# Lin2-Xor Lemma and Log-size Linkable Ring Signature

Anton A. Sokolov

acmxddk@gmail.com

**Abstract**

In this paper we introduce a novel method to 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 are 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.

3$^{rd}$ June, 2020

# 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 telecommunication and in 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 [17], Abe, Ohkubo, and Suzuki [1], Liu, Wei, and Wong [13], an example of a system that uses linkable ring signatures is, for instance, CryptoNote [15]. 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 [6], 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 [10], 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 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 $k_0P_s+k_1Q_s$ and the pair $(P_s, Q_s)$ is taken from a publicly known set of element pairs 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 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 $k_0P_s+k_1Q_s$. Once the lemma's pure $n$-move public coin protocol is successfully completed, the verifier is convinced in both the form $Z=k_0P_s+k_1Q_s$ 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 $Z=k_0P_s+k_1Q_s$ 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 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, 6, 12], that originate from the ideas of Jens Groth and Markulf Kohlweiss [9], Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell [2], our signature scheme is constructed on a basis different from [9, 2].

A parallel can be drawn with the work of Jens Groth and Markulf Kohlweiss [9]. A mechanism resembling the Kronecker's delta is introduced in [9] 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 [9] 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-Xor and Lin2-Selector lemmas

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

$$R=P_1+c_1Q_2+c_4(c_2P_2+c_3Q_2),$$

where $c_1, c_2, c_3, c_4$ are random scalars.

It appears that under certain conditions it's possible to pick elements $Z, H_1, H_2$ and scalars $w, r_1, r_2$ such that

$$wR=Z+r_1H_1+r_2H_2,$$

where $Z$ has the following property: it equals to exactly one of $(aP_1+bQ_1)$ and $(aP_2+bQ_2)$ for some known scalars $a, b$.

That is, $Z$ 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, for instance, of $P_1, P_2, Q_1, Q_2$. Also, there is no possibility for $Z$ to be not a linear combination of $P_1, Q_1, P_2, Q_2$.

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 $(c_1, c_2, c_3, c_4)$, and $(r_1, H_2)$ are to be chosen without knowing $c_4$.

Next, it appears that the Lin2-Xor lemma can be 'stacked', i.e., applied a number of times to an arbitrary number of orthogonal elements. We assume the number of elements is a power of 2. For instance, for eight elements $P_1, Q_1, P_2, Q_2, P_3, Q_3, P_4, Q_4$:

$$R=P_1+c_{11}Q_2+c_{21}(c_{12}P_2+c_{13}Q_2)+c_{31}(c_{22}(P_3+c_{11}Q_3)+c_{23}(c_{12}P_4+c_{13}Q_4)),$$
$$wR=Z+r_1H_1+r_2H_2+r_3H_3,$$

where $Z$ is exactly one of $aP_1+bQ_1$, $aP_2+bQ_2$, $aP_3+bQ_3$, $aP_4+bQ_4$ for some known $a$, $b$.

For a set of $2^{n-1}$ pairs: $\{(P_j, Q_j) \mid j \in [1, 2^{n-1}]\}$, we provide a general method for constructing $R$ such that

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

where $Z = k_0 P_s + k_1 Q_s$ for some $s \in [1, 2^{n-1}]$ and for some known $k_0$, $k_1$. This is the Lin2-Selector lemma protocol. Later on, the actual $s$ is made indistinguishable by keeping the scalars $k_0$ and $k_1$ in secret.

We call the Lin2-Selector lemma protocol a pure protocol, as it provides the bare minimum for convincing a verifier that $Z$ is a commitment to a pair from the set. That is, the protocol is only sound.

### 1.2.2. 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 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 and that it doesn't reveal any information beyond the fact of the 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 the 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 mL2SHPoM non-intaractive many-out-of-many proof of membership scheme and to mL2SLnkSig many-out-of-many linkable ring signature scheme with a linking tag in the form $x^{-1}H_{point}(P)$, where $P = xG$ and $H_{point}$ is a hash 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 $H_{point}$-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 scalar field of $\mathbb{G}$. The field $\mathbb{F}$ is finite, of the same order.

- 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 $A$, $B$, $X$, $P$, $H$, … denote the elements of $\mathbb{G}$. Indices and apostrophes can be appended: $A_1$, $B'$, $X_{12}$, $P_{11}$, $H_1$, … . 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* denotes the zero element of $\mathbb{G}$ and also denotes 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 $\mathbb{G}$, so *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 him.

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 in 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. Sets and vectors.

