Lin2-Xor Lemma and Log-size Linkable Ring Signature

Anton A. Sokolov

acmxddk@gmail.com

Abstract *In this paper we introduce a novel method for constructing an efficient linkable ring signature without a trusted setup in a group where decisional Diffie-Hellman problem is hard and no bilinear pairings exist. Our linkable ring signature is logarithmic in the size of the signer anonymity set, its verification complexity is linear in the anonymity set size and logarithmic in the signer threshold. A range of the recently proposed setup-free logarithmic size signatures is based on the commitment-to-zero proving system by Groth and Kohlweiss or on the Bulletproofs inner-product compression method by Bünz et al. In contrast, we construct our signature from scratch using the Lin2-Xor and Lin2-Selector lemmas that we formulate and prove here. With these lemmas we construct an n-move public coin special honest verifier zero-knowledge membership proof protocol and instantiate the protocol in the form of a general-purpose setup-free signer-ambiguous linkable ring signature in the random oracle model.*

Keywords: Ring signature, linkable ring signature, log-size signature, membership proof, signer-ambiguity, zero-knowledge, disjunctive proof.

1 INTRODUCTION

In simple words, the problem is to sign a message m in such a way as to convince a verifier that someone out of a group of possible signers has actually signed the message, without revealing the signer identity. A group of signers is called a ring. It could be required that L signers sign a message, L is a threshold in this case.

As an extension, it could be required that every signer can sign only once, in this case the signature is called linkable. It is also desirable that the signature size and verification complexity are to be minimal.

An effective solution to this problem plays a role in cryptographic applications, for instance, in the telecommunication and peer-to-peer distributed systems.

A formal notion of ring signatures and the early yet efficient schemes are presented in the works of Rivest, Shamir, and Tauman [15], Abe, Ohkubo, and Suzuki [1], Liu, Wei, and Wong [13], an example of a system that uses linkable ring signatures is, for instance, CryptoNote [17]. Nice properties of the schemes are that there is no trusted setup process and no selected entities in them, an actual signer is able to frequently change its anonymity set without ever notifying the other participants about this.

The schemes in [1, 13] and other linkable ring signature schemes can be instantiated with a prime-order cyclic group under the discrete logarithm problem hardness (DL) assumption. Scheme security and the signer anonymity are usually, e.g., as in [13], reduced to one of the stronger hardness assumptions, for instance, to the decisional Diffie-Hellman (DDH) assumption in the random oracle model (ROM).

All these signatures have sizes that grow linearly in the signer anonymity set size. Their verification complexities are linear, too.

Recent works by Tsz Hon Yuen, Shi feng Sun, Joseph K. Liu, Man Ho Au, Muhammed F. Esgin, Qingzhao Zhang, and Dawu Gu [18], Sarang Noether [14], Benjamin E. Diamond [2], Russell W. F. Lai, Viktoria Ronge, Tim Ruffing, Dominique Schröder, Sri Aravinda Krishnan Thyagarajan, and Jiafan Wang [12], William Black and Ryan Henry [3], and others show that under the common assumptions for a prime-order cyclic group where the DL is hard and, maybe, with some rather natural assumptions about the participating public keys, it's possible to build a setup-free linkable ring signature with logarithmic size.

As another line of solutions, in the works of Jens Groth [9], Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox [11] and some others it is shown that signer-ambiguous signatures with asymptotically lower sizes and verification complexities can be built at the cost of requiring a trusted setup or bilinear pairings to the prime-order group. However, this line of solutions is out of the scope of our current work.

In this paper we construct a setup-free logarithmic-size linkable ring signature scheme over a prime-order cyclic group without bilinear pairings under the DDH assumption in the ROM.

1.1 CONTRIBUTION

1.1.1 LIN2-XOR AND LIN2-SELECTOR LEMMAS

We formulate and prove Lin2-Xor lemma that allows for committing to exactly one pair of elements out of two pairs of elements.

Using the Lin2-Xor lemma as a disjunction unit, we formulate and prove Lin2-Selector lemma that allows for committing to exactly one pair of elements out of many pairs of elements.

The Lin2-Selector lemma provides a pure n-move public coin protocol that, being successfully played between any prover and an honest verifier, convinces the verifier that the prover knows an opening (k_0, k_1, s) of a commitment Z, where the commitment Z has a form

$$Z = k_0 P_s + k_1 Q_s,$$

where the pair (P_s, Q_s), $s \in [0, N-1]$, is taken from a publicly known set of element pairs $\left\{(P_j, Q_j)\right\}_{j=0}^{N-1}$ such that there is no known discrete logarithm relationship between any elements in the set.

We show, that the amount of data transmitted from a prover to a verifier during the Lin2-Selector protocol execution is logarithmic in the size of the publicly known set of element pairs.

With the Lin2-Selector lemma, no additional proof is required for that the commitment has the form $k_0 P_s + k_1 Q_s$. Once the lemma's pure n-move public coin protocol is successfully completed, the verifier is convinced of both the form $Z = k_0 P_s + k_1 Q_s$ and the prover's knowledge of (k_0, k_1, s).

The Lin2-Xor and Lin2-Selector lemmas are proven for a prime-order group under the DL hardness assumption.

1.1.2 L2S SET MEMBERSHIP PROOF PROTOCOL

We construct an n-move public coin L2S set membership proof protocol on the base of the Lin2-Selector lemma pure n-move public coin protocol.

The L2S protocol inherits the properties of the Lin2-Selector lemma pure protocol and, thus, convinces a verifier that a commitment $Z = k_0 P_s + k_1 Q_s$ is built over a member (P_s, Q_s) of a set of element pairs with unknown discrete logarithm relationship between the elements from all the pairs.

We prove the L2S protocol is complete and sound under the DL, special honest verifier zero-knowledge (sHVZK) under the DDH.

1.1.3 SIGNER-AMBIGUOUS ML2SLNKSIG LINKABLE RING SIGNATURE

Using the L2S membership proof protocol we construct a non-interactive zero-knowledge many-out-of-many mL2SHPoM membership proof scheme and, consequently, construct a many-out-of-many mL2SLnkSig logarithmic-size linkable ring signature.

Compared to the setup-free log-size linkable ring signature schemes proposed in [18, 14, 2, 12], that originate from the ideas of Jens Groth and Markulf Kohlweiss [10], Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell [5], our signature scheme is constructed on a basis different from [10, 5].

A parallel can be drawn with the work of Jens Groth and Markulf Kohlweiss [10]. A mechanism resembling the Kronecker's delta is introduced in [10] for selecting an anonymity set member without revealing it. Our signature uses the Lin2-Selector lemma exactly in the same role. There is a difference in the anonymity sets: the anonymity sets in [10] lay in a plain built over the homomorphic commitment generators, whereas the anonymity sets for the Lin2-Selector lemma protocol are sets of orthogonal generators.

Requiring the anonymity sets to be the sets of orthogonal generators for the mL2SHPoM scheme, we completely drop this limitation for the mL2SLnkSig signature scheme.

We present our mL2SLnkSig signature scheme as a general-purpose log-size solution for the linkable ring signature problem, when the anonymity set is allowed to be an arbitrary set of distinct public keys.

The mL2SLnkSig signature is signer-ambiguous under the DDH in the ROM, it keeps this property even for the cases when relationship between the public keys is known to an adversary.

1.2 METHOD OVERVIEW

1.2.1 LIN2 LEMMA

In a nutshell, firstly we formulate and prove a helper lemma, that connects Z to the P and Q in the equation

$$w(P + cQ) = Z + rH,$$

where Z, H, P, Q are fixed elements of a primary-order group where DL is hard, c is a verifier's challenge, r is a prover's reply, and w is a non-zero scalar known to the prover.

The lemma states that if no discrete logarithm relationship between P and Q is known, if the prover is able to reply with a scalar r to a random challenge c and, in addition to this, if it is able to show that the above equation holds for some known to it and possibly secret w, then the scalars a and b in the equality

$$Z = aP + bQ$$

are certainly known to the prover.

1.2.2 LIN2-XOR LEMMA

Next, we consider a linear combination R of four fixed primary-order group elements P_1, Q_1, P_2, Q_2 with unknown discrete logarithm relationship to each other

$$R = c_{20}\left(c_{10}P_1 + c_{11}Q_2\right) + c_{21}\left(c_{12}P_2 + c_{13}Q_2\right),$$

where c_{11}, c_{13}, c_{21} are random scalars, and c_{10}, c_{12}, c_{20} are always equal to 1. That is, reducing the constant coefficients, we consider the following linear combination

$$R = \left(P_1 + c_{11}Q_2\right) + c_{21}\left(P_2 + c_{13}Q_2\right).$$

It appears that if a prover demonstrates a pair of fixed elements (Z, H_1) at the beginning of a protocol, receives a pair of random challenges (c_{11}, c_{13}) from a verifier, replies with a scalar r_1 and with an element H_2, receives a random challenge c_{21} after that, replies with a scalar r_2, and finally shows that the equation

$$wR = Z + r_1H_1 + r_2H_2$$

holds for some secretly known non-zero scalar w, then Z has the following property: it equals to exactly one of $(aP_1 + bQ_1)$ and $(aP_2 + bQ_2)$ for some known to the prover scalars a, b. We formulate this property and the necessary conditions as Lin2-Xor lemma. The key condition is that (Z, H_1) are to be chosen without knowing the (c_{11}, c_{13}, c_{21}), and (r_1, H_2) are to be chosen without knowing c_{21}.

In other words, the Lin2-Xor lemma states that the above protocol successfully completes only if the prover knows scalars a and b such that

$$(Z = aP_1 + bQ_1) \text{ or } (Z = aP_2 + bQ_2).$$

After the Lin2-Xor lemma protocol successful completion the verifier is convinced that Z is a linear combination of either (P_1, Q_1) or (P_2, Q_2). There exists no possibility for Z to be a linear combination of all the four elements, e.g., like $Z = aP_1 + bP_2 + dQ_1 + eQ_2$ with known to the prover non-zero a, b, d, e.

Moreover, after the protocol successful completion the verifier is convinced that H_1 is also a linear combination of either (P_1, Q_1) or (P_2, Q_2), that is, H_1 possesses a similar property:

$$(H_1 = fP_1 + gQ_1) \text{ or } (H_1 = fP_2 + gQ_2)$$

for some known to the prover f, g.

1.2.3 COROLLARY OF LIN2-XOR LEMMA

As a corollary we get that if the Lin2-Xor lemma protocol is successfully completed, then the verifier is convinced that the prover knows some secret scalar x such that

$$(Z + r_1H_1 = x(P_1 + c_{11}Q_1)) \text{ or } (Z + r_1H_1 = x(P_2 + c_{13}Q_2)).$$

1.2.4 LIN2-SELECTOR LEMMA

It appears that the Lin2-Xor lemma can be 'stacked', i.e., applied a number of times to an arbitrary number of fixed orthogonal elements. We assume the number of elements is a power of 2.

For instance, for eight fixed orthogonal elements P_1, Q_1, P_2, Q_2, P_3, Q_3, P_4, Q_4 and for two fixed elements Z, H_1:

$$\begin{aligned} R &= \left(\left(P_1 + c_{11}Q_1\right) + c_{21}\left(P_2 + c_{13}Q_2\right)\right) + c_{31}\left(\left(P_3 + c_{11}Q_3\right) + c_{23}\left(P_4 + c_{13}Q_4\right)\right), \\ wR &= Z + r_1H_1 + r_2H_2 + r_3H_3, \end{aligned}$$

where the (c_{11}, c_{13}) is the first challenge and the (r_1, H_2) is the first reply, the (c_{21}, c_{23}) is the second challenge and the (r_2, H_3) is the second reply, the c_{31} is the third challenge and the r_3 is the third reply, Lin2-Selector lemma provides a method to convince verifier that Z is exactly one of $(aP_1 + bQ_1)$, $(aP_2 + bQ_2)$, $(aP_3 + bQ_3)$, $(aP_4 + bQ_4)$ for some known to prover a, b.

Namely, applying the Lin2-Xor lemma and its corollary we can find that exactly one equality of the following two

$$(Z + r_1 H_1 + r_2 H_2) = x((P_1 + c_{11} Q_1) + c_{21}(P_2 + c_{13} Q_2)),$$
$$(Z + r_1 H_1 + r_2 H_2) = x((P_3 + c_{11} Q_3) + c_{23}(P_4 + c_{13} Q_4))$$

holds for some known to the prover x. Applying the Lin2-Xor lemma to that equality that holds, suppose, to the first one, we find that Z is exactly one of $(aP_1 + bQ_1)$, $(aP_2 + bQ_2)$ for some known to the prover a, b.

For a set of 2^{n-1} pairs $\{(P_j, Q_j)\}_{j=0}^{2^{n-1}-1}$, the Lin2-Selector lemma provides a general method and a protocol for constructing R such that

$$wR = Z + \sum_{i=1 \ldots n} r_i H_i,$$

where the verifier is convinced that $Z = k_0 P_s + k_1 Q_s$ for some known to prover $s \in [0, 2^{n-1} - 1]$, k_0, k_1. The actual s can be made indistinguishable by keeping the scalars k_0 and k_1 in secret.

1.2.5 PURE PROTOCOLS AND SOUNDNESS

Overall, the following three lemmas: Lin2, Lin2-Xor, and Lin2-Selector have similar structure of their premises and conclusions in our work.

The structure is: a premise declares necessary assumptions about the publicly seen values and defines, as we call it, a pure protocol. A conclusion is that if the assumptions hold and the protocol is successfully completed, then verifier is convinced that prover knows some secret values.

A pure protocol specifies what the verifier has to do in detail, however, it doesn't specify the same for the prover. It describes only what the prover has to reply to the verifier without specifying how it prepares the replies.

With this structure we are able to prove the pure protocol soundness, i.e., that the successful protocol completion implies the prover's knowledge of the secret values. The Lin2, Lin2-Xor, and Lin2-Selector lemmas provide the proofs of soundness for their pure protocols.

We don't consider completeness and zero-knowledge for the pure protocols, as these properties depend on how the prover prepares the replies.

If a pure protocol soundness is proven, then a derived protocol defining prover's behavior in details inherits the soundness. Once the prover's behavior is completely defined in the derived protocol, we consider completeness and zero-knowledge for it.

1.2.6 SOUNDNESS PROOFS FOR THE PURE PROTOCOLS

We assume the prover and verifier are probabilistic polynomial-time Turing machines equipped with a common tape, where they write their conversation transcript.

When we prove soundness for a pure protocol, the verifier is assumed honest, whereas the prover is regarded as having a dishonest subroutine, that produces with overwhelming probability acceptable replies to the uniformly random challenges, such that the protocol completes successfully.

To prove soundness for the Lin2 lemma protocol we suppose that the secret values in question are not known to the prover. We consider two successful Lin2 lemma protocol transcripts, one of which is that of the prover-verifier conversation, and the other one is that the prover is capable of obtaining itself by calling the dishonest subroutine for another set of challenges taken from its random tape. We demonstrate a polynomial-time algorithm that extracts the secret values in question from these two transcripts using known to the prover information. Thus, we show that even not knowing these values the prover is capable of getting them in polynomial time once it is able to successfully complete the protocol. Therefore, the protocol is sound.

We use the same method for the Lin2-Xor lemma protocol, with the only difference that we don't demonstrate a polynomial-time extractor algorithm, we gradually find which values can be obtained by the prover in polynomial time and finally show that the values in question are among them. That is, we prove that a polynomial-time extactor can be built by the prover.

For the Lin2-Selector lemma protocol we do the same using the Lin2 and Lin2-Xor lemmas. Thus we prove the protocol soundness.

1.2.7 L2S MEMBERSHIP PROTOCOL, ML2SLNKSIG SIGNATURE

We construct L2S set membership proof protocol on top of the Lin2-Selector lemma pure protocol and prove that the L2S protocol is complete and sound, obtaining the soundness directly from the Lin2-Selector lemma.

Next, we analyze L2S protocol transcript and show that all its entries have distributions indistinguishable from independent and uniform randomness, except for one entry which is a linear combination of the other entries. From this, we show that the protocol is sHVZK using the definition and method by Ronald Cramer, Ivan Damgård, and Berry Schoenmakers [6] and, consequently, that it doesn't reveal any information beyond the fact of membership. This allows us to build a signer-ambiguous signature on its base.

The L2S protocol is efficient, it requires transmitting one Z and $n\,(r_i, H_i)$ pairs, and computing one multi-exponentiation for 2^n summands when calculating R during verification. Overall, in all schemes and protocols in this paper the value R is calculated only once during verification.

Using the Fiat-Shamir heuristic, we turn the L2S protocol to the mL2SHPoM non-intaractive many-out-of-many proof of membership scheme and to the mL2SLnkSig many-out-of-many linkable ring signature scheme with a linking tag in the form $x^{-1}\mathbf{H_{point}}\,(P)$, where $P = xG$ and $\mathbf{H_{point}}$ is a hash function on the group elements.

While the mL2SHPoM proof of membership scheme requires all elements of its anonymity set to be orthogonal to each other, the mL2SLnkSig scheme removes this limitation by 'lifting' the anonymity set to an orthogonal set of images of an $\mathbf{H_{point}}$-based hash function and then applying the mL2SHPoM to that orthogonal set.

2 PRELIMINARIES

- Let $\mathbb{G}$ be a cyclic group of prime order in which the discrete logarithm problem is hard, and let $\mathbb{F}$ be a scalar field of $\mathbb{G}$. The field $\mathbb{F}$ is finite, of the same order as $\mathbb{G}$.
- Let lowercase italic letters and words a, b, sum, ... designate scalars in $\mathbb{F}$. Sometimes indices and apostrophes are appended: a_{12}, b', sum_1, ... Also, lowercase italic letters and words can be used to designate integers used as indices, e.g., i, j_1, idx_1, ... , this usage is clear from the context.
- Let the uppercase italic letters and words $A, B, X, P, H, \ldots$ denote the elements of $\mathbb{G}$. Indices and apostrophes can be appended: $A_1, B', X_{12}, P_{11}, H_1, \ldots$. Also, italic uppercase letters denote sets and integers that is clear from the context. The letters N and M are reserved for integer powers of 2.
- Let 0 denote the zero element of $\mathbb{G}$ and also denote the zero scalar in $\mathbb{F}$, it's easy to distinguish its meaning from the context.
- Let G be a generator of $\mathbb{G}$. As $\mathbb{G}$ is a prime-order group, any non-zero element A is a generator of $\mathbb{G}$, hence we assume G is an a-priory chosen element.

2.1 A NOTE ABOUT CONTEXT

All definitions and lemmas below are given in the context of a game between Prover and Verifier, unless otherwise stated.

During the game Prover tries to convince Verifier that certain facts are true. For the sake of this, Prover may disclose some information to Verifier, the latter may pick some, e.g., random, challenges, send them to Prover and get some values back from it.

The game may contain a number of subsequent protocols. That is, Prover and Verifier may execute protocols between each other a number of times, so that Verifier gradually becomes convinced of the facts.

A protocol can be translated to a non-interactive scheme using the Fiat-Shamir heuristic in the ROM. We start with proving the lemmas in the interactive setting, next they are turned into the non-interactive setting with the Fiat-Shamir heuristic.

2.2 DEFINITIONS

2.2.1 SECURITY PARAMETER

We assume security parameter $\boldsymbol{\lambda}$ is equal to the logarithm of cardinality of $\mathbb{F}$. The cardinalities of $\mathbb{F}$ and $\mathbb{G}$ are equal to each other, so $\boldsymbol{\lambda}$ is equal to the logarithm of cardinality of $\mathbb{G}$

We omit mentioning $\boldsymbol{\lambda}$ in the protocols, implying polynomial time is a polynomial time in $\boldsymbol{\lambda}$ everywhere.

2.2.2 SETS AND VECTORS

Sets are assumed having cardinalities that are polynomial in $\boldsymbol{\lambda}$ everywhere. Vectors are ordered sets.

Sets are denoted by uppercase italic letters or curly brackets. Vectors of scalars or elements are denoted using either square brackets [] or arrows over italic lowercase or uppercase letters, respectively: $\vec{x}$, $\vec{X}$.

Brackets can be omitted where it is not ambiguous, e.g., if $S = \{B_1, B_2, \ldots, B_n\}$, then the sequence $B_1, B_2, \ldots, B_n$ represents the same set S.

