, it can leak information that depends on the particular transaction input and output structure. To mitigate this leakage, it is possible to modify the protocol to instead prove knowledge of the commitment mask sums, rather than reveal them. Because amount verification requires the use of these sums, the check in verification step 6 also must be slightly modified.

To do so, generation step 10 is replaced. Instead of revealing the mask sums a_T and $\overline{a}_T$ in the transaction data, the prover instead defines

$$A = \mathsf{Com}\left(0, \sum_{u=0}^{w-1} x'_u - \sum_{j=0}^{n_T-1} \overline{x}_j\right)$$

and produces a discrete logarithm proof

$$\Pi_{\mathrm{dl},T} = \mathsf{DLProve}\left(pp_{\mathrm{dl}}, \mathsf{Com}(0,1), A; \sum_{u=0}^{w-1} x'_u - \sum_{j=0}^{n_T-1} \overline{x}_j\right)$$

on this value, and includes $\Pi_{\mathrm{dl},T}$ in the transaction.

Verification step 6 is then modified to account for this change. The verifier defines

$$A' = \sum_{u=0}^{w-1} C'_u - \sum_{j=0}^{n_T-1} \overline{C}_j - \mathsf{Com}(f,0) - \mathsf{Com}(y_T,0)$$

and checks that $\mathsf{DLVerify}(pp_{\mathrm{dl}}, \mathsf{Com}(0,1), A', \Pi_{\mathrm{dl},T})$ succeeds.

4.4 Select all commitment masks independently

The balance equations used in the protocol (for both transfer and conversion transactions) relies on commitment masks being chosen in such a way that input commitment offsets and output commitments sum to zero. While this means that such masks can be distributed uniformly at random, they are not chosen independently for this reason. An adversary who knows all but one of these masks can therefore identify the remaining mask and corresponding value. Other transaction structures and components can also leak information.

Similarly to the above notes on type-T commitment mask sums, it is possible to choose all commitment masks independently and prove knowledge of them in zero knowledge. This means that knowledge of any particular subset of masks gives an adversary no inherent advantage in identifying unknown values.

4.4.1 Transfer transactions

To do this for transfer transactions, generation step 5 is removed, and step 4 is modified to select all masks uniformly at random independently. Because verifier step 4 no longer holds, the prover defines

$$B = \mathsf{Com}\left(0, \sum_{u=0}^{w-1} x'_u - \sum_{j=0}^{t-1} \overline{x}_j\right)$$

and produces a discrete logarithm proof

$$\Pi_{\text{dl}} = \mathsf{DLProve}\left(pp_{\text{dl}}, \mathsf{Com}(0,1), B; \sum_{u=0}^{w-1} x'_u - \sum_{j=0}^{t-1} \overline{x}_j\right)$$

on this value, and includes Π_{dl} in the transaction.

The verification step 4 is then modified to account for this change. The verifier defines

$$B' = \sum_{u=0}^{w-1} C'_u - \sum_{j=0}^{t-1} \overline{C}_j - \mathsf{Com}(f, 0)$$

and checks that $\mathsf{DLVerify}(pp_{\text{dl}}, \mathsf{Com}(0,1), B', \Pi_{\text{dl}})$ succeeds.

4.4.2 Conversion transactions

To do this for conversion transactions, generation step 7 is removed, and step 6 is modified to select all masks uniformly at random independently. Because verifier step 5 no longer holds, the prover defines

$$B = \mathsf{Com}\left(0, \sum_{u=0}^{w-1} x'_u - \sum_{j=0}^{n_T-1} \overline{x}_j - \frac{1}{r} \sum_{j=n_T}^{t-1} \overline{x}_j\right)$$

and produces a discrete logarithm proof

$$\Pi_{\text{dl}} = \mathsf{DLProve}\left(pp_{\text{dl}}, \mathsf{Com}(0,1), B; \sum_{u=0}^{w-1} x'_u - \sum_{j=0}^{n_T-1} \overline{x}_j - \frac{1}{r} \sum_{j=n_T}^{t-1} \overline{x}_j\right)$$

on this value, and includes Π_{dl} in the transaction.

The verification step 5 is then modified to account for this change. The verifier defines

$$B' = \sum_{u=0}^{w-1} C'_u - \sum_{j=0}^{n_T-1} \overline{C}_j - \frac{1}{r} \sum_{j=n_T}^{t-1} \overline{C}_j - \mathsf{Com}(f, 0)$$

and checks that $\mathsf{DLVerify}(pp_{\text{dl}}, \mathsf{Com}(0,1), B', \Pi_{\text{dl}})$ succeeds.