Sets are assumed finite 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={B₁, B₂, ..., Bₙ}*, then the sequence *B₁, B₂, ..., Bₙ* represents the same set *S*.

### 2.2.2. 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 by Prover.

The phrase "efficiently calculated" means a probabilistic polynomial-time algorithm (PPT) solving the problem with non-negligible probability can be demonstrated. "Polynomial-time" means a polynomial time in the logarithm of cardinality of $\mathbb{F}$.

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~B* and *A!~B* never hold. It's not required for the statements *A~B* and *A!~B* to obey the law of excluded middle, the only assumed law and implication are:

- (not ($A$~$B$ and $A!$~$B$)), meaning that it's not possible simultaneously to know and not to know $x$ in the $A=xB$.

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

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

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

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

For any element $A$ and any finite number of elements $B_1, B_2, \ldots, B_n$, let's denote as
$$A=lin(B_1, B_2, ..., B_n)$$
the following fact: Prover knows or can efficiently calculate $x_1, x_2, \ldots, x_n$, such that
$$A=x_1B_1+x_2B_2+...+x_nB_n.$$
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 $A=x_1B_1+x_2B_2+...+x_nB_n$ 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
$$A!=lin(B_1, B_2, ..., B_n).$$
For any elements $A, B_1, B_2, \ldots, B_n$, both $A=lin(B_1, B_2, ..., B_n)$ and $A!=lin(B_1, B_2, ..., B_n)$ never hold. The law and implication for these statements are similar to those for $A$~$B$ and $A!$~$B$:

- (not ($A=lin(B_1, B_2, ..., B_n)$ and $A!=lin(B_1, B_2, ..., B_n)$))

- (not $A=lin(B_1, B_2, ..., B_n)$) $\Rightarrow$ $A!=lin(B_1, B_2, ..., B_n)$

For any elements $A$ and $B$, there is the following equivalence between the statements:
$$A=lin(B) \text{ is equivalent to } A\text{~}B,$$
$$\text{and } A!=lin(B) \text{ is equivalent to } A!\text{~}B.$$

Due to the constructive nature of the proofs, quantified statements for scalars *"for all $x$ ..."* and *"there exist $y$ ..."* are to be read as *"for any provided $x$ ..."* and *"there is a provided or a known by Prover $y$ ..."* respectively.

### 2.2.3. Orthogonal sets

For any set $S=\{B_1, B_2, ..., B_n\}$ of non-zero elements, we denote the following fact as
$$ort(S)$$
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: $B_i!=lin(S\backslash\{B_i\})$.

For any $S$, $ort(S)$ 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.

### 2.2.4. Evidence

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

For instance, an evidence of $A{\sim}B$ can be simply $x$, such that Verifier can check $A=xB$, or it can be another acceptable way to convince Verifier in $A{\sim}B$, 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 system-wide proofs of statements and proofs of facts that Prover provides to Verifier and the letter checks and accepts.

For all protocols below, if an evidence doesn't pass the 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.

## 2.2.5. 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 $A$ in the beginning of the protocol or, if $A=xB$, by revealing $x$ and $B$ in the beginning.

## 2.2.6. Random choice

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

## 2.2.7. Negligible probability and contradictions

We assume probability to be negligible if its inverse is order of magnitude of the cardinality of $\mathbb{F}$.

Consequently, if by implications we get a statement that holds with 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 a 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.

## 2.2.8. 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, he may own all decoys.

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

The cardinality of a decoy set is assumed to be much less than the cardinality of $\mathbb{F}$. Hence, 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 the 'decoy set', assuming the following semantic difference: the 'decoy sets' are usually parts of low-level protocols, the 'set' is used when talking about a set membership proof, 'ring' is related to a ring signature.

## 2.2.9. Linear combinations

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

$$a_1B_1+a_2B_2+\ldots+a_nB_n\,.$$

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.

# 3. Preliminary lemmas

**NotLin lemma:**

For any three non-zero *A, B, C:* if *A!=lin(B, C)*, then all three statements hold:

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

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

c) Both hold: *A!~B* and *A!~C*

**Proof**:

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

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

c) Suppose *A~B*, then by definition of *A~B* there is provided *x*, such that *A=xB*. That is, by definition of *lin()*, *(A=lin(B, C)* $\Rightarrow$ Contradiction) $\Rightarrow$ *A!~B*. Likewise, *A!~C*.