2.2.3 KNOWN AND UNKNOWN DISCRETE LOGARITHM RELATIONSHIP

For any two elements A and B, the notation

$$A \sim B$$

designates the fact of a known discrete logarithm relationship between A and B, that is, in the equation $A = xB$ the scalar x is known or can be efficiently calculated.

The term "efficiently calculated" means that a probabilistic polynomial-time algorithm (PPT) solving the problem with non-negligible probability can be demonstrated. As all sets in our paper have polynomial cardinality, and as all proofs have polynomial number of steps, we consider the terms "efficiently calculated" and "known" as carrying the same meaning elsewhere.

If calculating x in the equation $A = xB$ is hard, then a discrete logarithm relationship between A and B is unknown, this fact is designated as

$$A \,!\!\sim B.$$

For any A and B, both $A \sim B$ and $A \,!\!\sim B$ never hold. It's not required for the statements $A \sim B$ and $A \,!\!\sim B$ to obey the law of excluded middle, the only assumed law and implication are:

- (not ($A \sim B$ and $A \,!\!\sim B$)), meaning that it's not possible to know and not to know x in the $A = xB$ simultaneously.
- (not $A \sim B$) $\Rightarrow A \,!\!\sim B$, meaning that if knowing x in the $A = xB$ leads to a contradiction, then the discrete logarithm relationship between A and B is unknown.

Using the law and implication, if we can obtain a contradiction by guessing $A \sim B$, then we obtain $A \,!\!\sim B$ and (not $A \sim B$). We can't obtain anything by guessing $A \,!\!\sim B$.

Thus, the denotations $A \sim B$ and $A \,!\!\sim B$ together with the above law and implication for them provide a shorthand for the common way of reasoning about the knowledge of the discrete logarithm relationship. That is, instead of writing, e.g., "*suppose, x in the $A = xB$ is known, then . . . this is a contradiction, hence, solving $A = xB$ is hard*", we write

$$(A \sim B \Rightarrow \ldots \Rightarrow \text{Contradiction}) \Rightarrow A \,!\!\sim B.$$

For any element A and any finite number of elements $B_1, B_2, \ldots, B_n$, let's denote as

$$A = \text{lin}\,(B_1, B_2, \ldots, B_n)$$

the following fact: the scalars $x_1, x_2, \ldots, x_n$ in the equation

$$A = x_1B_1 + x_2B_2 + \ldots + x_nB_n.$$

can be efficiently calculated. Let's call this a known discrete logarithm relationship of A to $B_1, B_2, \ldots, B_n$.

If calculating $x_1, x_2, \ldots, x_n$ in the equation $A = x_1B_1 + x_2B_2 + \ldots + x_nB_n$ is hard, let's call this an unknown discrete logarithm relationship of A to $B_1, B_2, \ldots, B_n$ and designate it as

$$A \,!\!= \text{lin}\,(B_1, B_2, \ldots, B_n)\,.$$

For any elements $A, B_1, B_2, \ldots, B_n$, both $A = \text{lin}\,(B_1, B_2, \ldots, B_n)$ and $A \,!\!= \text{lin}\,(B_1, B_2, \ldots, B_n)$ never hold. The law and implication for these statements are similar to those for $A \sim B$ and $A \,!\!\sim B$:

- (not ($A = \text{lin}\,(B_1, B_2, \ldots, B_n)$ and $A \,!\!= \text{lin}\,(B_1, B_2, \ldots, B_n)$))
- (not $A = \text{lin}\,(B_1, B_2, \ldots, B_n)$) $\Rightarrow A \,!\!= \text{lin}\,(B_1, B_2, \ldots, B_n)$

Also, for any elements A and B:

$$A = \text{lin}\,(B) \quad \text{is equivalent to} \quad A \sim B,$$
$$\text{and} \quad A \,!\!= \text{lin}\,(B) \quad \text{is equivalent to} \quad A \,!\!\sim B.$$

2.2.4 ORTHOGONAL SETS

For any set $S = \{B_1, B_2, \ldots, B_n\}$ of non-zero elements, we denote the following fact as

$$\text{ort}\,(S)$$

and call it an unknown discrete logarithm of each element in the set to the other elements in the set: for each element $B_i \in S$ holds: $B_i \,!\!= \text{lin}\,(S \setminus \{B_i\})$.

For any S, $\text{ort}\,(S)$ means that no element in S can be expressed by means of other elements in S. So, as a shorthand, we call S a set of independent, or orthogonal, elements in this case.

2.2.5 EVIDENCE

Let's call a valid proof of a fact provided by Prover to Verifier as an evidence of the fact. Thus, the game's goal is for Prover to convince Verifier of facts using evidences.

For instance, if x in the relation $A = xB$ is known to Prover, we write this fact as $A \sim B$ for Prover. An evidence of this fact can be simply x that Prover provides to Verifier, so that the later can check that $A = xB$. Also, it can be another acceptable way to convince Verifier of Prover's knowledge of x in the $A \sim B$, e.g., an appropriate sigma-protocol or a Schnorr signature (s, c) where $sB + cA = R$ and c is an output of a pre-agreed ideal hash function on input (B, A, R).

The term 'evidence' is introduced to distinguish between the facts themselves and proofs of facts that Prover provides to Verifier and the latter checks and accepts. That is, for instance, we write

- simply ($A \sim B$ and $C \,!\!\sim D$), when the fact is that x in $A = xB$ is known to both Prover and Verifier and y in $C = yD$ is hard to compute for both of them,
- ($A \sim B$ and $C \,!\!\sim D$) for Prover, when the fact is that x in $A = xB$ is known to Prover and computing y in $C = yD$ is hard for Prover,
- evidence of ($A \sim B$ and $C \,!\!\sim D$), when there is a known to Verifier acceptable proof for the fact that x in $A = xB$ is known to Prover and calculating y in $C = yD$ is hard for Prover.

For all protocols below, if an evidence doesn't pass Verifier's check in a protocol, the protocol is assumed exited by error. For some protocols we define the function Verif instead, that returns 0 or a non-zero value. If 0 is returned, it means that a protocol immediately exits by error. If non-zero is returned, it means the protocol continues.

We call a protocol itself an evidence of a fact under certain conditions, if the protocol successful completion under these conditions implies that Verifier is convinced that the fact holds on the Prover's side.

2.2.6 FIXED ELEMENTS

Let's call an element A fixed if it is not changed during the game. An element A is fixed for a protocol, if it is not changed during its execution.

Prover can convince Verifier that A is fixed in different ways, e.g., by revealing A at the beginning of the protocol or, if $A = xB$, by revealing x and B at the beginning.

2.2.7 RANDOM CHOICE

We use only uniform random choice of scalars over $\mathbb{F}$ everywhere and call it simply 'random choice'.

2.2.8 NEGLIGIBLE PROBABILITY AND CONTRADICTIONS

We assume probability to be negligible if its inverse is exponential in the security parameter $\boldsymbol{\lambda}$. Consequently, if by implications we get a statement that holds with the negligible probability, we assume the statement does not hold.

The same is applied to contradictions: if we have an assumption and its implication such that the implication holds with the negligible probability, we get a contradiction. For example, (assumption holds) $\Rightarrow$ ($c = c'$, where c and c' are chosen uniformly and independently at random) $\Rightarrow$ Contradiction.

2.2.9 DECOY SETS AND THEIR CARDINALITY

We call the anonymity set as a decoy set. One entry of a decoy set belongs to an actual signer. We don't restrict the actual signer to own only one entry in the set, it may own all decoys.

An adversary may own any number of entries in a decoy set, usually except for the one that the actual signer signs with. Also, an adversary may know a relationship between some entries in a decoy set without owning them.

Decoy set cardinalities are assumed polynomial in $\boldsymbol{\lambda}$, that is, cardinality of a decoy set is assumed to be much less than the cardinality of $\mathbb{F}$. An algorithm that goes through all entries of a decoy set is assumed to run in a polynomial time.

We use the terms 'ring' and simply 'set' as the synonyms to 'decoy set', assuming the following semantic difference: 'decoy sets' are usually parts of low-level protocols, 'set' is used when talking about a set membership proof, 'ring'is related to a ring signature.

2.2.10 LINEAR COMBINATIONS

The terms 'linear combination' and 'weighted sum' that we apply to sums of elements multiplied by scalars are interchangeable, they both mean a sum

$$a_1 B_1 + a_2 B_2 + \ldots + a_n B_n.$$

The scalars in the sum are sometimes called ‘weights’, although they don’t carry any additional meaning except for being multipliers for the elements. That is, for instance, the weights aren’t required to be comparable.

2.2.11 UNIQUENESS

We call a vector as unique under certain conditions, when there can be efficiently calculated exactly one vector satisfying the conditions. Calculating a different vector satisfying these conditions is hard. Two vectors are called different if they have at least one position with different items.

For instance, the statement

$$\vec{x} \text{ is unique for the expression } A = \sum_{i=1\ldots n} x_i B_i$$

means that $\vec{x}$ is efficiently computable and it’s hard to calculate a different scalar vector $\vec{y}$ such that the expression holds for it.

3 PRELIMINARY LEMMAS

NotLin lemma:

For any three non-zero A, B, C: if $A \mathrel{!=} \operatorname{lin}(B, C)$, then all three statements hold:

a) For any D and any known e: $D = \operatorname{lin}(B, C) \Rightarrow (A + eD) \mathrel{!=} \operatorname{lin}(B, C)$.

b) For any T: (for some known e: $(A + eT) = \operatorname{lin}(B, C)) \Rightarrow T \mathrel{!=} \operatorname{lin}(B, C)$.

c) Both hold: $A \mathrel{!\sim} B$ and $A \mathrel{!\sim} C$

Proof:

a) Suppose $(A + eD) = \operatorname{lin}(B, C)$, then by definition of lin(), x, y, w, z are provided such that: $(A+eD = xB+yC \Rightarrow A + e(wB + zC) = xB + yC \Rightarrow A = (x - ew)B + (y - ez)C \Rightarrow A = \operatorname{lin}(B, C) \Rightarrow \text{Contradiction}) \Rightarrow (A + eD) \mathrel{!=} \operatorname{lin}(B, C)$

b) Suppose $T = \operatorname{lin}(B, C)$, then by definition of lin(), x, y, w, z are provided such that: $(A + eT = xB + yC \Rightarrow A + e(wB + zC) = xB + yC \Rightarrow A = (x - ew)B + (y - ez)C \Rightarrow A = \operatorname{lin}(B, C) \Rightarrow \text{Contradiction}) \Rightarrow T \mathrel{!=} \operatorname{lin}(B, C)$

c) Suppose $A \sim B$, then by definition of $A \sim B$, x is provided such that $A = xB$. That is, by definition of lin(), $(A = \operatorname{lin}(B, C) \Rightarrow \text{Contradiction}) \Rightarrow A \mathrel{!\sim} B$. Likewise, $A \mathrel{!\sim} C$.

OrtUniqueRepresentation lemma:

For any element A and any vector $\vec{B} = [B_i]_{i=1}^n$ of non-zero elements: if $\operatorname{ort}\left(\vec{B}\right)$ and $A = \operatorname{lin}\left(\vec{B}\right)$, then the vector $\vec{x} = [x_i]_{i=1}^n$ of scalars, such that

$$A = \sum_{i=1\ldots n} x_i B_i,$$

is unique.

Proof: Suppose $\vec{x}$ is not unique, that is, A has one more representation $\vec{y}$, then subtracting both representations we get

$$0 = \sum_{i=1\ldots n} z_i B_i,$$

where $\vec{z} = \vec{x} - \vec{y}$ has at least one non-zero scalar.

Suppose z_j is non-zero, then moving $z_j B_j$ to the left part and dividing by z_j we get

$$B_j = \sum_{i=1\ldots n, i \neq j} (z_i / z_j) B_i.$$

This means that $B_j = \operatorname{lin}\left(\vec{B} \setminus \{B_j\}\right)$, however $B_j \mathrel{!=} \operatorname{lin}\left(\vec{B} \setminus \{B_j\}\right)$ by definition of the $\operatorname{ort}\left(\vec{B}\right) \Rightarrow$ Contradiction.

OrtReduction lemma:

For any set of non-zero elements S, any two elements $B_j, B_k \in S$, any two non-zero scalars a, b:

$$\operatorname{ort}(S) \Rightarrow \operatorname{ort}\left(\left\{\left(aB_j + bB_k\right)\right\} \cup \left(S \setminus \left(\left\{B_j\right\} \cup \{B_k\}\right)\right)\right).$$

Proof: Suppose the opposite, that is, $\left(aB_j + bB_k\right) = \operatorname{lin}\left(S \setminus \left(\left\{B_j\right\} \cup \{B_k\}\right)\right) \Rightarrow$ moving B_k to the right: $aB_j = \operatorname{lin}\left(S \setminus \left\{B_j\right\}\right) \Rightarrow$ dividing by a: $B_j = \operatorname{lin}\left(S \setminus \left\{B_j\right\}\right) \Rightarrow$ Contradiction to the definition of $\operatorname{ort}(S)$.

ZeroRepresentation lemma:
For any $\vec{B} = [B_i]_{i=1}^n$ and any $\vec{x} = [x_i]_{i=1}^n$: if $\mathrm{ort}\left(\vec{B}\right)$ and $0 = \sum_{i=1\ldots n} x_i B_i$, then $\vec{x} = \vec{0}$.

Proof: By the OrtUniqueRepresentation lemma, $\vec{y} = \vec{0}$ is unique for $0 = \sum_{i=1\ldots n} y_i B_i$, hence $\vec{x} = \vec{y} = \vec{0}$.

OrtDisjunction lemma:
For any set of non-zero elements S, any vector of subsets $[S_i | S_i \subset S]_{i=0}^n$ such that for any $j, k \in [0, n]$, $j \neq k$: $S_j \cap S_k = \emptyset$, for any vector of non-zero elements $[Y_i | Y_i = \mathrm{lin}(S_i)]_{i=0}^n$:

$$\mathrm{ort}(S) \Rightarrow \mathrm{ort}\left([Y_i]_{i=0}^n\right).$$

Proof: Suppose the opposite, that is, by definition of lin() there is a vector of known scalars $[x_i]_{i=0}^n$, where at least one x_i is non-zero, such that the weighted sum of $[Y_i]_{i=0}^n$ with weights $[x_i]_{i=0}^n$ is zero:

$$0 = \sum_{i=0\ldots n} x_i Y_i.$$

By definition of lin(), each Y_i is a weighted sum of elements from S, and, as $S_j \cap S_k = \emptyset$, each element from S participates in no more than one of these sums.

Hence, we have a representation of the zero element as a weighted sum of elements from S, where at least one weight is non-zero. This contradicts the ZeroRepresentation lemma. Thus, $\mathrm{ort}\left([Y_i]_{i=0}^n\right)$.

Informally, the OrtDisjunction lemma states that a set of elements built as linear combinations of not-intersecting parts of an orthogonal set is an orthogonal set.

Lin2 lemma:
For any four non-zero fixed elements P, Q, Z, H such that $P \mathrel{!\sim} Q$, the following protocol (Table 1) is an evidence of ($Z = \mathrm{lin}(P, Q)$ and $H = \mathrm{lin}(P, Q)$):

Table 1: Lin2 lemma protocol.

	←	Verifier picks a non-zero random scalar c and sends it to Prover
Prover returns a non-zero scalar r and an evidence of $(P + cQ) \sim (Z + rH)$	→	Verifier checks $(Z + rH) \neq 0$, $r \neq 0$ Verifier checks the evidence $(P + cQ) \sim (Z + rH)$

Proof: Note, the protocol is not claimed to be a sigma-protocol. We have to prove only that (Verifier succeeds in checking $(P + cQ) \sim (Z + rH)$ where $(Z + rH) \neq 0$) $\Rightarrow$ (Prover knows a, b, x, y, such that: $Z = aP + bQ$ and $H = xP + yQ$).

As $(P + cQ) \sim (Z + rH)$ for Prover and $(Z + rH) \neq 0$, Prover knows t such that $P + cQ = tZ + trH$. Suppose $t = 0 \Rightarrow P + cQ = 0 \Rightarrow P \sim Q \Rightarrow$ Contradiction to $P \mathrel{!\sim} Q \Rightarrow t \neq 0$.

Finding Z from the $P + cQ = tZ + trH$:

$$Z = (P + cQ)/t - rH.$$

For another challenge c':

$$Z = (P + c'Q)/t' - r'H,$$

where r' and t' correspond to the $(P + c'Q) \sim (Z + r'H)$.

Eliminating Z: $(P + cQ)/t - rH = (P + c'Q)/t' - r'H \Rightarrow$

$$(1/t - 1/t')P + (c/t - c'/t')Q + (r' - r)H = 0.$$

Suppose $(r' - r) = 0$. We have two possibilities with this assumption: $(1/t - 1/t') = (c/t - c'/t') = 0$ or $(1/t - 1/t')P + (c/t - c'/t')Q = 0$.

$(1/t - 1/t') = (c/t - c'/t') = 0 \Rightarrow (c = c') \Rightarrow$ Contradiction, as c is a random choice.

$(1/t - 1/t')P + (c/t - c'/t')Q = 0 \Rightarrow P \sim Q \Rightarrow$ Contradiction to $P \mathrel{!\sim} Q$, as $P \sim Q$ and $P \mathrel{!\sim} Q$ can't hold together. Hence, $(r' - r) \neq 0$.

Finding H from the equation with the eliminated Z:

$$H = (1/t - 1/t')/(r' - r)P + (c/t - c'/t')/(r' - r)Q.$$

Thus, $H = xP + yQ$, where

$$x = (1/t - 1/t')/(r' - r),$$
$$y = (c/t - c'/t')/(r' - r).$$

Prover is able to efficiently calculate these x and y from the two successful transcripts and, hence, $H = \mathrm{lin}\,(P, Q)$ for Prover.

Finding $Z = aP + bQ$ from $Z = (P + cQ)/t - rH$:

$$Z = (1/t)\,P + (c/t)\,Q - r\,(1/t - 1/t')/(r' - r)\,P - r\,(c/t - c'/t')/(r' - r)\,Q$$

$\Rightarrow$

$$a = 1/t - r(1/t - 1/t')/(r' - r),$$
$$b = c/t - r(c/t - c'/t')/(r' - r).$$

$\Rightarrow Z = \mathrm{lin}\,(P, Q)$ for Prover.

Thus, ($Z = \mathrm{lin}\,(P, Q)$ and $H = \mathrm{lin}\,(P, Q)$) for Prover.

4 LIN2-XOR LEMMA AND ITS COROLLARY

Lin2-Xor lemma:

For any four non-zero fixed elements P_1, $Q_1 P_2$, Q_2, such that $\mathrm{ort}\,(P_1, Q_1 P_2, Q_2)$, and for any two non-zero fixed elements Z, H_1, the following protocol (Table 2) is an evidence of that exactly one of the following a) and b) holds on the Prover's side:

a) $Z = \mathrm{lin}\,(P_1, Q_1)$ and $H_1 = \mathrm{lin}\,(P_1, Q_1)$

b) $Z = \mathrm{lin}\,(P_2, Q_2)$ and $H_1 = \mathrm{lin}\,(P_2, Q_2)$

Table 2: Lin2-Xor lemma protocol.

	$\leftarrow$	Verifier picks two non-zero random scalars c_{11}, c_{13} and sends them to Prover
Prover returns a non-zero scalar r_1 and a non-zero element H_2	$\rightarrow$	Verifier checks $(Z + r_1 H_1) \neq 0$, $r_1 \neq 0$, $H_2 \neq 0$
	$\leftarrow$	Verifier picks a non-zero random scalar c_2 and sends it to Prover
Prover returns a non-zero scalar r_2 and an evidence of $(P_1 + c_{11}Q_1 + c_2 P_2 + c_2 c_{13} Q_2) \sim (Z + r_1 H_1 + r_2 H_2)$	$\rightarrow$	Verifier checks $(Z + r_1 H_1 + r_2 H_2) \neq 0$, $r_2 \neq 0$ Verifier checks the evidence $(P_1 + c_{11}Q_1 + c_2 P_2 + c_2 c_{13} Q_2) \sim (Z + r_1 H_1 + r_2 H_2)$

Proof: Applying the OrtReductionLemma two times, $\mathrm{ort}\,(P_1, Q_1 P_2, Q_2) \Rightarrow \mathrm{ort}\,((P_1 + c_{11}Q_1), (P_2 + c_{13}Q_2)) \Rightarrow$ by definition of ort(), $(P_1 + c_{11}Q_1)\,!\!\sim (P_2 + c_{13}Q_2)$.