4.5 Aggregate all discrete logarithm proofs

The recommendations in sections 4.3 and 4.4 each produce a proof of knowledge of a discrete logarithm against the same group generator, which can be considered proofs of commitments to zero. To optimize space and verifier efficiency, it may be desirable to combine these proofs. Doing so requires the use of an aggregate discrete logarithm proving system, which asserts knowledge of openings to multiple independent commitments to zero in zero knowledge.

Let the public parameters of an aggregate discrete logarithm proving system be $pp_{\text{agg}} = (pp_{\text{com}}, d)$, where pp_{com} is a set of public parameters for a Pedersen commitment scheme and $d > 0$ is the number of commitments. The proving scheme consists of functions $\mathsf{AggregateDLProve}$ and $\mathsf{AggregateDLVerify}$ for the following relation:

$$\{pp_{\text{agg}}, \{Y_i\}_{i=0}^{d-1} \subset \mathbb{G}; \{y_i\}_{i=0}^{d-1} \subset \mathbb{F} : Y_i = \mathsf{Com}(0, y_i) \forall i \in [0, d)\}$$

The batching construction of Gennaro *et al.* [3] is an efficient and straightforward instantiation of such a proving system. Proving knowledge of openings of $d > 1$ commitments to zero incurs the same communication cost as for $d = 1$ commitments, and the additional computational complexity is minimal.

For completeness, we list the algorithm steps here; the reader is referred to [3] for details and security proofs.

1. The prover chooses $r \in \mathbb{F}$ uniformly at random, sets $X = r\mathsf{Com}(0, 1)$, and sends X to the verifier.
2. The verifier chooses $c \in \mathbb{F} \setminus \{0\}$ uniformly at random, and sends c to the prover.
3. The prover sets $s = r + \sum_{i=0}^{d-1} y_i c^{i+1}$, and sends s to the verifier.
4. The verifier checks that the equality

$$X + \sum_{i=0}^{d-1} c^{i+1} Y_i = s\mathsf{Com}(0, 1)$$

holds.

If applying both recommendations that use discrete logarithm representation proofs, the goal is to combine the two proofs $\Pi_{\text{dl},T}$ and Π_{dl} into a single proof Π_{agg} using this method. In this case, we use the notation from the previous sections and produce the proof

$$\Pi_{\text{agg}} = \mathsf{AggregateDLProve}(pp_{\text{agg}}, \{Y_0, Y_1\}; \{y_0, y_1\}),$$

where

$$y_0 = \sum_{u=0}^{w-1} x'_u - \sum_{j=0}^{n_T-1} \overline{x}_j$$

and $Y_0 = A$ are the witness and statement from $\Pi_{\text{dl},T}$ in section 4.3, and

$$y_1 = \sum_{u=0}^{w-1} x'_u - \sum_{j=0}^{n_T-1} \overline{x}_j - \frac{1}{r} \sum_{j=n_T}^{t-1} \overline{x}_j$$

and $Y_1 = B$ are the witness and statement from Π_{dl} in section 4.4.

Verification of this single aggregate proof with $\mathsf{AggregateDLVerify}$ uses the corresponding verification statements described in those sections.

References

[1] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 315–334, 2018.

[2] Heewon Chung, Kyoohyung Han, Chanyang Ju, Myungsun Kim, and Jae Hong Seo. Bulletproofs+: Shorter proofs for privacy-enhanced distributed ledger. Cryptology ePrint Archive, Report 2020/735, 2020.

[3] R. Gennaro, D. Leigh, R. Sundaram, and W. Yerazunis. Batching Schnorr identification scheme with applications to privacy-preserving authorization and low-bandwidth communication devices. In Pil Joong Lee, editor, *Advances in Cryptology - ASIACRYPT 2004*, pages 276–292, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[4] Brandon Goodell, Sarang Noether, and RandomRun. Concise linkable ring signatures and forgery against adversarial keys. Cryptology ePrint Archive, Report 2019/654, 2019.

[5] Sarang Noether and Brandon Goodell. Triptych: Logarithmic-sized linkable ring signatures with applications. In Joaquin Garcia-Alfaro, Guillermo Navarro-Arribas, and Jordi Herrera-Joancomarti, editors, *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pages 337–354, Cham, 2020. Springer International Publishing.

[6] Shen Noether, Adam Mackenzie, et al. Ring confidential transactions. *Ledger*, 1:1–18, 2016.