**OrtUniqueRepresentation lemma:**

For any element *A* and any vector $\vec{B}=[B_i]_{i=1}^n$ of non-zero elements: if $ort(\vec{B})$ and $A=lin(\vec{B})$, 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 $B_j=lin(\vec{B}\setminus\{B_j\})$, however $B_j!=lin(\vec{B}\setminus\{B_j\})$ by definition of the $ort(\vec{B})$ $\Rightarrow$ 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 $a, b$:

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

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

## ZeroRepresentation lemma:

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

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

## OrtDisjunction lemma:

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

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

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

By definition of $lin()$, each $Y_i$ is a weighted sum of elements from $S$, and, from $S_j \cap S_k = \varnothing$, 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 with the ZeroRepresentation lemma. Thus, $ort([Y_i]_{i=0}^n)$.

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 $P!\sim Q$, the following protocol (Table 1.) is an evidence of $(Z=lin(P, Q)$ and $H=lin(P, Q))$:

| | |
|---|---|
| | ⟵ 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 the evidence $(P+cQ)\sim(Z+rH)$ |

Table 1. Lin2 lemma protocol.

**Proof**: Note, the protocol is not claimed to be a sigma-protocol. We have to prove only that

( Verifier succeeds in checking $(P+cQ)\sim(Z+rH)$ ) $\Rightarrow$ ( Prover knows $a, b, x, y$, such that: $Z=aP+bQ$ and $H=xP+yQ$ ).

As *(P+cQ)~(Z+rH)*, Prover knows *t*, such that *P+cQ=tZ+trH*.

Suppose *t=0 ⇒ P+cQ=0 ⇒ P~Q ⇒* Contradiction to *P!~Q ⇒ t≠0*.

Finding *Z* from the above equation: *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)~(Z+r'H)*.

Eliminating *Z*: *(P+cQ)/t-rH=(P+c'Q)/t'-r'H ⇒ (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 ⇒ (c=c') ⇒* Contradiction, as *c* is a random choice.

*(1/t-1/t')P+(c/t-c'/t')Q=0 ⇒ P~Q ⇒* Contradiction to *P!~Q*, as *P~Q* and *P!~Q* can't hold together. Hence, *(r'-r)≠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 ⇒*

*H=lin(P, Q)*. By the OrtUniqueRepresentation lemma: *x=(1/t-1/t')/(r'-r)* and *y=(c/t-c'/t')/(r'-r)*.

*Z=(P+cQ)/t-rH=(1/t)P+(c/t)Q-r(1/t-1/t')/(r'-r)P-r(c/t-c'/t')/(r'-r)Q ⇒ Z=lin(P, Q)*.

Thus, *(Z=lin(P, Q) and H=lin(P, Q))*.

# 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 *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) or b) holds:

    a)  *Z=lin($P_1$, $Q_1$) and $H_1$=lin($P_1$, $Q_1$)*

    b)  *Z=lin($P_2$, $Q_2$) and $H_1$=lin($P_2$, $Q_2$)*

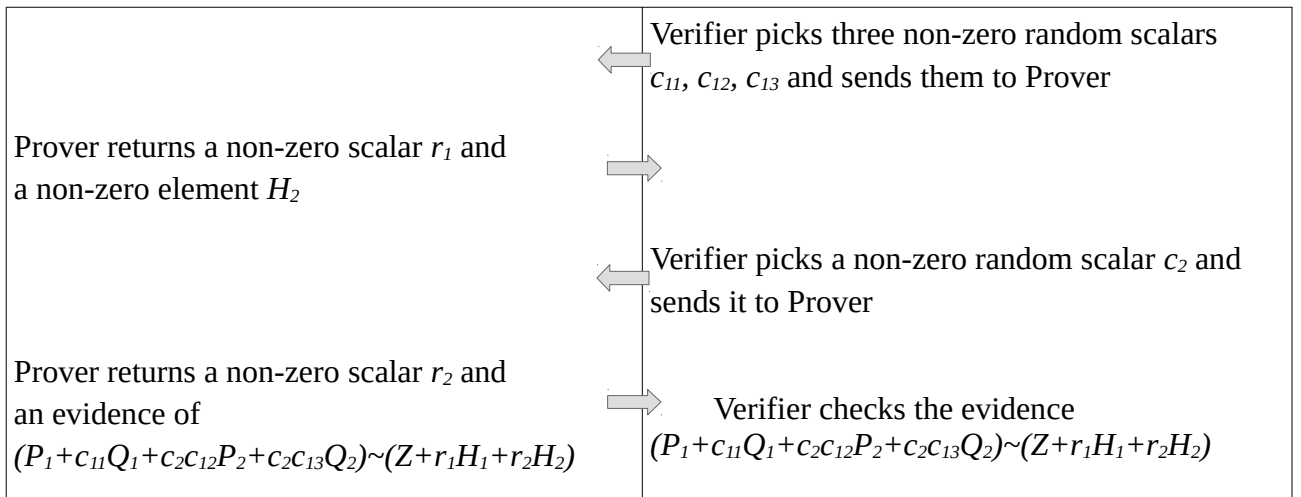| | |
|---|---|
| | Verifier picks three non-zero random scalars $c_{11}$, $c_{12}$, $c_{13}$ and sends them to Prover |
| Prover returns a non-zero scalar $r_1$ and a non-zero element $H_2$ | |
| | Verifier picks a non-zero random scalar $c_2$ and sends it to Prover |
| Prover returns a non-zero scalar $r_2$ and an evidence of $(P_1+c_{11}Q_1+c_2c_{12}P_2+c_2c_{13}Q_2)\sim(Z+r_1H_1+r_2H_2)$ | Verifier checks the evidence $(P_1+c_{11}Q_1+c_2c_{12}P_2+c_2c_{13}Q_2)\sim(Z+r_1H_1+r_2H_2)$ |