Table 3: Lin2-Xor lemma to Lin2 lemma protocol expressions substitution.

Lin2-Xor lemma expressions	**Lin2 lemma expressions**
c_2	c
r_2	r
$(P_1 + c_{11}Q_1)$	P
$(P_2 + c_{13}Q_2)$	Q
$(Z + r_1 H_1)$	Z
H_2	H
$(Z + r_1 H_1) = \mathrm{lin}\,(P_1 + c_{11}Q_1, P_2 + c_{13}Q_2)$	$Z = \mathrm{lin}\,(P, Q)$

Let's move the first two steps of the Lin2-Xor lemma protocol to its premise. After this, we get exactly the premise, protocol and conclusion of the Lin2 lemma with the shown expression substitution (Table 3). Thus, by the conclusion of the Lin2 lemma, Verifier has an evidence of

$$(Z + r_1 H_1) = \mathrm{lin}\left(P_1 + c_{11} Q_1, P_2 + c_{13} Q_2\right)$$

Rewriting this evidence using definition of lin(), we get

$$(Z + r_1 H_1) = a\left(P_1 + c_{11} Q_1\right) + b\left(P_2 + c_{13} Q_2\right), \tag{*}$$

where Verifier is convinced that the scalars a and b are known to Prover. Also, Verifier is convinced that at least one of a and b is non-zero, as $(Z + r_1 H_1) \neq 0$.

For another challenge $\left(c'_{11}, c'_{13}\right)$, reply r'_1, and scalars a', b' known to Prover:

$$\left(Z + r'_1 H_1\right) = a'\left(P_1 + c'_{11} Q_1\right) + b'\left(P_2 + c'_{13} Q_2\right),$$

where at least one of a' and b' is non-zero.

Excluding H_1 from both equations and extracting Z:

$$\left(a\left(P_1 + c_{11} Q_1\right) + b\left(P_2 + c_{13} Q_2\right) - Z\right)/r_1 = \left(a'\left(P_1 + c'_{11} Q_1\right) + b'\left(P_2 + c'_{13} Q_2\right) - Z\right)/r'_1 \Rightarrow$$
$$\left(r_1 - r'_1\right) Z = r_1 a'\left(P_1 + c'_{11} Q_1\right) + r_1 b'\left(P_2 + c'_{13} Q_2\right) - r'_1 a\left(P_1 + c_{11} Q_1\right) - r'_1 b\left(P_2 + c_{13} Q_2\right).$$

We can assume $r_1 \neq r'_1$, as $r_1 = r'_1$ for different random challenges immediately leads to contradiction, so we can divide by $(r_1 - r'_1)$:

$$Z = \left(\left(r_1 a' - r'_1 a\right) P_1 + \left(r_1 a' c'_{11} - r'_1 a c_{11}\right) Q_1 + \left(r_1 b' - r'_1 b\right) P_2 + \left(r_1 b' c'_{13} - r'_1 b c_{13}\right) Q_2\right)/\left(r_1 - r'_1\right)$$

That is, extracting the weights of P_1, Q_1, P_2, Q_2, we have:

$$Z = k_1 P_1 + k_2 Q_1 + k_3 P_2 + k_4 Q_2,$$

where

$$\begin{cases} k_1 = \left(r_1 a' - r'_1 a\right)/\left(r_1 - r'_1\right) \\ k_2 = \left(r_1 a' c'_{11} - r'_1 a c_{11}\right)/\left(r_1 - r'_1\right) \\ k_3 = \left(r_1 b' - r'_1 b\right)/\left(r_1 - r'_1\right) \\ k_4 = \left(r_1 b' c'_{13} - r'_1 b c_{13}\right)/\left(r_1 - r'_1\right) \end{cases}$$

Verifier is convinced that Prover knows the scalars k_1, k_2, k_3, k_4, as it is convinced that all scalars at the right-hand sides of the above equalities are known to Prover. Moreover, as $\mathrm{ort}\left(P_1, Q_1, P_2, Q_2\right)$ and as Z, P_1, Q_1, P_2, Q_2 are fixed by premise, by the OrtUniqueRepresentation lemma Verifier is convinced that the scalars k_1, k_2, k_3, k_4 are constants.

At least one of k_1, k_2, k_3, k_4 is non-zero, as the opposite contradicts to the premise of non-zero Z. Suppose $k_1 \neq 0$. From the first equality:

$$\left(r_1 - r'_1\right) k_1 = \left(r_1 a' - r'_1 a\right) \Rightarrow r_1\left(a' - k_1\right) = r'_1\left(a - k_1\right) \Rightarrow$$
$$\left(a' - k_1\right)/r'_1 = \left(a - k_1\right)/r_1$$

As the right-hand side of this equality depends only of the first transcript, and the left-hand side depends only of the second one, Verifier is convinced that both sides are equal to some constant q known to Prover:

$$\left(a' - k_1\right)/r'_1 = q \text{ and } \left(a - k_1\right)/r_1 = q \Rightarrow$$
$$a' = q * r'_1 + k_1 \text{ and } a = q * r_1 + k_1 \tag{**}$$

Let $t = (k_2/k_1)$. Dividing the equality for k_2 by the equality for k_1:

$$t\left(r_1 a' - r'_1 a\right) = \left(r_1 a' c'_{11} - r'_1 a c_{11}\right) \Rightarrow r'_1 a\left(c_{11} - t\right) = r_1 a'\left(c'_{11} - t\right) \Rightarrow$$
$$a\left(c_{11} - t\right)/r_1 = a'\left(c'_{11} - t\right)/r'_1 \tag{***}$$

As the right-hand side of this equality depends only of the first transcript, and the left-hand side depends only of the second one, Verifier is convinced that both sides are equal to some constant w known to Prover:

$$a\left(c_{11} - t\right)/r_1 = w \text{ and } a'\left(c'_{11} - t\right)/r'_1 = w \Rightarrow$$
$$r_1 = a\left(c_{11} - t\right)/w \text{ and } r'_1 = a'\left(c'_{11} - t\right)/w$$

Using equalities (**) for a and a':

$$\begin{aligned}
wr_1 &= (q * r_1 + k_1)(c_{11} - t) \quad \text{and} \quad wr_1' = \left(qr_1' + k_1\right)\left(c_{11}' - t\right) \Rightarrow \\
r_1 (w - q (c_{11} - t)) &= k_1 (c_{11} - t) \quad \text{and} \quad r_1' \left(w - q \left(c_{11}' - t\right)\right) = k_1 \left(c_{11}' - t\right) \Rightarrow \\
r_1 &= k_1 (c_{11} - t) / (w - q (c_{11} - t)) \quad \text{and} \quad r_1' = k_1 \left(c_{11}' - t\right) / \left(w - q \left(c_{11}' - t\right)\right) \qquad (****)
\end{aligned}$$

Thus, Verifier is convinced that r_1 and r_1' are expressed through the constants known to Prover and through the challenges c_{11} and c_{11}'.

Suppose $k_3 \neq 0$. Likewise Verifier is convinced that

$$r_1 = k_3 (c_{13} - s) / (u - p (c_{13} - s)) \text{ and } r1' = k_3 \left(c_{13}' - s\right) / \left(u - p \left(c_{13}' - s\right)\right) \qquad (*****)$$

for some constants s, u, p known to Prover.

If $k_1 \neq 0$ and $k_3 \neq 0$ is the case, then, according to the (****) and (*****), Verifier gets a contradiction, as r_1 gets completely expressed through each of the two independent randomness c_{11} and c_{13}. Thus, Verifier is convinced that

$$\text{both } k_1 \neq 0 \text{ and } k_3 \neq 0 \text{ never hold together.} \qquad (******)$$

The following implications hold when $k_1 = 0$: (from the (**): $a' = q * r_1'$ and $a = q * r_1$) $\Rightarrow$ ($a'/r_1' = q$ and $a/r_1 = q$) $\Rightarrow$ (from the (***): $q (c_{11} - t) = q \left(c_{11}' - t\right)$) $\Rightarrow q = 0 \Rightarrow$ ($a = 0$ and $a' = 0$) $\Rightarrow k_2 = 0$.

Likewise, when $k_3 = 0$: ($b = 0$ and $b' = 0$) $\Rightarrow k_4 = 0$.

Recalling (******), Verifier is convinced that $Z = k_1 * P_1 + k_2 * Q_1 + k_3 * P_2 + k_4 * Q_2$, where the k_1, k_2, k_3, k_4 are known to Prover and either of ($k_1 = 0$ and $k_2 = 0$) and ($k_3 = 0$ and $k_4 = 0$) holds, never both. That is, by the definitions of evidence and lin(), Verifier has an evidence of

either $Z = \text{lin}(P_1, Q_1)$ or $Z = \text{lin}(P_2, Q_2)$, never both.

Likewise, Verifier has an evidence of

either $H_1 = \text{lin}(P_1, Q_1)$ or $H_1 = \text{lin}(P_2, Q_2)$, never both.

It's not possible that ($Z = \text{lin}(P_1, Q_1)$ and $H_1 = \text{lin}(P_2, Q_2)$), now we prove it. Suppose ($Z = \text{lin}(P_1, Q_1)$ and $H_1 = \text{lin}(P_2, Q_2)$), then, using (*), Prover knows z_1, z_2, h_1, h_2: ($Z = z_1 P_1 + z_2 Q_1$ and $H_1 = h_1 P_2 + h_2 Q_2$) $\Rightarrow$

$$z_1 P_1 + z_2 Q_1 + r_1 (h_1 P_2 + h_2 Q_2) = a (P_1 + c_{11} Q_1) + b (P_2 + c_{13} Q_2).$$

By the OrtUniqueRepresentation lemma: ($z_1 = a$ and $z_2 = a c_{11}$) $\Rightarrow$ ($z_2/z_1 = c_{11}$). However, z_1, z_2 are constants, as the Z, P_1, $Q_1 P_2$, Q_2 are fixed by the premise. Hence, z_2/z_1 can't be equal to the random choice c_{11}, contradiction. Likewise, the case of ($Z = \text{lin}(P_2, Q_2)$ and $H_1 = \text{lin}(P_1, Q_1)$) is not possible.

Hence, either ($Z = \text{lin}(P_1, Q_1)$ and $H_1 = \text{lin}(P_1, Q_1)$) or ($Z = \text{lin}(P_2, Q_2)$ and $H_1 = \text{lin}(P_2, Q_2)$), never both. That is, exactly one of a) and b) holds.

Corollary of Lin2-Xor lemma:
If premise of the Lin2-Xor lemma is met and its protocol is successfully completed, then Verifier is convinced that exactly one of the following a) or b) holds for Prover:

a) $(Z + r_1 H_1) \sim (P_1 + c_{11} Q_1)$ and $(Z + r_1 H_1) \,!\!\sim (P_2 + c_{13} Q_2)$

b) $(Z + r_1 H_1) \sim (P_2 + c_{13} Q_2)$ and $(Z + r_1 H_1) \,!\!\sim (P_1 + c_{11} Q_1)$

Proof: If ($Z = \text{lin}(P_1, Q_1)$ and $H_1 = \text{lin}(P_1, Q_1)$) for Prover, then by definition of lin():

$$(Z + r_1 H_1) = \text{lin}(P_1, Q_1) \text{ for Prover.}$$

At the same time, according to (*), Verifier is convinced that Prover knows a, b in

$$(Z + r_1 H_1) = a (P_1 + c_{11} Q_1) + b (P_2 + c_{13} Q_2).$$

Combining both, by the OrtUniqueRepresentation lemma, definition of lin() and definition of '$\sim$':

$$(Z + r_1 H_1) \sim (P_1 + c_{11} Q_1) \text{ for Prover.}$$

Suppose, $(Z + r_1 H_1) \sim (P_2 + c_{13} Q_2)$ holds for Prover simultaneously with the above. This is a contradiction to the OrtUniqueRepresentation lemma, as the $(Z + r_1 H_1)$ gets two representations: $a (P_1 + c_{11} Q_1)$ and $b (P_2 + c_{13} Q_2)$, where a and b are known to Prover. Hence, $(Z + r_1H_1)\,!\sim (P_2 + c_{13}Q_2)$ for Prover. Thus, we have proven that the case a) of the Lin2-Xor lemma implies the case a) of this corollary:

$$(Z + r_1H_1) \sim (P_1 + c_{11}Q_1) \text{ and } (Z + r_1H_1)\,!\sim (P_2 + c_{13}Q_2)\,.$$

Likewise, the case b) of the Lin2-Xor lemma implies the case b) of this corollary:

$$(Z + r_1H_1) \sim (P_2 + c_{13}Q_2) \text{ and } (Z + r_1H_1)\,!\sim (P_1 + c_{11}Q_1)\,.$$

5 LIN2-SELECTOR LEMMA

5.1 PRELIMINARY DEFINITIONS AND LEMMAS

5.1.1 RSUM

Let's rewrite the $R = P_1 + c_{11}Q_1 + c_2P_2 + c_2c_{13}Q_2$ sum that we considered in the Lin2-Xor lemma as the following tree structure (see Figure 1), renaming P_1, Q_1, P_2, Q_2 as X_0, X_1, X_2, X_3:

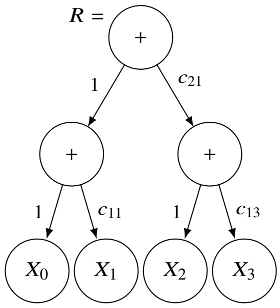

Figure 1: Rsum for four elements.

Informally, this tree structure is evaluated to R recursively, each node performs summation and each arrow performs multiplication by its tag. If all arrow tags are known, then R is easily evaluated as a multi-exponent sum of four summands.

Let's generalize this structure. For instance, for $\left[X_j\right]_{j=0}^{15}$ it will look as in Figure 2:

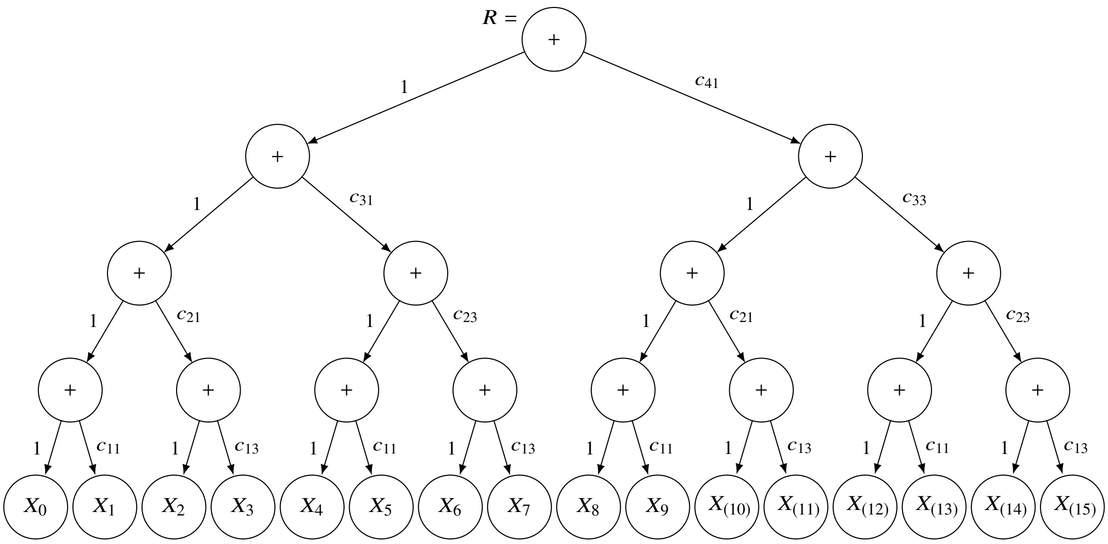

Figure 2: Rsum for sixteen elements.

This is the sum $R = X_0 + c_{11}X_1 + c_{21}X_2 + c_{21}c_{13}X_3 + c_{31}X_4 + c_{31}c_{11}X_5 + c_{31}c_{23}X_6 + c_{31}c_{23}c_{13}X_7 + c_{41}X_8 + c_{41}c_{11}X_9 + c_{41}c_{21}X_{(10)} + c_{41}c_{21}c_{13}X_{(11)} + c_{41}c_{33}X_{(12)} + c_{41}c_{33}c_{11}X_{(13)} + c_{41}c_{33}c_{23}X_{(14)} + c_{41}c_{33}c_{23}c_{13}X_{(15)}$.

Rsum definition:
We call the above tree structure as Rsum and, formally, define it recursively as follows.

For any $n > 0$, for $N = 2^n$, a vector of N elements $\left[X_j\right]_{j=0}^{N-1}$, a vector of 2-tuples of scalars $[(c_{i1}, c_{i3})]_{i=1}^{n-1}$, a scalar c_{n1}, let Rsum $\left(n, N, \left[X_j\right]_{j=0}^{N-1}, [(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_{n1}\right)$ be an element, such that:

$$\left[\begin{array}{l} \text{Rsum}\left(n, N, \left[X_j\right]_{j=0}^{N-1}, [(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_{n1}\right) = \\ \qquad \text{Rsum}\left(n-1, N/2, \left[X_j\right]_{j=0}^{N/2-1}, [(c_{i1}, c_{i3})]_{i=1}^{n-2}, c_{(n-1),1}\right) + \\ \qquad c_{n1}\,\text{Rsum}\left(n-1, N/2, \left[X_j\right]_{j=N/2}^{N-1}, [(c_{i1}, c_{i3})]_{i=1}^{n-2}, c_{(n-1),3}\right) \\ \text{Rsum}\left(1, 2, \left[X_j\right]_{j=2k}^{2k+1}, [], c_{11}\right) = X_{(2k)} + c_{11}X_{(2k+1)}, \text{ where } k \in [0, (N/2)-1]\,. \end{array}\right.$$

Informally, for $n > 1$, Rsum $\left(n, N, \left[X_j\right]_{j=0}^{N-1}, [(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_{n1}\right)$ is a weighted sum of its left and right subtrees with the weights 1 and c_{n1}, respectively. The subtrees are the weighted sums of their left and right subtrees, and so on. For $n = 1$, the Rsum's are leaves and are calculated directly as weighted sums of two elements, with the weights $1, c_{11}$.

Rsum property:
This property follows from the definitions of Rsum and lin():

$$\text{Rsum}\left(n, N, \left[X_j\right]_{j=0}^{N-1}, [(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_{n1}\right) = \text{lin}\left(\left[X_j\right]_{j=0}^{N-1}\right)$$

5.2 LIN2-SELECTOR LEMMA

Lin2-Selector lemma:
For any $n > 1$ and $N = 2^n$, any vector of non-zero fixed elements $\left[X_j\right]_{j=0}^{N-1}$ such that ort $\left(\left[X_j\right]_{j=0}^{N-1}\right)$ holds, for any non-zero fixed element Z, a vector of n non-zero elements $[H_i]_{i=1}^{n}$ where H_1 is fixed, and for a vector of non-zero scalars $[r_i]_{i=1}^{n}$, the following protocol (Table 4) is an evidence of $Z = \text{lin}\left(X_{(2s)}, X_{(2s+1)}\right)$, $s \in [0, N/2-1]$:

Table 4: Lin2-Selector lemma protocol.

Prover	Verifier
Prover and Verifier share a variable i with assigned value $i = 1$	
	Verifier picks two non-zero random scalars c_{i1}, c_{i3} and sends them to Prover
Prover returns a non-zero scalar r_i and a non-zero element H_{i+1}	Verifier checks $(Z + \sum_{j=1\ldots i} r_j H_j) \neq 0, r_i \neq 0, H_{i+1} \neq 0$
	Verifier increments $i = i + 1$ If $(i < n)$, then Verifier goes to the step above: Otherwise, Verifier goes to the step below:
	Verifier picks a non-zero random scalar c_n and sends it to Prover
Prover returns a non-zero scalar r_n and an evidence of	Verifier checks $(Z + \sum_{i=1\ldots n} r_i H_i) \neq 0, r_n \neq 0$ Verifier checks the evidence
Rsum $\left(n, N, \left[X_j\right]_{j=0}^{N-1}, [(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n\right) \sim \left(Z + \sum_{i=1\ldots n} r_i H_i\right)$	Rsum $\left(n, N, \left[X_j\right]_{j=0}^{N-1}, [(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n\right) \sim \left(Z + \sum_{i=1\ldots n} r_i H_i\right)$

Proof: We prove this lemma by induction for every n starting from 2, where n is an integer equal to the logarithm of the $\left[X_j\right]_{j=0}^{N-1}$ vector size.

For the induction base case, $n = 2$, we have exactly the premise of the Lin2-Xor lemma. That is, there are four elements X_0, X_1, X_2, X_3 and also there is one round of the c_{i1}, c_{i3} pair generation, where $i = 1$.

As

$$\mathrm{Rsum}\left(2, 4, \left[X_j\right]_{j=0}^{3}, [(c_{i1}, c_{i3})]_{i=1}^{1}, c_n\right) = X_0 + c_{11}X_1 + c_{21}X_2 + c_{21}c_{13}X_3,$$

Verifier has an evidence of

$$(X_0 + c_{11}X_1 + c_{21}X_2 + c_{21}c_{13}X_3) \sim (Z + r_1H_1 + r_2H_2)$$

in the last step of the protocol.

By the conclusion of the Lin2-Xor lemma, thus, Verifier has an evidence that exactly one of the following holds for Prover

$$Z = \mathrm{lin}\,(X_0, X_1) \text{ and } Z = \mathrm{lin}\,(X_2, X_3),$$

that is, an evidence of $Z = \mathrm{lin}\left(X_{(2s)}, X_{(2s+1)}\right)$, $s \in [0, 1]$. The base case is proven.

The induction hypothesis is that the lemma holds for $n = m > 1$. Let's prove it for $n = (m + 1)$ from the hypothesis. For the sake of this, let's write the lemma premise, protocol and conclusion for $n = (m + 1)$ unwinding the last round of the c_{i1}, c_{i3} challenge pair generation, where $i = m$:

For $n = (m + 1) > 2$ and $N = 2^n = 2(2^m) = 2M$, for any vector of non-zero fixed elements $\left[X_j\right]_{j=0}^{2M-1}$, such that $\mathrm{ort}\left(\left[X_j\right]_{j=0}^{2M-1}\right)$ holds, any non-zero fixed element Z, a vector of $(m + 1)$ non-zero elements $[H_i]_{i=1}^{m+1}$ where H_1 is fixed, and a vector of non-zero scalars $[r_i]_{i=1}^{m+1}$, the following protocol (Table 5) is an evidence of $Z = \mathrm{lin}\left(X_{(2s)}, X_{(2s+1)}\right)$, $s \in [0, M - 1]$:

Table 5: Lin2-Selector lemma protocol for $n = (m + 1)$.

Prover		Verifier
Prover and Verifier share a variable i with assigned value $i = 1$		
	$\leftarrow$	Verifier picks two non-zero random scalars c_{i1}, c_{i3} and sends them to Prover
Prover returns a non-zero scalar r_i and a non-zero element H_{i+1}	$\rightarrow$	Verifier checks $(Z + \sum_{j=1\ldots i} r_jH_j) \neq 0, r_i \neq 0, H_{i+1} \neq 0$
		Verifier increments $i = i + 1$ If $(i < m)$, then Verifier goes to the step above: Otherwise, Verifier goes to the step below:
	$\leftarrow$	Verifier picks two non-zero random scalars c_{m1}, c_{m3} and sends them to Prover
Prover returns a non-zero scalar r_m and a non-zero element H_{m+1}	$\rightarrow$	Verifier checks $(Z + \sum_{j=1\ldots m} r_jH_j) \neq 0, r_m \neq 0, H_{m+1} \neq 0$
	$\leftarrow$	Verifier picks a non-zero random scalar c_{m+1} and sends it to Prover
Prover returns a non-zero scalar r_{m+1} and an evidence of $\mathrm{Rsum}(m + 1, 2M, \left[X_j\right]_{j=0}^{2M-1}, [(c_{i1}, c_{i3})]_{i=1}^{m}, c_{m+1}) \sim \left(Z + \sum_{i=1\ldots(m+1)} r_iH_i\right)$	$\rightarrow$	Verifier checks $(Z + \sum_{i=1\ldots(m+1)} r_iH_i) \neq 0, r_{m+1} \neq 0$ Verifier checks the evidence $\mathrm{Rsum}(m + 1, 2M, \left[X_j\right]_{j=0}^{2M-1}, [(c_{i1}, c_{i3})]_{i=1}^{m}, c_{m+1}) \sim \left(Z + \sum_{i=1\ldots(m+1)} r_iH_i\right)$

Let the Rsum $\left(m + 1, 2M, \left[X_j\right]_{j=0}^{2M-1}, [(c_{i1}, c_{i3})]_{i=1}^{m}, c_{m+1}\right)$ be rewritten by the definition of the Rsum as a sum of four Rsum's Y_0, Y_1, Y_2, Y_3:

$$\begin{aligned}
&\mathrm{Rsum}\left(m+1, 2M, \left[X_j\right]_{j=0}^{2M-1}, [(c_{i1}, c_{i3})]_{i=1}^{m}, c_{m+1}\right) \\
&\quad = \mathrm{Rsum}\left(m, M, \left[X_j\right]_{j=0}^{M-1}, [(c_{i1}, c_{i3})]_{i=1}^{m-1}, c_{m1}\right) \\
&\qquad + c_{m+1}\,\mathrm{Rsum}\left(m, M, \left[X_j\right]_{j=M}^{2M-1}, [(c_{i1}, c_{i3})]_{i=1}^{m-1}, c_{m3}\right) \\
&\quad = \mathrm{Rsum}\left(m-1, M/2, \left[X_j\right]_{j=0}^{M/2-1}, [(c_{i1}, c_{i3})]_{i=1}^{m-2}, c_{(m-1),1}\right) \\
&\qquad + c_{m1}\,\mathrm{Rsum}\left(m-1, M/2, \left[X_j\right]_{j=M/2}^{M-1}, [(c_{i1}, c_{i3})]_{i=1}^{m-2}, c_{(m-1),3}\right) \\
&\qquad + c_{m+1}\,\mathrm{Rsum}\left(m-1, M/2, \left[X_j\right]_{j=M}^{3M/2-1}, [(c_{i1}, c_{i3})]_{i=1}^{m-2}, c_{(m-1),1}\right) \\
&\qquad + c_{m+1}c_{m3}\,\mathrm{Rsum}\left(m-1, M/2, \left[X_j\right]_{j=3M/2}^{2M-1}, [(c_{i1}, c_{i3})]_{i=1}^{m-2}, c_{(m-1),3}\right) \\
&\quad = Y_0 + c_{m1}Y_1 + c_{m+1}Y_2 + c_{m+1}c_{m3}Y_3,
\end{aligned}$$

where:

$$\begin{cases}
Y_0 = \mathrm{Rsum}\left(m-1, M/2, \left[X_j\right]_{j=0}^{M/2-1}, [(c_{i1}, c_{i3})]_{i=1}^{m-2}, c_{(m-1),1}\right) \\
Y_1 = \mathrm{Rsum}\left(m-1, M/2, \left[X_j\right]_{j=M/2}^{M-1}, [(c_{i1}, c_{i3})]_{i=1}^{m-2}, c_{(m-1),3}\right) \\
Y_2 = \mathrm{Rsum}\left(m-1, M/2, \left[X_j\right]_{j=M}^{3M/2-1}, [(c_{i1}, c_{i3})]_{i=1}^{m-2}, c_{(m-1),1}\right) \\
Y_3 = \mathrm{Rsum}\left(m-1, M/2, \left[X_j\right]_{j=3M/2}^{2M-1}, [(c_{i1}, c_{i3})]_{i=1}^{m-2}, c_{(m-1),3}\right)
\end{cases}$$

By the Rsum property,

$$Y_0 = \mathrm{lin}\left(\left[X_j\right]_{j=0}^{M/2-1}\right), \qquad Y_1 = \mathrm{lin}\left(\left[X_j\right]_{j=M/2}^{M-1}\right),$$
$$Y_2 = \mathrm{lin}\left(\left[X_j\right]_{j=M}^{3M/2-1}\right), \qquad Y_3 = \mathrm{lin}\left(\left[X_j\right]_{j=3M/2}^{2M-1}\right).$$

As the subsets $\left[X_j\right]_{j=0}^{M/2-1}$, $\left[X_j\right]_{j=M/2}^{M-1}$, $\left[X_j\right]_{j=M}^{3M/2-1}$, $\left[X_j\right]_{j=3M/2}^{2M-1}$ of the set $\left[X_j\right]_{j=0}^{2M-1}$ don't intersect pairwise, and as $\mathrm{ort}\left(\left[X_j\right]_{j=0}^{2M-1}\right)$ by the premise, we have $\mathrm{ort}\,(Y_0, Y_1, Y_2, Y_3)$ by the OrtDisjunction lemma. Thus, the evidence in the last step of the protocol rewrites as follows:

$$Y_0 + c_{m1}Y_1 + c_{m+1}Y_2 + c_{m+1}c_{m3}Y_3 \sim \left(Z + \sum_{i=1\ldots(m+1)} r_i H_i\right).$$

Defining element F: $F = Z + \sum_{i=1\ldots(m-1)} r_i H_i$, the evidence rewrites

$$Y_0 + c_{m1}Y_1 + c_{m+1}Y_2 + c_{m+1}c_{m3}Y_3 \sim (F + r_m H_m + r_{m+1}H_{m+1})\,.$$

Now, let's look at the step where Verifier picks the challenges c_{m1}, c_{m3}. At that moment, all c_{i1}, c_{i3} and r_i for $i < m$ are already returned by Prover and thus are fixed. Hence, at that moment Y_0, Y_1, Y_2, Y_3 and F are fixed. In addition to this, at that moment H_m is already returned by Prover and thus is fixed.

Hence, having the evidence of $(Y_0 + c_{m1}Y_1 + c_{m+1}Y_2 + c_{m+1}c_{m3}Y_3) \sim (F + r_m H_m + r_{m+1}H_{m+1})$ in the last step, we have the premise and the protocol of the Lin2-Xor lemma here. Namely, we have the fixed $Y_0, Y_1, Y_2, Y_3, F, H_m$ and $\mathrm{ort}\,(Y_0, Y_1, Y_2, Y_3)$. Verifier picks the challenges c_{m1}, c_{m3}, Prover replies with r_m and H_{m+1}, Verifier picks c_{m+1}, Prover replies with r_{m+1} and with the evidence of $(Y_0 + c_{m1}Y_1 + c_{m+1}Y_2 + c_{m+1}c_{m3}Y_3) \sim (F + r_m H_m + r_{m+1}H_{m+1})$.

Hence, if Verifier successfully completes the protocol for $n = (m+1)$, that is, if Verifier accepts that

$$\mathrm{Rsum}\left(m+1, 2M, \left[X_j\right]_{j=0}^{2M-1}, [(c_{i1}, c_{i3})]_{i=1}^{m}, c_{m+1}\right) \sim \left(Z + \sum_{i=1\ldots(m+1)} r_i H_i\right),$$

then it accepts that

$$Y_0 + c_{m1}Y_1 + c_{m+1}Y_2 + c_{m+1}c_{m3}Y_3 \sim (F + r_m H_m + r_{m+1}H_{m+1})\,,$$

and, then, the protocol of the Lin2-Xor lemma is successfully completed, and, by the Corollary of Lin2-Xor lemma, exactly one of the following a) and b) holds for Prover:

a) $(F + r_m H_m) \sim (Y_0 + c_{m1} Y_1)$

b) $(F + r_m H_m) \sim (Y_2 + c_{m3} Y_3)$

Here we can rewrite $Y_0 + c_{m1}Y_1$ and $Y_2 + c_{m3}Y_{12}$ using the definitions of Y_0, Y_1, Y_2, Y_3 and the definition of Rsum as

$$\begin{aligned}
Y_0 + c_{m1}Y_1 &= \text{Rsum}\left(m-1, M/2, \left[X_j\right]_{j=0}^{M/2-1}, [(c_{i1}, c_{i3})]_{i=1}^{m-2}, c_{(m-1),1}\right) \\
&\quad + c_{m1}\,\text{Rsum}\left(m-1, M/2, \left[X_j\right]_{j=M/2}^{M-1}, [(c_{i1}, c_{i3})]_{i=1}^{m-2}, c_{(m-1),3}\right) \\
&= \text{Rsum}\left(m, M, \left[X_j\right]_{j=0}^{M-1}, [(c_{i1}, c_{i3})]_{i=1}^{m-1}, c_{m1}\right) \\
Y_2 + c_{m3}Y_3 &= \text{Rsum}\left(m-1, M/2, \left[X_j\right]_{j=M}^{3M/2-1}, [(c_{i1}, c_{i3})]_{i=1}^{m-2}, c_{(m-1),1}\right) \\
&\quad + c_{m3}\,\text{Rsum}\left(m-1, M/2, \left[X_j\right]_{j=3M/2}^{2M-1}, [(c_{i1}, c_{i3})]_{i=1}^{m-2}, c_{(m-1),3}\right) \\
&= \text{Rsum}\left(m, M, \left[X_j\right]_{j=M}^{2M-1}, [(c_{i1}, c_{i3})]_{i=1}^{m-1}, c_{m3}\right)
\end{aligned}$$

Thus, using the definition of F and the two above equalities, inserting $r_m H_m$ into the sum, we obtain that exactly one of the following a) or b) holds for Prover:

a) $\left(Z + \sum_{i=1\ldots m} r_i H_i\right) \sim \text{Rsum}\left(m, M, \left[X_j\right]_{j=0}^{M-1}, [(c_{i1}, c_{i3})]_{i=1}^{m-1}, c_{m1}\right)$

b) $\left(Z + \sum_{i=1\ldots m} r_i H_i\right) \sim \text{Rsum}\left(m, M, \left[X_j\right]_{j=M}^{2M-1}, [(c_{i1}, c_{i3})]_{i=1}^{m-1}, c_{m3}\right)$

If a) holds, then, renaming c_{m1} to be c_m, the premise and protocol of this lemma for the case $n = m$ are met, and, by the induction hypothesis, Verifier has an evidence of

$$Z = \text{lin}\left(X_{(2s)}, X_{(2s+1)}\right), \ s \in [0, M/2-1].$$

If b) holds, then, renaming c_{m3} to be c_m, the premise and protocol of this lemma for the case $n = m$ are met, and, by the induction hypothesis, Verifier has an evidence of

$$Z = \text{lin}\left(X_{(2s)}, X_{(2s+1)}\right), \ s \in [M/2, M-1].$$

Putting it all together, from the induction hypothesis for $n = m$, we have obtained, for $n = (m+1)$, that if the premise and protocol of this lemma are met, then Verifier has exactly one of the two evidences:

$$\begin{gathered}
\left(Z = \text{lin}\left(X_{(2s)}, X_{(2s+1)}\right), \ s \in [0, M/2-1]\right) \\
or \left(Z = \text{lin}\left(X_{(2s)}, X_{(2s+1)}\right), \ s \in [M/2, M-1]\right).
\end{gathered}$$

Unifying the intervals for s, we obtain, that Verifier has an evidence of

$$Z = \text{lin}\left(X_{(2s)}, X_{(2s+1)}\right), \ s \in [0, M-1].$$

That is, recalling $M = 2^m = 2^{m+1}/2$, we have obtained the conclusion of this lemma for $n = (m+1)$.

Thus, the lemma is proven for all $n > 1$.

6 L2S MEMBERSHIP PROOF

We construct a proof of membership (PoM) protocol called **L2S**. Verifier is provided with an element Z, and, upon successful completion of all steps of the protocol, Verifier is convinced that Z is a commitment to a pair of elements from a publicly known set of element pairs, such that Prover knows an opening for Z.

We prove that the **L2S** protocol is complete, sound, special honest verifier zero-knowledge, and no possibility exists for identifying a pair in the set that the element Z corresponds to.

6.1 COM2 COMMITMENT

Com2 definition:
Given a vector $\vec{X} = \left[X_j\right]_{j=0}^{N-1}$ of $N = 2^n$ elements such that $\text{ort}\left(\vec{X}\right)$ holds, two scalars k_0, k_1, and an integer index $s \in [0, N/2-1]$, let's define $\text{Com2}\left(k_0, k_1, s, \vec{X}\right)$ as an element $(k_0 X_{2s} + k_1 X_{2s+1})$. That is,

$$\text{Com2}\left(k_0, k_1, s, \vec{X}\right) = k_0 X_{2s} + k_1 X_{2s+1}$$

A 3-tuple (k_0, k_1, s) is an opening to the Com2 $\left(k_0, k_1, s, \vec{X}\right)$.

Knowing $\vec{X}$, a Com2 commitment Z over $\vec{X}$, and the scalars k_0, k_1 of its opening, it's possible to efficiently calculate the index s by iterating through $\vec{X}$ and checking if $Z = k_0 X_{2s} + k_1 X_{2s+1}$.

By the OrtUniqueRepresentation lemma, if Z has a (k_0, k_1, s) opening over $\vec{X}$, then the opening (k_0, k_1, s) is unique.

We call a Com2 commitment as a commitment to a member-pair. Set of member-pairs $\left[X_j\right]_{j=0}^{N-1}$ is called a decoy set.

6.2 L2S MEMBERSHIP PROOF PROTOCOL

We define **L2S** PoM protocol as four procedures

$$\mathbf{L2S} = \{\mathbf{DecoySetGen}, \mathbf{ComGen}, \mathbf{InteractionProcedure}, \mathbf{Verif}\},$$

where:

- **DecoySetGen** (n) is an arbitrary function that returns an element vector $\vec{X} = \left[X_j\right]_{j=0}^{N-1}$ of $N = 2^n$ elements, such that ort $\left(\vec{X}\right)$ holds. Each element in the generated $\vec{X}$ has a distribution that is independent of the distributions of other elements in the same $\vec{X}$ and is indistinguishable from the uniform randomness. Two vectors generated by the **DecoySetGen** may have non-empty intersection.

  For any **DecoySetGen** implementation choice, the returned vector $\vec{X}$ orthogonality, independence of the element distributions of each other within the vector and their uniform randomness are to be guaranteed.

- **ComGen**$\left(\vec{X}\right)$ is an arbitrary function that returns a pair $((k_0, k_1, s), Z)$, where k_0 is non-zero and chosen uniformly at random, k_1 is arbitrary, $s \in [0, N/2 - 1]$, and $Z = \text{Com2}\left(k_0, k_1, s, \vec{X}\right)$.

  For any **ComGen** implementation choice, the independence and random uniformity of k_0 distribution together with $Z = \text{Com2}\left(k_0, k_1, s, \vec{X}\right)$ and $k_0 \neq 0$ are to be guaranteed.

- **InteractionProcedure** is depicted in Table 6. It starts with Prover having an opening (k_0, k_1, s) and Verifier having a commitment Z.

  On completion of the **InteractionProcedure**, Verifier has a tuple $\left([(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n, Z, [(r_i, H_i)]_{i=1}^{n}, c, T, t\right)$ that contains Z together with all the challenges and replies occurred during the Prover and Verifier interaction.

- **Verif** function is shown in Table 7. It takes the tuple that Verifier has upon completion of the **Interaction-Procedure** together with the decoy set from the **DecoySetGen**. It returns 1 or 0, meaning the verification is completed successfully or failed.

Overall, the **L2S** protocol steps are the following:

- A decoy set $\vec{X}$ is generated using same implementation of the **L2S.DecoySetGen** at both Prover's and Verifier's sides.
- Prover gets an opening (k_0, k_1, s) from the **L2S.ComGen**. At the same time, Verifier gets some element Z.
- All steps of the **L2S.InteractionProcedure** are performed between the Prover and Verifier. On completion of the **L2S.InteractionProcedure** Verifier has a tuple $\left([(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n, Z, [(r_i, H_i)]_{i=1}^{n}, c, T, t\right)$.
- Verifier calls the **L2S.Verif** for the decoy set and tuple obtained above. Iff the **L2S.Verif** returns 1, then the **L2S** protocol is completed successfully. As we prove below, the successful completion means $Z = \text{Com2}\left(k_0, k_1, s, \vec{X}\right)$.