Table 2. Lin2-Xor lemma protocol.

**Proof**: Let's move the first two steps of the Lin2-Xor lemma protocol to its premise. Applying the OrtReductionLemma two times, *ort($P_1$, $Q_1$ $P_2$, $Q_2$) ⇒ ort(($P_1+c_{11}Q_1$), ($c_{12}P_2+c_{13}Q_2$)) ⇒* by definition of *ort()*, *($P_1+c_{11}Q_1$)!~($c_{12}P_2+c_{13}Q_2$)*.

After this, we get exactly the premise, protocol and conclusion of the Lin2 lemma with the following substitution (Table 3.):

| Lin2-Xor lemma expressions | Lin2 lemma expressions |
|---|---|
| $c_2$ | $c$ |
| $r_2$ | $r$ |
| $(P_1+c_{11}Q_1)$ | $P$ |
| $(c_{12}P_2+c_{13}Q_2)$ | $Q$ |
| $(Z+r_1H_1)$ | $Z$ |
| $H_2$ | $H$ |
| $(Z+r_1H_1)=lin(P_1+c_{11}Q_1, c_{12}P_2+c_{13}Q_2)$ | $Z=lin(P, Q)$ |

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

Thus, by the conclusion of the Lin2 lemma, Verifier has an evidence of
$(Z+r_1H_1)=lin(P_1+c_{11}Q_1, c_{12}P_2+c_{13}Q_2)$ (*)

Rewriting this evidence using definition of *lin()*, we get
$(Z+r_1H_1)=a(P_1+c_{11}Q_1)+b(c_{12}P_2+c_{13}Q_2)$, where $a$ and $b$ are some scalars known to Prover.

For another challenge $(c_{11}', c_{12}', c_{13}')$, reply $r_1'$ and scalars $a'$ and $b'$:
$(Z+r_1'H_1)=a'(P_1+c_{11}'Q1)+b'(c_{12}'P2+c_{13}'Q2)$

Excluding $H_1$ from both equations and extracting $Z$:
$(a(P_1+c_{11}Q_1)+b(c_{12}P_2+c_{13}Q_2)-Z)/r_1=(a'(P_1+c_{11}'Q_1)+b'(c_{12}'P_2+c_{13}'Q_2)-Z)/r_1'$
$(r_1-r_1')Z=r_1a'(P_1+c_{11}'Q_1)+r_1b'(c_{12}'P_2+c_{13}'Q_2)-r_1'a(P_1+c_{11}Q_1)-r_1'b(c_{12}P_2+c_{13}Q_2)$

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:
$Z=((r_1a'-r_1'a)P_1+(r_1a'c_{11}'-r_1'ac_{11})Q_1+(r_1b'c_{12}'-r_1'bc_{12})P_2+(r_1b'c_{13}'-r_1'bc_{13})Q_2)/(r_1-r_1')$

As *ort(P_1, Q_1 P_2, Q_2)* and as $Z$, $P_1$, $Q_1$ $P_2$, $Q_2$ are fixed by the premise, by the OrtUniqueRepresentation lemma for this equality to hold, all coefficients of $P_1$, $Q_1$ $P_2$, $Q_2$ are to be constants for any choice of $c_{11}$, $c_{12}$, $c_{13}$, $c_{11}'$, $c_{12}'$, $c_{13}'$. Let's designate these constants as $k_1$, $k_2$, $k_3$, $k_4$ and write a system of equalities for them:

$$\begin{cases} k_1=(r_1a'-r_1'a)/(r_1-r_1') \\ k_2=(r_1a'c_{11}'-r_1'ac_{11})/(r_1-r_1') \\ k_3=(r_1b'c_{12}'-r_1'bc_{12})/(r_1-r_1') \\ k_4=(r_1b'c_{13}'-r_1'bc_{13})/(r_1-r_1') \end{cases}$$

Verifier is convinced in that Prover knows the values of the constants $k_1$, $k_2$, $k_3$, $k_4$, as all scalars at the right-hand sides of the equalities are known to Prover.