Note, the *InvertLastBit* function used in the **L2S.InteractionProcedure** takes an unsigned integer and returns this integer with inverted least significant bit in its binary representation. That is, *InvertLastBit* $(i) = (2\,(i//2) + (i+1)\,\%2)$, where the $//$ and $\%$ are the quotient and remainder operators. We use the *InvertLastBit* for indexes to switch between the left and right subtrees of a binary tree node.

6.2.1 PROOF OF THE RELATION BETWEEN R AND W

Now, we show that

$$\text{Rsum}\left(n, N, \left[X_j\right]_{j=0}^{N-1}, [(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n\right) = xW,$$

where $x = a/w$ is calculated on the Prover's side.

Table 6: **L2S.InteractionProcedure**.

Prover and Verifier common parameters:

- $n, N = 2^n$,
- a set of elements $\left[X_j\right]_{j=0}^{N-1}$ = **DecoySetGen**(n)
- a shared variable i with assigned value $i = 1$

Prover:		Verifier:
(k_0, k_1, s), where: $k_0 \neq 0$, $s \in [0, N/2 - 1]$		Z
$w = k_0$ $k = k_1/w$ $M = N$ $\left[Y_j\right]_{j=0}^{M-1} = \left[X_j\right]_{j=0}^{N-1}$ $(z, h) = (2s, 2s+1)$ $a = 1$ $q \leftarrow$ random, non-zero		
$H_1 = wY_h/q$	$\xrightarrow{H_1}$	
	$\longleftarrow$	$(c_{i1}, c_{i3}) \leftarrow$ random, non-zero
$(c_{i0}, c_{i2}) = (1, 1)$ $(e, g) = (c_{i,(z\%4)}, c_{i,(h\%4)})$		
$r_i = q((g/e) - k)$	$\xrightarrow{r_i}$	
$k = 0$ $a = ea$		
$M = M/2$ $\left[Y_j\right]_{j=0}^{M-1} = \left[\left(Y_{(2j)} + c_{i,((2j+1)\%4)} Y_{(2j+1)}\right)/e\right]_{j=0}^{M-1}$		
$z = (z//2)$ $h = InvertLastBit(z)$ $q \leftarrow$ random, non-zero		
$H_{i+1} = wY_h/q$	$\xrightarrow{H_{i+1}}$	
		$i = i + 1$ If $(i < n)$, then: Otherwise:
	$\longleftarrow$	$c_n \leftarrow$ random, non-zero
$(c_{n0}, c_{n1}) = (1, c_n)$ $(e, g) = (c_{n,z}, c_{n,h})$		
$r_n = q\,(g/e)$	$\xrightarrow{r_n}$	
$a = ea$ $x = a/w$ $q \leftarrow$ random, non-zero $W = (k_0 X_{2s} + k_1 X_{2s+1}) + \sum_{i=1 \ldots n} r_i H_i$		
$T = qW$	$\xrightarrow{T}$	
	$\longleftarrow$	$c \leftarrow$ random, non-zero
$t = q - xc$	$\xrightarrow{t}$	
		Verifier has a tuple: $\left([(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n, Z, [(r_i, H_i)]_{i=1}^{n}, c, T, t\right)$

Table 7: **L2S.Verif** function.

Input: n, $\left[X_j\right]_{j=0}^{N-1}$, $\left([(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n, Z, [(r_i, H_i)]_{i=1}^{n}, c, T, t\right)$, where $N = 2^n$

$S = Z$
For $i = 1 \dots n$:
    If ($r_i == 0$ or $H_i == 0$) then return 0
    $S = S + r_i H_i$
    If $S == 0$ then return 0
$W = S$
$R = \text{Rsum}\left(n, N, \left[X_j\right]_{j=0}^{N-1}, [(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n\right)$
If $(tW + cR) == T$ then return 1
Else return 0.

The expression

$$\left[Y_j\right]_{j=0}^{M-1} = \left[X_j\right]_{j=0}^{N-1}, \text{ where } M = N,$$

at the beginning of the Prover's part of the **L2S.InteractionProcedure** lets all Y_j's be X_j's.
Next, down the protocol execution flow, when $i = 1$, the expression

$$\left[Y_j\right]_{j=0}^{M-1} = \left[\left(Y_{(2j)} + c_{i,((2j+1)\%4)} Y_{(2j+1)}\right)/e\right]_{j=0}^{M-1}, \text{ where } M = N/2,$$

lets the Y_j's vector contain $N/2$ Rsum's

$$\text{Rsum}\left(1, 2, [X_t]_{t=2j}^{2j+1}, [], c_{1,((2j+1)\%4)}\right),$$

each divided by the common factor e, which is equal to 1 for $i = 1$. The variable a accumulates the common factor, that is, remains to be 1.

When $i = 2$, the expression

$$\left[Y_j\right]_{j=0}^{M-1} = \left[\left(Y_{(2j)} + c_{i,((2j+1)\%4)} Y_{(2j+1)}\right)/e\right]_{j=0}^{M-1}, \text{ where } M = N/4,$$

lets the Y_j's vector contain $N/4$ Rsum's:

$$\text{Rsum}\left(2, 4, [X_t]_{t=4j}^{4(j+1)-1}, \left[\left(c_{d,1}, c_{d,3}\right)\right]_{d=1}^{1}, c_{2,((2j+1)\%4)}\right)$$

divided by the common factor $c_{2,(s\%4)}$ simultaneously accumulated in a. Note, for all s': $c_{s',0} = c_{s',2} = 1$.

When $i = 3$, the expression

$$\left[Y_j\right]_{j=0}^{M-1} = \left[\left(Y_{(2j)} + c_{i,((2j+1)\%4)} Y_{(2j+1)}\right)/e\right]_{j=0}^{M-1}, \text{ where } M = N/8,$$

lets the Y_j's vector contain $N/8$ Rsum's:

$$\text{Rsum}\left(3, 8, [X_t]_{t=8j}^{8(j+1)-1}, \left[\left(c_{d,1}, c_{d,3}\right)\right]_{d=1}^{2}, c_{3,((2j+1)\%4)}\right)$$

divided by the common factor $c_{2,(s\%4)} c_{3,((s//2)\%4)}$. The variable a contains the common factor $c_{2,(s\%4)} c_{3,((s//2)\%4)}$.

And so on, until $i = n$. At that moment Y_j's vector contains 2 Rsum's representing the left and right subtrees of the root, both divided by a, where a is the product of all challenges on the path from the pair with index s to the root.

At the same time, from the beginning, Prover composes H_i's and r_i's using the Y_j's.

When $i = 1$, Prover sends to Verifier:

$$H_1 = w X_{(2s+1)}/q, \qquad \text{where } q \text{ is random,}$$
$$r_1 = q\left(c_{1,((2s+1)\%4)} - k\right), \qquad \text{where } q \text{ is the same and } k = k_1/w,$$

so that $(Z + r_1 H_1) = w\,\text{Rsum}\left(1, 2, [X_t]_{t=2s}^{2s+1}, [], c_{1,((2s+1)\%4)}\right)$.
Next, Prover reshuffles q, sets $h = \mathit{InvertLastBit}(s)$ and sends:

$$H_2 = w\,\text{Rsum}\left(1, 2, [X_t]_{t=2h}^{2h+1}, [], c_{1,((2h+1)\%4)}\right)/q$$

When $i = 2$, Prover sets $k = 0$ and sends:

$$r_2 = q\left(c_{2,(h\%4)}/c_{2,(s\%4)}\right),$$

so that

$$\begin{aligned}(Z + r_1H_1 + r_2H_2) = w\,\text{Rsum}\left(1, 2, [X_t]_{t=2s}^{2s+1}, [], c_{1,((2s+1)\%4)}\right) + \\ w(c_{2,(h\%4)}/c_{2,(s\%4)})\,\text{Rsum}\left(1, 2, [X_t]_{t=2h}^{2h+1}, [], c_{1,((2h+1)\%4)}\right) = \\ w\,\text{Rsum}\left(2, 4, [X_t]_{t=4(s//2)}^{4((s//2)+1)-1}, \left[\left(c_{d,1}, c_{d,3}\right)\right]_{d=1}^{1}, c_{2,((2(s//2)+1)\%4)}\right)/c_{2,(s\%4)}\end{aligned}$$

Next, Prover reshuffles q, sets $h = \mathit{InvertLastBit}(s//2)$ and sends:

$$H_3 = w\,\text{Rsum}\left(2, 4, [X_t]_{t=4h}^{4(h+1)-1}, \left[\left(c_{d,1}, c_{d,3}\right)\right]_{d=1}^{1}, c_{2,((2h+1)\%4)}\right)/(c_{2,(s\%4)}q)$$

When $i = 3$, Prover sends:

$$r_3 = q\left(c_{3,(h\%4)}/c_{3,((s//2)\%4)}\right),$$

so that

$$\begin{aligned}(Z + r_1H_1 + r_2H_2 + r_3H_3) = w\,\text{Rsum}\left(2, 4, [X_t]_{t=4(s//2)}^{4((s//2)+1)-1}, \left[\left(c_{d,1}, c_{d,3}\right)\right]_{d=1}^{1}, c_{2,((2(s//2)+1)\%4)}\right)/c_{2,(s\%4)} + \\ w(c_{3,(h\%4)}/c_{3,((s//2)\%4)})\,\text{Rsum}\left(2, 4, [X_t]_{t=4h}^{4(h+1)-1}, \left[\left(c_{d,1}, c_{d,3}\right)\right]_{d=1}^{1}, c_{2,((2h+1)\%4)}\right)/c_{2,(s\%4)} = \\ w\,\text{Rsum}\left(2, 4, [X_t]_{t=8(s//4)}^{8((s//4)+1)-1}, \left[\left(c_{d,1}, c_{d,3}\right)\right]_{d=1}^{2}, c_{3,((2(s//4)+1)\%4)}\right)/(c_{2,(s\%4)}c_{3,((s//2)\%4)})\end{aligned}$$

And so on, until $i = n$ and

$$W = (Z + r_1H_1 + r_2H_2 + \ldots + r_nH_n) = w\,\text{Rsum}\left(n, N, \left[X_j\right]_{j=0}^{N-1}, [(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n\right)/a$$

Thus, $\text{Rsum}\left(n, N, \left[X_j\right]_{j=0}^{N-1}, [(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n\right) = xW$.

6.2.2 PROOF THAT CORRECT OPENING LETS L2S.VERIF RETURN 1

The (T, c, t) part of the **L2S.Verif** input is the Schnorr identification scheme [16] initial message, challenge and reply for the relation $R = xW$.

If $Z = \text{Com2}\left(k_0, k_1, s, \left[X_j\right]_{j=0}^{N-1}\right)$, then the values of W calculated on the Prover's side and in the **L2S.Verif** are identical, as in both places W is calculated by the same formula with the same $[(r_i, H_i)]_{i=1}^{n}$ and Z.

As proven in 6.2.1, $\text{Rsum}\left(n, N, \left[X_j\right]_{j=0}^{N-1}, [(1, c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n\right) = xW$. Thus, on the Prover's side xW is equal to R used in the **L2S.Verif**. As the Schnorr identification scheme [16] is complete, this implies $(tW + cR) == T$.

Hence, $Z = \text{Com2}\left(k_0, k_1, s, \left[X_j\right]_{j=0}^{N-1}\right)$ implies **L2S.Verif** returns 1.

6.3 LS2 PROTOCOL PROPERTIES

6.3.1 COMPLETENESS

As proven in 6.2.2, if Z on Verifier's input is equal to the commitment $\text{Com2}\left(k_0, k_1, s, \left[X_j\right]_{j=0}^{N-1}\right)$, where the opening (k_0, k_1, s) is the Prover's input, then the **L2S.Verif** returns 1. This means that the **LS2** protocol is complete.

6.3.2 SOUNDNESS

The **L2S.InteractionProcedure** with the subsequent call to the **L2S.Verif** meets the Lin2-Selector lemma protocol.

If the **L2S.Verif** returns 1, then $(tW + cR) == T$, and, as the Schnorr identification scheme is sound, Verifier has an evidence of $W \sim R$, that is, an evidence of

$$\text{Rsum}\left(n, N, \left[X_j\right]_{j=0}^{N-1}, [(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n\right) \sim \left(Z + \sum_{i=1\ldots n} r_iH_i\right).$$

Thus, by the Lin2-Selector lemma, if the **L2S.Verif** returns 1, then Verifier is convinced that $Z = \text{lin}\left(X_{(2s)}, X_{(2s+1)}\right)$ for some member-pair $\left(X_{(2s)}, X_{(2s+1)}\right)$, where $s \in [0, N/2 - 1]$.

That is, using the definitions of lin() and Com2, if the **L2S.Verif** returns 1, then Verifier is convinced that Prover knows an opening (k_0, k_1, s) of the commitment Z such that $Z = \text{Com2}\left(k_0, k_1, s, \left[X_j\right]_{j=0}^{N-1}\right)$, where s corresponds to a member-pair in the decoy set. Thus, the **LS2** protocol is sound.

6.3.3 STRUCTURE AND VIEW OF THE L2S PROVER-VERIFIER PUBLIC TRANSCRIPT

The **LS2** protocol Prover-Verifier public transcript is the following tuple

$$\left([(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n, Z, [(r_i, H_i)]_{i=1}^{n}, c, T, t\right).$$

The items T and t in the transcript are related to the Schnorr id scheme, they are distributed uniformly at random. However, they are not independent.

Here we are interested only in the transcripts that Verifier accepts, that is, in those for which $(tW + cR) == T$. The W and R are calculated from the publicly visible elements and scalars

$$\left(Z, [(r_i, H_i)]_{i=1}^{n}\right) \text{ and } \left(\left[X_j\right]_{j=0}^{N-1}, [(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n\right),$$

respectively. Thus, the element T is a linear combination of the variables seen for anyone. Hence, we exclude T from our consideration: for any transcript accepted by Verifier the item T carries no information and can be restored from the other items of the transcript and elements of the decoy set.

All the challenges are independent and uniformly random. All r_i's are independent and uniformly random, too, as each r_i is obfuscated by the private multiplier q, which is reshuffled for each r_i.

The random multiplier q is reduced in the products $r_i H_i$. These products represent Rsum's, i.e., the subtree sums at heights i. That is, for each height i, the element $(Z + r_1 H_1 + \ldots + r_{i-1} H_{i-1})$ corresponds to a subtree that the index s belongs to. At the same time, the element $r_i H_i$ corresponds to a complimentary subtree that the index s doesn't belong to. The height $i = 1$ is the only exclusion from this, as Z has a fraction k_1/k_0 of its complimentary subtree, nevertheless, this difference has no effect on the transcript item independencies and uniformities.

All the elements $Z, r_1 H_1, \ldots, r_i H_i$ are obfuscated by the multiplier w. The multiplier w is private and uniformly random, as $w = k_0$, where k_0 is uniformly random by the definition of **L2S.ComGen**. By the definition of Rsum, each $r_i H_i$ is a linear combination of the elements from the $\left[X_j\right]_{j=0}^{N-1}$ with efficiently computable scalar coefficients. Moreover, all $r_i H_i$'s depend of the different non-intersecting subsets of the $\left[X_j\right]_{j=0}^{N-1}$.

Using the terms introduced in [4], the $r_i H_i$'s and Z are linearly independent degree 2 polynomials of a private set of the independent and random uniform scalars

$$\left\{\{w\} \cup \left\{\text{discrete logarithms of } \left[X_j\right]_{j=0}^{N-1}\right\}\right\}.$$

The coefficients of these polynomials are efficiently computable from the $[(c_{i1}, c_{i3})]_{i=1}^{n-1}$, c_n, and k_1. Thus, reducing the question of the $r_i H_i$'s distributions to the (P, Q)-DDH problem [4], we have

$$P = \left\{\left[X_j\right]_{j=0}^{N-1}\right\} \text{ and } Q = \left\{\{Z\} \cup \{r_i H_i\}_{i=1}^{n}\right\},$$
$$\text{Span}(P) \cap \text{Span}(Q) = \emptyset.$$

By the (P, Q)-DDH assumption, the distributions of all the $r_i H_i$'s and Z are indistinguishable from $\{e_i G\}_{i=1}^{n+1}$, where all the e_i's are independent and uniformly random.

As the DDH assumption implies the (P, Q)-DDH [4] for our polynomials in the above sets P and Q, we have all the $r_i H_i$'s and Z distributed independently and uniformly at random under the DDH. We have proven this for any conversation transcript between honest Prover and Verifier over any fixed decoy set $\left[X_j\right]_{j=0}^{N-1}$ generated by the **L2S.DecoySetGen**. For readability, we omit the word 'indistinguishable', reserving it for the distributions.

For all honest conversation transcripts over all really used and possibly intersecting decoy sets, we reduce the question to the same (P, Q)-DDH problem with

$$P = \emptyset \text{ and } Q = \cup_{\text{all transcripts TR with their decoy sets}} \left\{\{Z\} \cup \{r_i H_i\}_{i=1}^{n} \cup \left[X_j\right]_{j=0}^{N-1}\right\}_{\text{TR}},$$
$$\text{Span}(P) \cap \text{Span}(Q) = \emptyset,$$

where the private set of the independent and random uniform scalars is

$$\cup_{\text{all transcripts TR with their decoy sets}} \Big\{\{w\} \cup \Big\{\text{discrete logarithms of } \left[X_j\right]_{j=0}^{N-1}\Big\}\Big\}_{\text{TR}}.$$

By requiring w to be chosen independently and uniformly at random for each transcript, meaning same Z is never used in any two different conversations, we obtain that all the $r_i H_i$'s and Z's publicly seen across all the accepted transcripts are distributed independently and uniformly at random under the DDH. Their distributions are independent of each other and of the distributions of the elements X_j's of decoy sets.

Thus, we conclude, that all items, except for the items T, of all honest **L2S** conversation transcripts have independent and random uniform distributions under the DDH, provided that the input commitments Z are never reused. That is, the input commitments are to be generated anew with the **L2S.ComGen** for each conversation.

As for the transcript items T, each honest transcript item T is efficiently computable from the other items of the transcript. Overall, the items T carry no information in honest transcripts, they serve only to distinguish honest transcripts, i.e. the proofs that Verifier accepts, from the transcripts where Prover tries to dishonestly prove knowledge of opening, that Verifier rejects.

6.3.4 SPECIAL HONEST VERIFIER ZERO-KNOWLEDGE

We show the L2S protocol is sHVZK following the definition by Ronald Cramer, Ivan Damgård, and Berry Schoenmakers [6]. We use a natural extrapolation of the sHVZK definition to the n-round public-coin protocols: we require a simulated transcript to be indistinguishable from the space of honest conversation transcripts with the same challenges.

Having the random independence property proven for the transcript items in 6.3.3, it's easy to build a simulator, that for any given challenges and for any given input Z generates a simulated transcript that Verifier accepts, and no PPT algorithm is able to distinguish it from the space of honest transcripts with the same challenges.

The simulator acts as follows:

- It takes an empty L2S transcript placeholder and puts the given input Z and challenges $[(c_{i1}, c_{i3})]_{i=1}^{n-1}$, c_n in their places.
- It independently generates random uniform scalars and puts them in the places of scalars in the placeholder.
- It independently generates random uniform scalars and puts their exponents in the places of elements in the placeholder, except for the place of element T.
- It takes the values $[(c_{i1}, c_{i3})]_{i=1}^{n-1}$, c_n, Z, $[(r_i, H_i)]_{i=1}^{n}$, c, t from the already filled in places of the placeholder, obtains $\left[X_j\right]_{j=0}^{N-1}$ by calling **L2S.DecoySetGen**, calculates

$$R = \text{Rsum}\left(n, N, \left[X_j\right]_{j=0}^{N-1}, [(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n\right),$$
$$W = Z + \sum_{i=1\ldots n} r_i H_i,$$

  and puts value $(tW + cR)$ in the place of T.

Thus, the simulated transcript is ready. Verifier accepts it, as it passes the $(tW + cR) == T$ check in the **L2S.Verif**. Not mentioning the other checks in the **L2S.Verif** that are also passed with overwhelming probability as the checked values are uniformly random.

Suppose, there exists a PPT algorithm that distinguishes with non-negligible probability the simulated transcript from the space of honest transcripts with the same challenges. As proven in 6.3.3, the space contains the transcripts with all items having distributions indistinguishable from the distributions of the items of the simulated transcript, except for the item T. However, T is calculated the same way from the same sources for honest and for simulated transcripts, hence the algorithm is not able to distinguish the transcripts by T's. Hence, we have that the PPT algorithm is able to distinguish indistinguishable distributions, contradiction.

We have proven the **L2S** protocol is sHVZK under the DDH, provided that the input commitments Z are generated anew with the **L2S.ComGen** for each Prover-Verifier conversation.

6.3.5 INDISTINGUISHABILITY OF THE MEMBER-PAIR INDEX

Here we prove, that the member-pair index s in the opening (k_0, k_1, s) of the input commitment Z can not be distinguished from a honest conversation transcript.

Suppose, there exists a PPT algorithm that distinguishes s with non-negligible probability from a honest Prover-Verifier conversation transcript. Applying the algorithm to all transcripts in the honest transcript space, we obtain a partitioning of the space where each partition with non-negligible probability distinguishes some information about the actual values of s in it. However, according to 6.3.3 the space entries contain only the items indistinguishable from the independent and uniform randomness, with the exclusion of the items T that carry no information. Thus, we have the algorithm that distinguishes with non-negligible probability some information about the actual values of s from the independent and uniform randomness, that is a contradiction.

We have proven the member-pair index s in the **L2S** proof of membership protocol is indistinguishable under the DDH, as long as the input commitments Z are generated anew with the **L2S.ComGen** for each Prover-Verifier conversation.

6.3.6 NOTE ABOUT SPECIAL SOUNDNESS

We have already proven the properties that we need for further consideration in this paper. Anyway, interestingly, there exists a possibility to prove special soundness for the **L2S** protocol for the case if $k_1 = 0$, extrapolating the definition of special soundness by R.Cramer et al. [6] to n-round protocols.

The extrapolation is that in addition to the 3-round protocol special soundness definition in [6] we require the first message of an n-round protocol to contain a commitment of the Prover's random tape state. Thus, two honest Prover-Verifier conversation transcripts produced with different challenges and with the same first Prover's message represent two conversations where the Prover's random tape is fixed and the Verifier's random tape is reshuffled. For the 3-round protocols the extrapolated definition reduces exactly to the special soundness definition in [6].

To comply with this definition, the L2S protocol is extended with the first message containing Prover's random tape commitment, that Prover sends to Verifier at the beginning of the conversation. The random tape commitment serves only to ensure equality of the random values used by Prover internally in two conversations, it can be, for instance, a hash of the Prover's random tape. Thus, the first message carries no information about the (k_0, k_1, s), where the k_0 is the witness w, the k_1 is guaranteed to be always zero by some other means, and the s is an auxiliary scalar that can be obtained in polynomial time from w.

With this extension, let's build a PPT witness extractor for the L2S protocol. The extractor acts as following (sketch):

- It runs $N/2$ parallel guesses about s. For each guess:
  - It extracts x for the relation $R = xW$ using the Schnorr id witness extractor.
  - From x it finds w as $w = (a/x)$, where a is known for a guess.
- One of the parallel guesses ends up with w successfully found. Thus, the witness is extracted.

7 L2S PROTOCOL EXTENSIONS

7.1 IL2S PROTOCOL, SHVZK FOR NON-RANDOM INPUT

As shown in 6.3.4, the **L2S** is sHVZK under the DDH, provided that the scalar k_0 in the Prover's input (k_0, k_1, s) has independent and randomly uniform distribution. To remove this restriction and to allow the protocol to keep the sHVZK property for any input commitment distribution, including the cases when a linear relationship between different input commitments is known to an adversary, we extend the **L2S** protocol with an input randomization. Of course, as the input commitments are publicly seen in the transcripts, the adversary is still able to track the known relationships between them, however, with the sHVZK the adversary is not able to obtain any information beyond that from the transcripts.

The idea of the input randomization is that right at the beginning of the **L2S.InteractionProcedure** Prover multiplies the opening-commitment pair $((k_0, k_1, s), Z)$ by a private random uniform scalar f and supports Verifier with an evidence of $(Z \sim fZ)$ in the form of Schnorr id tuple. Next, the **L2S.InteractionProcedure** is run for the multiplied by f opening and commitment:

$$((k_0, k_1, s), Z) \leftarrow ((fk_0, fk_1, s), fZ).$$

We define **iL2S** protocol as four procedures

$$\textbf{iL2S} = \{\textbf{DecoySetGen=L2S.DecoySetGen}, \textbf{ComGen}, \textbf{InteractionProcedure}, \textbf{Verif}\},$$

where

- **ComGen** $\left(\vec{X}\right)$ is an arbitrary function that returns a pair $((k_0, k_1, s), Z_0)$, where k_0 is arbitrary non-zero, k_1 is arbitrary, $s \in [0, N/2 - 1]$, and $Z_0 = \text{Com2}\left(k_0, k_1, s, \vec{X}\right)$.

  For any **ComGen** implementation choice, the $k_0 \neq 0$ and $Z_0 = \text{Com2}\left(k_0, k_1, s, \vec{X}\right)$ are to be guaranteed.

- **iL2S.InteractionProcedure** is depicted in Table 8. It starts with Prover having (k_0, k_1, s), $k_0 \neq 0$, and Verifier having Z_0.

  On completion of the **iL2S.InteractionProcedure**, Verifier has two tuples: (Z_0, c_0, T_0, t_0) and $\left([(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n, Z, [(r_i, H_i)]_{i=1}^{n}, c, T, t\right)$, that contain the initial input as Z_0 and the randomized input as Z together with all the challenges and replies occurred during the Prover and Verifier interaction.

Table 8: **iL2S.InteractionProcedure**.

Prover and Verifier common parameters:		
– $n, N = 2^n$, – a set of elements $\left[X_j\right]_{j=0}^{N-1}$ = **DecoySetGen**(n)		
Prover:		Verifier:
(k_0, k_1, s), where:$k_0 \neq 0$		Z_0
s – secret index, $s \in [0, N/2 - 1]$		
$Z_0 = \text{Com2}\left(k_0, k_1, s, \left[X_j\right]_{j=0}^{N-1}\right)$		
$f \leftarrow$ random, non-zero		
$Z = fZ_0$	$Z \rightarrow$	
$(k_0, k_1, s) = (fk_0, fk_1, s)$		
$q \leftarrow$ random, non-zero		
$T_0 = qZ_0$	$T_0 \rightarrow$	
	$\leftarrow c_0$	$c_0 \leftarrow$ random, non-zero
$t_0 = q - fc_0$	$t_0 \rightarrow$	
Run **L2S.InteractionProcedure** for the new (k_0, k_1, s) and Z		
		Verifier has two tuples: (Z_0, c_0, T_0, t_0), $\left([(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n, Z, [(r_i, H_i)]_{i=1}^{n}, c, T, t\right)$

- **iL2S.Verif** function is shown in Table 9. It takes the two tuples from the **iL2S.InteractionProcedure** together with the decoy set from the **DecoySetGen** and returns 1 or 0.

Table 9: **iL2S.Verif** function.

Input: n, $\left[X_j\right]_{j=0}^{N-1}$, where $N = 2^n$, (Z_0, c_0, T_0, t_0), $\left([(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n, Z, [(r_i, H_i)]_{i=1}^{n}, c, T, t\right)$
If $(t_0 Z_0 + c_0 Z) == T_0$ then continue Else return 0 Run **L2S.Verif** for the n, $\left[X_j\right]_{j=0}^{N-1}$, $\left([(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n, Z, [(r_i, H_i)]_{i=1}^{n}, c, T, t\right)$

The steps for the **iL2S** protocol are the same as for the **L2S** protocol.

7.1.1 IL2S PROTOCOL COMPLETENESS, SOUNDNESS AND SHVZK

As the Schnorr id and the **L2S** protocols are complete and sound, the **iL2S** protocol is complete and sound.

The **iL2S** protocol is sHVZK. To prove this, we repeat the same steps as those for the **L2S** sHVZK proof in 6.3.4 with the only two additions:

- As the (Z_0, c_0, T_0, t_0) tuple is put at the beginning of the public Prover-Verifier transcript and as Z in the transcript becomes $Z = fZ_0$, it's necessary to determine the distributions of them:
  - c_0 is an independent and randomly uniform honest Verifier's challenge.
  - Z has independent and random uniform distribution, as f in the equation $Z = fZ_0$ is private, independent, and uniformly random.
  - t_0 is independent and uniformly random, as it is obfuscated by the private independent and randomly uniform scalar q in $t_0 = q - fc_0$.
  - Z_0 is independent of the other items in the transcript, however, it is not uniformly random.
  - T_0 is not independent, it is evaluated as $T_0 = (t_0 Z_0 + c_0 Z)$ from the items (Z_0, Z, c_0, t_0).

  Thus, all T_0's can be excluded from consideration, as they carry no information. We get to conclusion, that the **iL2S** transcript contains two dependent items: T_0 and T, that are evaluated from the other items. It contains the input commitments as Z_0, and there is no item, except for T_0, distinguishably dependent of Z_0 in the transcript. All the other items are independent and uniformly random.
- **iL2S** simulator puts the input commitment in the place of Z_0 and fills in all the other places, except for the ones of T_0 and T, with the independent and uniformly random values. It puts the evaluated values $(t_0 Z_0 + c_0 Z)$ and $(tW + cR)$ in the places of T_0 and T, respectively.

7.2 ML2S PROTOCOL

A parallel version of the **iL2S** protocol is a protocol that runs multiple instances of the **iL2S.InteractionProcedure** in parallel and thus proves membership for multiple commitments at once. We call it **mL2S** protocol.

The **mL2S** protocol is four procedures

$$\textbf{mL2S} = \{\textbf{DecoySetGen=L2S.DecoySetGen}, \textbf{ComGen=iL2S.ComGen}, \textbf{MapInteractionProcedure}, \textbf{JoinVerif}\},$$

where:

- **mL2S.MapInteractionProcedure** is depicted in Table 10. It starts with Prover having L openings $\left[\left(k_0^p, k_1^p, s^p\right) | k_0^p \neq 0\right]_{p=1}^{L}$ and Verifier having L commitments $\left[Z_0^p\right]_{p=1}^{L}$.

  On completion of the **mL2S.InteractionProcedure**, Verifier has L tuples:

  $$\left(\left(Z_0^p, c_0, T_0^p, t_0^p\right), \left([(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n, Z^p, \left[\left(r_i^p, H_i^p\right)\right]_{i=1}^{n}, c, T^p, t^p\right)\right)_{p=1}^{L},$$

  that contain the outputs of L **iL2S.InteractionProcedure** parallel runs with the common decoy set and challenges.

Table 10: **mL2S.MapInteractionProcedure**.

<table>
<tr><td colspan="2">Prover and Verifier common parameters:
• L
• $n, N = 2^n$</td></tr>
<tr><td>Prover:
$\left[\left(k_0^p, k_1^p, s^p\right) | k_0^p \neq 0\right]_{p=1}^{L}$</td><td>Verifier:
$\left[Z_0^p\right]_{p=1}^{L}$</td></tr>
<tr><td colspan="2">For each $p \in [1, L]$: run iL2S.InteractionProcedure using n, $\left(k_0^p, k_1^p, s^p\right)$ as arguments for Prover, and n, Z_0^p as arguments for Verifier.
All the parallel iL2S.InteractionProcedure instances share the same decoy set $\left[X_j\right]_{j=0}^{N-1} = \textbf{DecoySetGen}(n)$ and same Verifier's challenges $c_0, [(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n, c$</td></tr>
<tr><td></td><td>Verifier has L tuples:
$\left[\left(\left(Z_0^p, c_0, T_0^p, t_0^p\right), \left([(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n, Z^p,\right.\right.\right.$
$\left.\left.\left.\left[\left(r_i^p, H_i^p\right)\right]_{i=1}^{n}, c, T^p, t^p\right)\right)\right]_{p=1}^{L}$</td></tr>
</table>

Table 11: **mL2S.JoinVerif** function.

Input: $L, n, \left[X_j\right]_{j=0}^{N-1}$, where $N = 2^n$, $\left(\left(Z_0^p, c_0, T_0^p, t_0^p\right), \left([(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n, Z^p, \left[\left(r_i^p, H_i^p\right)\right]_{i=1}^{n}, c, T^p, t^p\right)\right)_{p=1}^{L}$
$R = \text{Rsum}\left(n, N, \left[X_j\right]_{j=0}^{N-1}, [(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n\right)$ For each $p \in [1, L]$: run **iL2S.Verif** using n, $\left[X_j\right]_{j=0}^{N-1}$ and $\left(Z_0^p, c_0, T_0^p, t_0^p\right), \left([(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n, Z^p, \left[\left(r_i^p, H_i^p\right)\right]_{i=1}^{n}, c, T^p, t^p\right)$ as arguments. Inside each **iL2S.Verif** call, within nested **L2S.Verif** call, use the calculated above R for the **iL2S.Verif.L2S.Verif**.R Return 0 if one of the **iL2S.Verif** calls returns 0. Otherwise, return 1.

- **mL2S.JoinVerif** function is shown in Table 11. It takes the L tuples from the **mL2S.MapInteractionProcedure** together with the decoy set from the **DecoySetGen** and returns 1 or 0.

  The **mL2S.JoinVerif** performs L verifications in parallel. As all the Rsum's R inside the nested **iL2S.Verif.L2S.Verif** calls are the same, the **mL2S.JoinVerif** performs their calculation only once, at the beginning, and uses the calculated value
  $$R = \text{Rsum}\left(n, N, \left[X_j\right]_{j=0}^{N-1}, [(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n\right) \text{ for them.}$$

The steps for the **mL2S** protocol are identical to the steps of the **iL2S** protocol, with the only difference in that the parallel procedure versions are used instead of the sequential ones:

$$\textbf{MapInteractionProcedure} \rightarrow \textbf{InteractionProcedure},$$
$$\textbf{JoinVerif} \rightarrow \textbf{Verif}$$

7.2.1 ML2S PROTOCOL COMPLETENESS, SOUNDNESS AND SHVZK

The **mL2S** protocol completeness and soundness immediately follow from the completeness and soundness of the **iL2S** protocol.

The **mL2S** protocol is sHVZK. To prove this, we repeat the same steps as for the **iL2S** sHVZK proof in 7.1.1 and, consequently, as for the **L2S** sHVZK proof in 6.3.4 with the only addition below.

The space of honest **mL2S** transcripts is the space of honest **iL2S** transcripts with the only difference in that it is partitioned by the **mL2S** proof. Each partition contains **iL2S** transcripts with the same challenges. Nevertheless, all their items, except for those challenges and Z_0, T_0, T discussed above, are distributed independently and uniformly at random. Hence, the honest **mL2S** transcript space reveals no information beyond the information accessible from the input commitments and partitioning per se.

A simulator for the **mL2S** protocol runs L **iL2S** protocol simulators in parallel, and, after completion, the simulated transcript contains L indistinguishable from the honest **iL2S** simulated transcripts. Thus, an **mL2S** simulated transcript is indistinguishable from an honest **mL2S** transcript.

7.2.2 ML2S PROTOCOL COMPLEXITIES

Recalling the **mL2S** transcript

$$\left(\left(Z_0^p, c_0, T_0^p, t_0^p\right), \left([(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n, Z^p, \left[\left(r_i^p, H_i^p\right)\right]_{i=1}^{n}, c, T^p, t^p\right)\right)_{p=1}^{L},$$

where all data except for the initial elements Z_0^p's and challenges are transmitted, the amount of data transmitted from Prover to Verifier is shown in Table 12.

Table 12: **mL2S** transmitted data amount.

	$\mathbb{G}$	$\mathbb{F}$
mL2S	$L\,(n+3)$	$L\,(n+2)$

The $R = \text{Rsum}\left(n, N, \left[X_j\right]_{j=0}^{N-1}, [(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n\right)$ calculation, performed only once for all L verifications, requires only one multi-exponentiation for n summands. This is seen from the Rsum recursive definition in 5.1.1

that can be unwound, so that all the scalar coefficients for the elements from the $\left[X_j\right]_{j=0}^{N-1}$ are calculated as the scalar-scalar multiplications and, after that, a single multi-exponentiation of the elements from the $\left[X_j\right]_{j=0}^{N-1}$ to their respective coefficients is performed.

The **mL2S** verification complexity is shown in Table 13, where $N = 2^n$:

Table 13: **mL2S** verification complexity.

	multi-exp(N)	single-exp
mL2S	1	$nL + 3L + 1$

8 ML2S-BASED NON-INTERACTIVE POM AND SIGNATURE

Having an interactive honest verifier zero-knowledge interactive PoM protocol, it's possible to turn it to a non-interactive zero-knowledge PoM scheme using the Fiat-Shamir heuristic in the ROM [8].

We create a non-interactive zero-knowledge PoM scheme on the base of the **mL2S**. After that, we construct a signer-ambiguous linkable ring signature scheme on the base of the created PoM scheme.

As the **mL2S** requires an orthogonal decoy set with element distributions indistinguishable from independent uniform randomness, we employ a 'point-to-point' hash function $\mathbf{H_{point}}(\dots)$ defined below.

8.1 PRELIMINARIES

Elliptic curve points and elements, point definition:
We assume the prime-order group $\mathbb{G}$ is instantiated with an elliptic curve point group of the same order, so that the curve points represent the elements of $\mathbb{G}$ hereinafter. Thus, we use the term 'points' instead of 'elements', they become equivalent below.

Any to scalar hash function $H_{\mathbf{scalar}}(\dots)$ definition:
We call $\mathbf{H_{scalar}}(\dots)$ an ideal hash function that accepts any number of arguments of any type, i.e., the arguments are scalars in $\mathbb{F}$ and points in $\mathbb{G}$. It returns a scalar from $\mathbb{F}$. The function is sensitive to its arguments order.

Point to point hash function $H_{\mathbf{point}}(\dots)$ definition:
We call $\mathbf{H_{point}}()$ an ideal hash function that accepts a points in $\mathbb{G}$ and returns a point in $\mathbb{G}$.

Ideal hash functions and random oracles:

We use the term 'ideal hash function' as a shorthand for the term 'cryptographic hash function that is indifferentiable from a random oracle'. For the $\mathbf{H_{scalar}}$ it can be, for instance, SHA-3. For the $\mathbf{H_{point}}$ it can be, for instance, function described in [7].

Integers n, N, L:

We assume the integers n, N, L have the following meaning hereinafter:

- $N > 1$ is a number of decoys, N is a power of 2 each time, $N/2$ is the number of decoy pairs
- $n = \log_2(N)$
- L is a threshold for signature: $0 < L < (N/2 + 1)$. For membership proof, L is any number: $0 < L$

Decoy vector as a vector of pairs:

The procedure **mL2S.DecoySetGen** in 7.2 returns a decoy vector $\left[X_j\right]_{j=0}^{N-1}$. We reshape this vector to be a vector of pairs $\left[(P_j, Q_j)\right]_{j=0}^{N/2-1}$ below.

Thus, the vector $\left[X_j\right]_{j=0}^{N-1}$ becomes a flattened view of the $\left[(P_j, Q_j)\right]_{j=0}^{N/2-1}$, where for any $s \in [0, N/2-1]$: $P_s = X_{2s}, Q_s = X_{2s+1}$. We write $\left[X_j\right]_{j=0}^{N-1} = Flatten\left(\left[(P_j, Q_j)\right]_{j=0}^{N/2-1}\right)$ for this.

Procedure substitution and lambda function:

To denote procedure substitution, we use the notion of lambda functions. For instance, if we have a **Sheme** = $\{\mathbf{\ldots, ProcedureB}\}$, where the **ProcedureB** is defined as taking X and returning $\mathbf{H_{point}}(X)$, then, if we use the **Scheme** within another scheme and want the **ProcedureB** to return $\left(X + \mathbf{H_{point}}(X)\right)$, we write: **Sheme.ProcedureB** = $\lambda(X) \,.\, \left(X + \mathbf{H_{point}}(X)\right)$.

8.2 NIZK PROOF OF MEMBERSHIP BASED ON THE ML2S

We construct a non-interactive zero-knowledge proof for the following statement: given two vectors of points $\left[B_j\right]_{j=0}^{N/2-1}$ and $[A^p]_{p=1}^{L}$, Prover knows a vector of scalar-integer pairs

$$\left[(v^p, s^p) \,|\, A^p = v^p \mathbf{H_{point}}(B_{s^p}), s^p \in [0, N/2-1]\right]_{p=1}^{L}.$$

That is, for each point A^p from the $[A^p]_{p=1}^{L}$ Prover knows a scalar v^p, such that (A^p/v^p) is a member of $\left[\mathbf{H_{point}}(B_j)\right]_{j=0}^{N/2-1}$.

Note, the s^p's are not required to be different, that is, only membership is going to be proved.

8.2.1 PROOF DATA STRUCTURE

For $L = 1$ the proof data structure transmitted from Prover to Verifier is

$$\sigma = \left(Z_0, T_0, Z, t_0, [(r_i, H_i)]_{i=1}^{n}, T, t\right)$$

Essentially, this data structure is a part of the **mL2S** transcript that is interactively transmitted from Prover to Verifier for each of L parallel membership proofs. The only exclusion is Z_0, which the **mL2S** Verifier knows beforehand.

For any L, the proof data transmitted from Prover to Verifier is L instances of σ, that is, $[\sigma^p]_{p=1}^{L}$.

8.2.2 ML2SHPOM NON-INTERACTIVE SCHEME

The abbreviation **mL2SHPoM** stands for the **mL2S**-based hashed proof of membership scheme, i.e., the aforementioned non-interactive proof, that we create. The **mL2SHPoM** is seven procedures:

mL2SHPoM = {**PreimageSetGen**, **HashPoint**, **GetImageSet**, **MemberSetGen**, **GetDecoySet**, **GetProof**, **Verif**},

where:

- **mL2SHPoM.PreimageSetGen** returns a vector $\left[B_j\right]_{j=0}^{N/2-1}$ of arbitrary points, the points in the returned vector are only required to be unequal to each other.
- **mL2SHPoM.HashPoint** takes a point B and returns a point-hash of B. An implementation is shown in Listing 1, although this implementation can be changed.

  The only requirement for the **HashPoint** is that any its implementation be an ideal point-to-point hash function.

  Listing 1: **mL2SHPoM.HashPoint** initial implementation.

  ```
  Input: B
  Output: A point-hash of B
  Procedure:
      Return H_point(B)
  ```

- **mL2SHPoM.GetImageSet** maps the **HashPoint** to the pre-image set and returns a set of images. Implementation is in Listing 2.

  Listing 2: **mL2SHPoM.GetImageSet** implementation.

  ```
  Input: none
  Output: image set [P_j]_{j=0}^{N/2-1}, HashPoint mapped to the pre-images
  Procedure:
      [B_j]_{j=0}^{N/2-1} = PreimageSetGen()
      [P_j]_{j=0}^{N/2-1} = [HashPoint(B_j)]_{j=0}^{N/2-1}
  ```

```
    Return [P_j]_{j=0}^{N/2-1}
```

- **mL2SHPoM.MemberSetGen** returns a vector $[A^p]_{p=1}^{L}$ of points that are going to be proven members of the image set returned by the **GetImageSet** multiplied by some scalar coefficients known to Prover.
- **mL2SHPoM.GetDecoySet** returns a decoy set $\left[X_j\right]_{j=0}^{N-1}$ for use in the proof. Even elements of the $\left[X_j\right]_{j=0}^{N-1}$ are elements of the image set, while odd elements are composed in such a way, so the possibility of knowledge of linear relationship between them and the elements of the member set together with the elements of the image set is excluded. Implementation is in Listing 3.

Listing 3: **mL2SHPoM.GetDecoySet** implementation.

```
Input: none
Output: decoy set [X_j]_{j=0}^{N-1}
Procedure:
    [P_j]_{j=0}^{N/2-1} = GetImageSet()
    [A^p]_{p=1}^{L} = MemberSetGen()
    [B_j]_{j=0}^{N/2-1} = PreimageSetGen()
    Qshift = H_scalar([A^p]_{p=1}^{L}, [P_j]_{j=0}^{N/2-1}) G
    [Q_j]_{j=0}^{N/2-1} = [H_point(Qshift + B_j)]_{j=0}^{N/2-1}
    [X_j]_{j=0}^{N-1} = Flatten([(P_j, Q_j)]_{j=0}^{N/2-1})
    Return [X_j]_{j=0}^{N-1}
```

- **mL2SHPoM.GetProof** takes a vector of private pairs $[(v^p, s^p)]_{p=1}^{L}$ together with a public scalar seed e and returns a vector $[\sigma^p]_{p=1}^{L}$, that is, returns a non-interactive proof, or 0 on error. The **GetProof** is the **mL2S.MapInteractionProcedure** translated to non-interactive setting. Specification is in Listing 4.

Listing 4: **mL2SHPoM.GetProof** specification.

```
Input:  [(v^p, s^p)]_{p=1}^{L}      --private keys
        e                          --scalar seed
Output: [σ^p]_{p=1}^{L} or 0       --proof, vector of σ's on success,
                                   --0 on failure
Procedure:
• Let [X_j]_{j=0}^{N-1} = GetDecoySet()
• Let [A^p]_{p=1}^{L} = MemberSetGen()
• Ensure the private keys correspond to the member set elements:
    For p = 1...L:
        If A^p ≠ v^p X_{2s^p} then Return 0
• Let [(k_0^p, k_1^p, s^p)]_{p=1}^{L} = [(v^p, 0, s^p)]_{p=1}^{L}
• [Z_0^p]_{p=1}^{L} = [A^p]_{p=1}^{L}
• Run all L iL2S.InteractionProcedure's in parallel with the
   [(k_0^p, k_1^p, s^p)]_{p=1}^{L} and [Z_0^p]_{p=1}^{L} as arguments. Stop all them at the point,
   where the first challenge c_0 is to be obtained. At that moment
   the values [(Z_0^p, T_0^p, Z^p)]_{p=1}^{L} are already calculated.
• Calculate e = H_scalar(e, [X_j]_{j=0}^{N-1}, [(Z_0^p, T_0^p, Z^p)]_{p=1}^{L})
• Let c_0 = e
• Continue all the L parallel procedures to the point, where the
   challenge tuple (c_11, c_13) is to be obtained. At that moment the
   [t_0^p]_{p=1}^{L} and [H_1^p]_{p=1}^{L} are already calculated.
```

- Calculate $e = \mathbf{H_{scalar}}\left(e, \left[t_0^p\right]_{p=1}^L, \left[H_1^p\right]_{p=1}^L\right)$
- Let $(c_{11}, c_{13}) = (e, \mathbf{H_{scalar}}(e))$
- Continue all the L parallel procedures to the point, where the challenge tuple (c_{21}, c_{23}) is to be obtained. At that moment the $\left[r_1^p\right]_{p=1}^L$ and $\left[H_2^p\right]_{p=1}^L$ are already calculated.
- Calculate $e = \mathbf{H_{scalar}}\left(e, \left[r_1^p\right]_{p=1}^L, \left[H_2^p\right]_{p=1}^L\right)$
- Let $(c_{21}, c_{23}) = (e, \mathbf{H_{scalar}}(e))$
- And so on..., until all values $\left[\left(Z_0^p, T_0^p, Z^p, t_0^p, \left[\left(r_i^p, H_i^p\right)\right]_{i=1}^n, T^p, t^p\right)\right]_{p=1}^L$ and $\left(c_0, [(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n, c\right)$ are calculated.
- Let $[\sigma^p]_{p=1}^L = \left[\left(Z_0^p, T_0^p, Z^p, t_0^p, \left[\left(r_i^p, H_i^p\right)\right]_{i=1}^n, T^p, t^p\right)\right]_{p=1}^L$
- Return $[\sigma^p]_{p=1}^L$

- **mL2SHPoM.Verif** takes a proof generated by the **GetProof** and returns 0 or 1. It is the **mL2S.JoinVerif** translated to non-interactive setting. Specification is in Listing 5.

Listing 5: **mL2SHPoM.Verif** specification.

Input: $[\sigma^p]_{p=1}^L$ -- proof, a vector of σ's
e -- scalar seed, same as used for **GetProof** call
Output: 0 or 1 -- verification is failed or completed ok
Procedure:
- Let $\left[X_j\right]_{j=0}^{N-1} = \mathbf{GetDecoySet}()$
- Extract the values of $\left[\left(Z_0^p, T_0^p, Z^p\right)\right]_{p=1}^L$ from the $[\sigma^p]_{p=1}^L$
- Calculate $e = \mathbf{H_{scalar}}\left(e, \left[X_j\right]_{j=0}^{N-1}, \left[\left(Z_0^p, T_0^p, Z^p\right)\right]_{p=1}^L\right)$
- Let $c_0 = e$
- Extract the values of $\left[t_0^p\right]_{p=1}^L$ and $\left[H_1^p\right]_{p=1}^L$ from the $[\sigma^p]_{p=1}^L$
- Calculate $e = \mathbf{H_{scalar}}\left(e, \left[t_0^p\right]_{p=1}^L, \left[H_1^p\right]_{p=1}^L\right)$
- Let $(c_{11}, c_{13}) = (e, \mathbf{H_{scalar}}(e))$
- Extract the values of $\left[r_1^p\right]_{p=1}^L$ and $\left[H_2^p\right]_{p=1}^L$ from the $[\sigma^p]_{p=1}^L$
- Calculate $e = \mathbf{H_{scalar}}\left(e, \left[r_1^p\right]_{p=1}^L, \left[H_2^p\right]_{p=1}^L\right)$
- Let $(c_{21}, c_{23}) = (e, \mathbf{H_{scalar}}(e))$
- And so on..., until all values $\left(c_0, [(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n, c\right)$ are restored. At this moment all values of $\left[\left(Z_0^p, T_0^p, Z^p, t_0^p, \left[\left(r_i^p, H_i^p\right)\right]_{i=1}^n, T^p, t^p\right)\right]_{p=1}^L$ are extracted from the $[\sigma^p]_{p=1}^L$.
- For $p = 1 \ldots L$:
  - If $\left(t_0^p Z_0^p + c_0 Z^p\right) \neq T_0^p$ then Return 0
- Calculate $R = \mathrm{Rsum}\left(n, N, \left[X_j\right]_{j=0}^{N-1}, [(1, c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n\right)$
- For $p = 1 \ldots L$:
  - Let $S = Z^p$
  - For $i = 1 \ldots n$:
    - $S = S + H_i^p$
    - If $(S == 0)$ or $(r_i^p == 0)$ then Return 0
  - $W = S$
  - If $(t^p W + cR) \neq T^p$ then Return 0
- Return 1

Overall, the **mL2SHPoM** non-interactive proof scheme works in the following scenario:

- Prover and Verifier agree on the scheme implementation, particularly, on the **PreimageSetGen** and **HashPoint** functions.
- Knowing a set of private keys $[(v^p, s^p)]_{p=1}^{L}$, that connect the elements of the member set $[A^p]_{p=1}^{L}$ returned by the **MemberSetGen** to the elements of the image set $\left[P_j\right]_{j=0}^{N/2-1}$ returned by the **GetImageSet**, Prover calls the **GetProof** using a seed e and obtains a proof $[\sigma^p]_{p=1}^{L}$.
- Prover sends the proof $[\sigma^p]_{p=1}^{L}$ and the seed e to Verifier.
- Verifier extracts $\left[Z_0^p\right]_{p=1}^{L}$ from the $[\sigma^p]_{p=1}^{L}$. The set $\left[Z_0^p\right]_{p=1}^{L}$ is exactly the set $[A^p]_{p=1}^{L}$ returned by the **MemberSetGen** on Prover's side.
- Verifier calls **Verif** for the $[\sigma^p]_{p=1}^{L}$ and e. If 1 is returned, then Verifier is convinced that Prover knows the private keys, which connect each element of the set $\left[Z_0^p\right]_{p=1}^{L}$ to an element of the $\left[P_j\right]_{j=0}^{N/2-1}$.

8.2.3 ML2SHPOM COMPLETENESS, SOUNDNESS AND ZERO-KNOWLEDGE

The procedures of the **mL2SHPoM** scheme meet the **mL2S** procedures translated to non-interactive setting with the Fiat-Shamir heuristic. The **mL2SHPoM** scheme inherits the completeness and soundness from the **mL2S**.

As the **mL2S** is honest verifier zero-knowledge, the **mL2SHPoM** scheme, where Verifier restores the random challenges from the transcript and, thus, is not able to cheat, is zero-knowledge.

8.2.4 ML2SHPOM COMPLEXITIES

The **mL2SHPoM** proof size, recalling the proof is $[\sigma^p]_{p=1}^{L}$, is shown in Table 14. The scalar seed is not accounted, as it can have any value agreed between Prover and Verifier, e.g., be fixed as $e = 0$.

Table 14: **mL2SHPoM** proof size.

	$\mathbb{G}$	$\mathbb{F}$
mL2SHPoM	$L(n+4)$	$L(n+2)$

The **mL2SHPoM** verification complexity is shown in Table 15, where $N = 2^n$. We use the same optimization for the Rsum calculation, as in the **mL2S**. The scalar-scalar multiplications and $\mathbf{H_{scalar}}$ calls are assumed taking a negligible amount of the computational time.

Table 15: **mL2SHPoM** verification complexity.

	multi-exp(N)	single-exp	$\mathbf{H_{point}}$
mL2SHPoM	1	$nL + 3L + 1$	$N + 1$

8.3 LINKABLE RING SIGNATURE BASED ON THE ML2SHPOM

We construct **mL2SLnkSig** linkable ring signature scheme on the base of the **mL2SHPoM** scheme.

8.3.1 REALIZATION IDEA

The idea is the following: suppose, we have a ring of public keys $\left[B_j\right]_{j=0}^{N/2-1}$ and want to prove knowledge of L private keys

$$\left[(b^p, s^p)\,|\,b^p G = B_{s^p}, s^p \in [0, N/2-1], \forall i,j : s^i \neq s^j\right]_{p=1}^{L}.$$

Also, we want to detect the cases when a private key $(b, \_)$ participates in different proofs. Defining I as $\mathbf{H_{point}}(B)/b$, we have a set

$$\left[I^p | b^p I^p = \mathbf{H_{point}}(B_{s^p}), s^p \in [0, N/2-1], \forall i,j : s^i \neq s^j\right]_{p=1}^{L}.$$

Using the **mL2SHPoM** and defining the pre-image set as $\left[B_j\right]_{j=0}^{N/2-1}$ and member set as $[I^p]_{p=1}^{L}$ in it, we obtain a proof and convince Verifier that

$$\forall I \in [I^p]_{p=1}^{L} \exists B \in \left[B_j\right]_{j=0}^{N/2-1} : I \sim \mathbf{H_{point}}(B).$$

This is not enough, so we take another instance of the **mL2SHPoM** and define the pre-image set as $\left[B_j\right]_{j=0}^{N/2-1}$, the member set as $[(G + I^p)]_{p=1}^{L}$, and **PointHash** as another ideal point-to-point hash function $\lambda(B) . \left(B + \mathbf{H_{point}}(B)\right)$ in it. From this, we obtain another proof and convince Verifier that

$$\forall (G + I) \in [(G + I^p)]_{p=1}^{L} \exists B \in \left[B_j\right]_{j=0}^{N/2-1} : (G + I) \sim \left(B + \mathbf{H_{point}}(B)\right).$$

Thus, Verifier is convinced of

$$\forall I \in [I^p]_{p=1}^{L} \exists \left(B \in \left[B_j\right]_{j=0}^{N/2-1}, B' \in \left[B_j\right]_{j=0}^{N/2-1}, b, b'\right) : bI = \mathbf{H_{point}}(B) \text{ and } b'(G + I) = \left(B' + \mathbf{H_{point}}(B')\right).$$

From this, Verifier is convinced that

$$\begin{aligned}(b'(bG + bI) = (bB' + b\mathbf{H_{point}}(B'))) \Rightarrow \\ (b'(bG + \mathbf{H_{point}}(B)) = (bB' + b\mathbf{H_{point}}(B'))) \Rightarrow \\ (b'\mathbf{H_{point}}(B) - b\mathbf{H_{point}}(B') = (bB' - bb'G)).\end{aligned}$$

This equality, by definition of ideal hash function, can hold only if $B = B'$ and $b = b'$. Hence, Verifier is convinced that

$$\forall I \in [I^p]_{p=1}^{L} \exists \left(B \in \left[B_j\right]_{j=0}^{N/2-1}, b\right) : \left(bI = \mathbf{H_{point}}(B) \text{ and } b(G + I) = \left(B + \mathbf{H_{point}}(B)\right)\right) \Rightarrow$$
$$\left(B = bG \text{ and } I = \mathbf{H_{point}}(B)/b\right).$$

That is, after accepting both proofs, Verifier is convinced that each point I maps one-to-one to a point B in a subset of the ring, such that Prover knows b in the equality $B = bG$, and I is equal to $\mathbf{H_{point}}(B)/b$.

Here I is a linking tag, as it is uniquely bound to a point B from the ring. The linking tag hides b, and any accepted proof that uses B as an actual signer public key implies disclosure of I. Also, I is called a key-image for B.

8.3.1.1 Optimized signature idea

The above idea implies running the **mL2SHPoM** scheme twice. The optimization below is about running it only once. So, we have to convince Verifier that

$$\forall I \in [I^p]_{p=1}^{L} \exists B \in \left[B_j\right]_{j=0}^{N/2-1}, b : \left(B = bG \text{ and } I = \mathbf{H_{point}}(B)/b\right).$$

For the sake of this, we separate G from I in the member set and B from $\mathbf{H_{point}}(B)$ in the image set using random weighting.

That is, we take a random factor z as a hash of the input parameters, namely, as a hash of all B's and I's, and multiply all I's and $\mathbf{H_{point}}(B)$'s by it in the proof. Next, with a single run of the **mL2SHPoM** we convince Verifier that

$$\forall (G + zI) \in [(G + zI^p)]_{p=1}^{L} \exists B \in \left[B_j\right]_{j=0}^{N/2-1} : (G + zI) \sim \left(B + z\mathbf{H_{point}}(B)\right).$$

From this, Verifier is convinced of

$$\forall I \in [I^p]_{p=1}^{L} \exists \left(B \in \left[B_j\right]_{j=0}^{N/2-1}, b\right) : \left(B = bG \text{ and } I = \mathbf{H_{point}}(B)/b\right).$$

Thus, the signature size is now equal to the size of one **mL2SHPoM** proof. The signature verification complexity is equal to the **mL2SHPoM** proof verification complexity plus L exponentiations for checking the points zI in the member set and plus $N/2$ exponentiations for calculating the points $z\mathbf{H_{point}}(B)$ in the image set.

We optimize the $N/2$ exponentiations for the $z\mathbf{H_{point}}(B)$'s further: we redefine the **mL2SHPoM.PointHash** as $\lambda(B) . \left(B + z\mathbf{H_{point}}(B)\right)$ and let the returned point $\left(B + z\mathbf{H_{point}}(B)\right)$ be lazily evaluated. That is, internally, the **mL2SHPoM.PointHash** (B) becomes returning a 3-tuple $\left(B, z, \mathbf{H_{point}}(B)\right)$ that evaluates to $\left(B + z\mathbf{H_{point}}(B)\right)$ only where the evaluation result is actually consumed. We strictly define a law that regulates the meaning of the phrase 'evaluation result is actually consumed' for it. The law is the following:

- a 3-tuple $\left(B, z, \mathbf{H_{point}}(B)\right)$ doesn't evaluate to $\left(B + z\mathbf{H_{point}}(B)\right)$ when it is moved to or from a vector or another data structure.
- a 3-tuple $\left(B, z, \mathbf{H_{point}}(B)\right)$ doesn't evaluate to $\left(B + z\mathbf{H_{point}}(B)\right)$ when the latter participates, directly or within a vector, as an argument to the $\mathbf{H_{scalar}}$. The $\mathbf{H_{scalar}}$ takes a hash of the 3-tuple in this case.

- a 3-tuple $(B, z, \mathbf{H_{point}}(B))$ evaluates in a special way to $(B + z\mathbf{H_{point}}(B))$ when the latter participates, directly or within a vector, as an argument to the Rsum. In this case, the Rsum calculation is performed as a weighted sum multi-exponentiation, where all weights are calculated prior to the exponentiations. That is, for each lazy $(B, z, \mathbf{H_{point}}(B))$ entry, instead of immediate evaluation of the $(B + z\mathbf{H_{point}}(B))$ the weights for the B and $\mathbf{H_{point}}(B)$ are calculated and, then, a single multi-exponentiation for all entries is performed.
- a 3-tuple $(B, z, \mathbf{H_{point}}(B))$ evaluates to $(B + z\mathbf{H_{point}}(B))$ for all the other cases.

With this law for the lazy evaluation we have the same values for the points and scalars as in the **mL2SHPoM** scheme without the lazy evaluation, except for the values of the challenges that still remain indistinguishable from the values generated by a random oracle. The challenges become the $\mathbf{H_{scalar}}$ hashes of the decoy set $[X_j]_{j=0}^{N-1}$ (and of the other parameters to the $\mathbf{H_{scalar}}$), where even entries of the $[X_j]_{j=0}^{N-1}$ are not evaluated to points and taken as hashes of the 3-tuples instead. As this is performed in the same way on both the Prover's and Verifier's sides, and as z's are the same for all such 3-tuples, the challenges restored in the **Verif** remain equal to the challenges used in the **GetProof**.

Thus, the optimized **mL2SHPoM** scheme remains complete, sound and zero-knowledge. The $N/2$ additional exponentiations required for the $z\mathbf{H_{point}}(B)$'s calculation on the Verifier's side move under the single multi-exponentiation for the $R = \text{Rsum}\left(n, N, [X_j]_{j=0}^{N-1}, [(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n\right)$ in the **Verif**. The verification complexity for the updated **mL2SHPoM** is shown in Table 16.

Table 16: Optimized **mL2SHPoM** verification complexity.

	multi-exp($3N/2$)	single-exp	$\mathbf{H_{point}}$
mL2SHPoM	1	$nL + 4L + 1$	$N + 1$

8.3.2 ML2SLNKSIG LINKABLE SIGNATURE

Using the idea from 8.3.1.1 we define **mL2SLnkSig** linkable signature scheme as four procedures

$$\mathbf{mL2SLnkSig} = \{\mathbf{RingGen}, \mathbf{Sign}, \mathbf{Verif}, \mathbf{Link}\},$$

where:

- **mL2SLnkSig.RingGen** returns a vector $[B_j]_{j=0}^{N/2-1}$ of arbitrary points. These points are only required to be unequal to each other. This procedure contract is the same as for the **mL2SHPoM.PreimageSetGen**.
- **mL2SLnkSig.Sign** takes an actual signer's vector of private keys $\left[(b^p, s^p) \mid b^p G = B_{s^p}, s^p \in [0, N/2-1], \forall i, j : s^i \neq s^j\right]_{p=1}^{L}$, a scalar message m and returns a signature $\left(z, [\sigma^p]_{p=1}^{L}\right)$ on success or 0 on failure. Implementation is shown in Listing 6.

Listing 6: **mL2SLnkSig.Sign** implementation.

```
Input: [(b^p, s^p)]_{p=1}^L          -- private keys
m                                    -- message
Output: (z, [σ^p]_{p=1}^L) or 0      -- signature on success,
                                     -- 0 on failure
Procedure:
    [B_j]_{j=0}^{N/2-1} = RingGen()
    [I^p]_{p=1}^L = [H_point(b^p G) / b^p]_{p=1}^L
    z = H_scalar(m, [B_j]_{j=0}^{N/2-1}, [I^p]_{p=1}^L)
    mL2SHPoM.PreimageSetGen = λ.([B_j]_{j=0}^{N/2-1})
    mL2SHPoM.HashPoint = λ(X).(X + z H_point(X))
    mL2SHPoM.MemberSetGen = λ.([G + z I^p]_{p=1}^L)
    e = H_scalar(z)
    proof = mL2SHPoM.GetProof([(1/b^p, s^p)]_{p=1}^L, e)
    If proof == 0 then Return 0
```

```
    [σ^p]_{p=1}^L = proof
    Return (z, [σ^p]_{p=1}^L)
```

- **mL2SLnkSig.Verif** takes a scalar message m, a signature generated by the **Sign** and returns 0 or $[I^p]_{p=1}^L$, meaning failed or successful verification completion. When $[I^p]_{p=1}^L$ is returned, it contains the key-images used in the signature. Implementation is in Listing 7.

Listing 7: **mL2SLnkSig.Verif** implementation.

```
Input: m                  -- message
(z, [σ^p]_{p=1}^L)        -- signature
Output: [I^p]_{p=1}^L or 0   -- key-images [I^p]_{p=1}^L on successful,
                          -- 0 on failed verification
Procedure:
    [B_j]_{j=0}^{N/2-1} = RingGen()
    [Z_0^p]_{p=1}^L = [σ^p.Z_0]_{p=1}^L        -- extract all Z_0's from the proof
    [I^p]_{p=1}^L = [(Z_0^p - G) /z]_{p=1}^L   -- find all key-images [I^p]_{p=1}^L from Z_0's
    z' = H_scalar(m, [B_j]_{j=0}^{N/2-1}, [I^p]_{p=1}^L)
    If z ≠ z' then Return 0   -- check that z was honestly generated
    mL2SHPoM.PreimageSetGen = λ.([B_j]_{j=0}^{N/2-1})
    mL2SHPoM.HashPoint = λ(X).(X + zH_point(X))
    mL2SHPoM.MemberSetGen = λ.([Z_0^p]_{p=1}^L)
    e = H_scalar(z)
    If mL2SHPoM.Verif([σ^p]_{p=1}^L, e) == 0 then Return 0
    Return [I^p]_{p=1}^L
```

- **mL2SLnkSig.Link** takes a pair $\left([I_0^p]_{p=1}^L, [I_1^p]_{p=1}^L\right)$ of key-image sets returned by two successful **Verif** calls. It returns 1 or 0, meaning the corresponding signatures are linked or not-linked. Implementation is in Listing 8.

Listing 8: **mL2SLnkSig.Link** implementation.

```
Input: ([I_0^p]_{p=1}^L, [I_1^p]_{p=1}^L)   -- two key-image sets from two signatures
Output: 0 or 1          -- 0 means the signatures are not-linked,
                        -- 1 means the signatures are linked
Procedure:
    For j = 1...L:
        If I_0^j ∈ [I_1^p]_{p=1}^L then Return 1
    Return 0
```

A scenario for the **mL2SLnkSig** signature is as follows:

- Prover and Verifier agree on the **mL2SLnkSig.RingGen** to return the same public key ring $[B_j]_{j=0}^{N/2-1}$ on the both sides.
- Prover signs a message m with L private keys $[(b^p, s^p)]_{p=1}^L$ by calling the **mL2SLnkSig.Sign** and obtains a signature $\left(z, [\sigma^p]_{p=1}^L\right)$.
- Verifier takes the message and the signature and calls **mL2SLnkSig.Verif** for them. Iff the call returns $[I^p]_{p=1}^L$, then the Verifier is convinced that Prover signed the message m with the private keys corresponding to some L public keys in the ring and that the vector $[I^p]_{p=1}^L$ contains their key-images. Note, iff Prover signs with a repeating private key, then the vector of key-images contains repeated entries.

- Having performed the above steps two times, Verifier is convinced that two messages were actually signed. Also, Verifier has two vectors $\left[I_0^p\right]_{p=1}^{L}$ and $\left[I_1^p\right]_{p=1}^{L}$ returned by the **mL2SLnkSig.Verif**. Verifier calls **mL2SLnkSig.Link** for them and, iff it returns 1, the Verifier is convinced that there is at least one common private key used for both signatures.

8.3.3 ML2SLNKSIG SCHEME COMPLETENESS, SOUNDNESS AND SIGNER-AMBIGUITY

The **mL2SLnkSig** scheme inherits completeness and soundness from the **mL2SHPoM**. As the **mL2SHPoM** scheme is zero-knowledge, that is proven in 8.2.3, and as the key-images of the form $\mathbf{H_{point}}(bG)/b$ reveal no information about the keys used, which follows from [13] where the same key-image form is proven revealing no information, it is not possible to distinguish signers from the signatures.

The only distinguishable thing about the signers is the case when two or more signatures are signed by a common signer, i.e., the case when the **mL2SLnkSig.Link** returns 1. Even revealing the fact of common signers, the signatures don't reveal any further information about them. Thus, the **mL2SLnkSig** signature scheme is linkable, complete, sound and signer-ambiguous under the DDH.

Note, the **mL2SLnkSig** signature doesn't impose any requirements on the public keys used in its ring, except for that the public keys are to be different. Even knowing a relationship between the public keys, an adversary has no advantage, as the ideal hash function **mL2SLnkSig.HashPoint** breaks any known relationship between them. Hence, we call the **mL2SLnkSig** a general-purpose linkable signature.

8.3.4 ML2SLNKSIG COMPLEXITIES

The **mL2SLnkSig** signature size is the size of its internal **mL2SHPoM** proof plus the size of one scalar z. The **mL2SLnkSig** verification complexity is explained in 8.3.1.1. The size and verification complexity are shown in Tables 17, 18, respectively.

Table 17: **mL2SLnkSig** signature size.

	$\mathbb{G}$	$\mathbb{F}$
mL2SLnkSig	$L(n+4)$	$L(n+2)+1$

Table 18: **mL2SLnkSig** verification complexity.

	multi-exp($3N/2$)	single-exp	$\mathbf{H_{point}}$
mL2SLnkSig	1	$nL+4L+1$	$N+1$

Recalling N commonly denotes a ring size, whereas we use N to denote the internal decoy set size, which is two times larger than the ring size, in Table 19 we provide the same data as in Tables 17, 18 in the common terms. Also, in Table 19 we assume the size of a point from $\mathbb{G}$ is equal to the size of a scalar from $\mathbb{F}$.

Table 19: **mL2SLnkSig** signature size and verification complexity, where:
- N is the ring size
- L is the threshold
- $\boldsymbol{mexp}(3N)$ is the multi-exponentiation of $3N$ summands
- $\mathbf{H_{pt}}(2N+1)$ is $2N+1$ calls to the $\mathbf{H_{point}}$

	Size	Verification complexity
mL2SLnkSig	$2L \cdot \log_2 N + 8L + 1$	$\boldsymbol{mexp}(3N) + L \cdot \log_2 N + 5L + 1 + \mathbf{H_{pt}}(2N+1)$

8.3.5 COMPARISON WITH THE RECENTLY PROPOSED LOG-SIZE SCHEMES

For the comparison we refer to the work of Sarang Noether [14], where proof sizes and verification complexities for some of the recently proposed top-performative schemes are shown in Tables 1, 2 [14].

A direct performance comparison of our **mL2SLnkSig** signature to the schemes analyzed in [14] is not possible due to the following reasons:

- The linkable signature schemes analyzed in [14] includes a proof for a sum of homomorphic commitments as well, whereas our scheme is just a linkable signature.

- Our linkable signature operates with the linking tags of the form $x^{-1}\mathbf{H_{point}}(xG)$, whereas, for instance, Triptych-2 scheme from [14] operates with the linking tags of the form $x^{-1}H$. An additional analysis of the supported security models is probably needed here to compare.

Nevertheless, assuming an $\mathbf{H_{point}}$ call is about ten times faster than an exponentiation, we can see that, for instance, for big N's our signature asymptotic is not far from the RingCT 3.0 and from the Triptych-2 asymptotics

$$\boldsymbol{mexp}(3N) + \mathbf{H_{pt}}(2N)\, vs. \boldsymbol{mexp}(4N) \text{ and vs. } \boldsymbol{mexp}(2N)\text{, respectively.}$$

Although, we have to acknowledge the RingCT 3.0 and the Triptych-2 provide asymptotically better verification time.

The size comparison for the big N's depends on the threshold L: $2L \cdot \log_2 N$ vs. $2 \cdot \log_2(L \cdot N)$ for the RingCT 3.0, and vs. $(L+3) \cdot \log_2 N$ for the Tryptich-2.

Various protocols may scale differently under the real-world conditions. Our **mL2SLnkSig** signature is a general-purpose protocol, so a more elaborated comparison can be made in the future with respect to an application in a particular domain.

Worth to mention, we consider the use of the linking tag form $x^{-1}\mathbf{H_{point}}(xG)$ as an advantage of our signature, as it provides independent random uniform distribution of the tag values regardless of the distribution of the private key values.

We provide a couple of notes below regarding possible modifications to our signature that include a proof for the homomorphic commitment sum and a better verification time. Our estimation is that the homomorphic commitment sum proof will not change the **mL2SLnkSig** verification time asymptotic for big N's. Also, we estimate the **mL2SLnkSig** can be optimized, so that its verification will take asymptotically $\boldsymbol{mexp}(3D) + \mathbf{H_{pt}}(2D)$ time only, where D is a number of distinct public keys in a batch of signatures.

9 POSSIBLE EXTENSION NOTES

9.1 PROOF FOR A SUM OF HOMOMORPHIC COMMITMENTS

It seems not difficult to append a simultaneous proof for the homomorphic commitment sum to the **mL2SLnkSig** linkable signature scheme.

Assuming the homomorphic commitments are built using distinct generators G_1 and G_2, it's possible to add them to the elements of the ring. To separate them back from the L proven members, it would require to extract the parts proportional to G_1 and G_2 along with the parts proportional to G and to $\mathbf{H_{point}}(B)$.

An intuition is that this will require no additional N-size multi-exponentiation, only $\log_2 N$ and L components of the verification complexity will be increased.

9.2 BATCH VERIFICATION

The **mL2SLnkSig** signature verification time grows almost linearly in the ring size *RingN* due to the need of calculating $R = \mathrm{Rsum}\left(n, N, \left[X_j\right]_{j=0}^{N-1}, [(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_{n1}\right)$. This calculation reduces to a multi-exponentiation of $3\mathit{RingN}$ summands with weights composed as multiplications of the scalars from the $([(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_{n1})$ and the scalar z. That is, the verification time is asymptotically $3\mathit{RingN}/\lg(3\mathit{RingN})$.

Suppose, we have d signatures with the ring sizes *RingN* each. Suppose, they have totally *DistinctN* distinct elements in the rings. A question is: is it possible to make the verification time asymptotic to be $3\mathit{DistinctN}/\lg(3\mathit{DistinctN})$ instead of $(3d)\mathit{RingN}/\lg(3\mathit{RingN})$ for this case? Here we have two problems:

- To combine all the Schnorr proofs of $R \sim W$ in the signatures together.
- To combine all the signatures R's into a single multi-exponentiation of $3\mathit{DistinctN}$ summands. The problem is about the odd part of the internal decoy set, which has different counterparts for the same points in different rings.

An intuition is that the first problem can be solved using random weighting, whereas the second problem is solvable with defining the odd part in another way, so that the orthogonality will be kept safe and, at the same time, each point will have a counterpart unchangeable among the decoy sets, that will allow to combine the R's together into $3\mathit{DistinctN}$ summands.

10 CONCLUSION

We have formulated and proven the Lin2-Xor lemma for a primary-order group without bilinear parings, requiring only the discrete logarithm assumption for the group. We have formulated and proven the Lin2-Selector lemma as a generalization of the Lin2-Xor lemma.

These two lemmas allowed us to develop a novel efficient method for convincing a verifier that a given element is a commitment to a linear combination of elements in a pair from a set of orthogonal element pairs.

Using the Lin2-Selector lemma we have built a proof of membership protocol called L2S. We have proven the L2S protocol is complete, sound and zero-knowledge under the decisional Diffie-Hellman assumption.

On the base of the L2S protocol, with the Fiat-Shamir heuristic in the random oracle model we have constructed a non-interactive logarithmic-size zero-knowledge proof of membership scheme called mL2SHPoM.

Using the mL2SHPoM scheme, under the decisional Diffie-Hellman assumption in the random oracle model, we have constructed a setup-free general-purpose logarithmic-size linkable ring signature called mL2SLnkSig that provides signer-ambiguity for a wide range of anonymity sets, including the sets with known to an adversary relationships between elements.

ACKNOWLEDGEMENTS

Author thanks all people, who had occasionally talked with him about privacy systems during this paper writing, and gratefully thanks Olga Kolesnikova for reading the early drafts and making amicable comments on the narrative.

REFERENCES

[1] Masayuki Abe, Miyako Ohkubo, and Koutarou Suzuki. “1-out-of-n signatures from a variety of keys”. In: *ASIACRYPT 2002*. Springer-Verlag. 2002, pp. 415–432.

[2] E. Diamond Benjamin. *“Many-out-of-many” proofs with applications to anonymous Zether*. Tech. rep. Cryptology ePrint Archive, Report 2020/293, 2020. `https://eprint.iacr.org/2020/293`, 2020.

[3] William Black and Ryan Henry. *There Are 10 Types of Vectors (and Polynomials) Efficient Zero-Knowledge Proofs of “One-Hotness” via Polynomials with One Zero*. Tech. rep. Cryptology ePrint Archive, Report 2019/968, 2019. `https://eprint.iacr.org/2020/968` `https://eprint.iacr.org/2020/968`, 2019.

[4] Emmanuel Bresson et al. “A generalization of DDH with applications to protocol analysis and computational soundness”. In: *CRYPTO 2007, LNCS 4622*. Springer. 2007, pp. 482–499.

[5] Benedikt Bünz et al. “Bulletproofs: Short proofs for confidential transactions and more”. In: *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2018, pp. 315–334.

[6] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. “Proofs of partial knowledge and simplified design of witness hiding protocols”. In: *CRYPTO ’94, LNCS 839*. Springer-Verlag. 1994, pp. 174–187.

[7] Reza R Farashahi et al. *Indifferentiable Deterministic Hashing to Elliptic and Hyperelliptic Curves*. Tech. rep. Cryptology ePrint Archive, Report 2010/539, 2010. `https://eprint.iacr.org/2010/539.pdf`, 2010.

[8] Amos Fiat and Adi Shamir. “How To Prove Yourself: Practical Solutions to Identification and Signature Problems”. In: *CRYPTO 1986. Lecture Notes in Computer Science*. Vol. 263. Springer Berlin Heidelberg. 1986, pp. 186–194.

[9] Jens Groth. *On the Size of Pairing-based Non-interactive Arguments*. Tech. rep. Cryptology ePrint Archive, Report 2016/260, 2016. `https://eprint.iacr.org/2016/260.pdf`.

[10] Jens Groth and Markulf Kohlweiss. “One-out-of-many proofs: Or how to leak a secret and spend a coin”. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2015, pp. 253–280.

[11] Daira Hopwood et al. *Zcash protocol specification*. Tech. rep. Tech. rep. 2016–1.10. Zerocoin Electric Coin Company, Tech. Rep., 2016.

[12] Russell WF Lai et al. “Omniring: Scaling private payments without trusted setup”. In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2019, pp. 31–48.

[13] Joseph K Liu, Victor K Wei, and Duncan S Wong. “Linkable Spontaneous Anonymous Group Signature for Ad Hoc Groups (Extended Abstract)”. In: *Proc. Ninth Australasian Conf. Information Security and Privacy (ACISP)*. 2004.

[14] Sarang Noether. *Triptych-2: efficient proofs for confidential transactions*. Tech. rep. Cryptology ePrint Archive, Report 2020/312, 2020. `https://eprint.iacr.org/2020/312/20200315:162105`, 2020.

[15] Ronald L Rivest, Adi Shamir, and Yael Tauman. “How to leak a secret”. In: *Asiacrypt 2001, LNCS 2248*. Springer-Verlag. 2001, pp. 552–565.

[16] Claus-Peter Schnorr. "Efficient Signature Generation by Smart Cards". In: *J. Cryptology* 4.3 (1991), pp. 161–174.

[17] Nicolas Van Saberhagen. *CryptoNote v 2.0*. https://cryptonote.org/whitepaper.pdf. 2013.

[18] Tsz Hon Yuen et al. *RingCT 3.0 for Blockchain Confidential Transaction: Shorter Size and Stronger Security*. Tech. rep. Cryptology ePrint Archive, Report 2019/508, 2019. https://eprint.iacr.org/2019/508, 2019.