To simplify calculations, let's define $d'=b'c_{12}'$, $d=bc_{12}$, $e_{13}'=c_{13}'/c_{12}'$, $e_{13}'=c_{13}'/c_{12}'$ also known to Prover, and rewrite:

$$\left\{\begin{array}{l} k_1=(r_1a'-r_1'a)/(r_1-r_1') \\ k_2=(r_1a'c_{11}'-r_1'ac_{11})/(r_1-r_1') \\ k_3=(r_1d'-r_1'd)/(r_1-r_1') \\ k_4=(r_1d'e_{13}'-r_1'de_{13})/(r_1-r_1') \end{array}\right.$$

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

$(r_1-r_1')k_1=(r_1a'-r_1'a) \Rightarrow r_1(a'-k_1)=r_1'(a-k_1) \Rightarrow$

$(a'-k_1)/r_1'=(a-k_1)/r_1$

As the right-hand side of this equality depends only of the first random choice, and the left-hand side depends only of the second choice, both sides are to be equal to some known to Prover constant $q$:

$(a'-k_1)/r_1'=q$ and $(a-k_1)/r_1=q \Rightarrow$

$a'=q*r_1'+k_1$ and $a=q*r_1+k_1$                    (**)

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

$t(r_1a'-r_1'a)=(r_1a'c_{11}'-r_1'ac_{11}) \Rightarrow r_1'a(c_{11}-t)=r_1a'(c_{11}'-t) \Rightarrow$

$a(c_{11}-t)/r_1=a'(c_{11}'-t)/r_1'$                    (***)

As the right-hand side of this equality depends only of the first random choice, and the left-hand side depends only of the second choice, both sides are to be equal to some known to Prover constant $w$:

$a(c_{11}-t)/r_1=w$ and $a'(c_{11}'-t)/r_1'=w \Rightarrow$

$r_1=a(c_{11}-t)/w$ and $r_1'=a'(c_{11}'-t)/w$

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

$wr_1=(q*r_1+k_1)(c_{11}-t)$ and $wr_1'=(qr_1'+k_1)(c_{11}'-t) \Rightarrow$

$r_1(w-q(c_{11}-t))=k_1(c_{11}-t)$ and $r_1'(w-q(c_{11}'-t))=k_1(c_{11}'-t) \Rightarrow$

$r_1=k_1(c_{11}-t)/(w-q(c_{11}-t))$ and $r_1'=k_1(c_{11}'-t)/(w-q(c_{11}'-t))$                    (****)

Thus, $r_1$ and $r_1'$ are expressed through the known to Prover constants and challenges $c_{11}$ and $c_{11}'$.

Suppose $k_3 \neq 0$. Likewise we obtain:
$r_1=k_3*(e_{13}-s)/(u-p*(e_{13}-s))$ and $r1'=k_3*(e_{13}'-s)/(u-p*(e_{13}'-s))$                    (*****)
for some known to Prover constants $s$, $u$, $p$.

If $k_1 \neq 0$ and $k_3 \neq 0$ is the case, then, according to the (****) and (*****), we get contradiction, as $r_1$ gets completely expressed through either of the two independent randomness $c_{11}$ and $e_{13}$.
Thus, the both $k_1 \neq 0$ and $k_3 \neq 0$ never hold together.                    (******)

The following implications hold:

$k_1=0 \Rightarrow$ from the (**): $a'=q*r_1'$ and $a=q*r_1 \Rightarrow a'/r_1'=q$ and $a/r_1=q \Rightarrow$

from the (***): $q(c_{11}-t)=q(c_{11}'-t) \Rightarrow q=0 \Rightarrow a=0$ and $a'=0 \Rightarrow k_2=0$

Likewise, $k_3=0 \Rightarrow d=0$ and $d'=0 \Rightarrow b=0$ and $b'=0 \Rightarrow k_4=0$

Thus, recalling (******), we have: $Z=k_1*P_1+k_2*Q_1+k_3*P_2+k_4*Q_2$, 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 definition of $lin()$, either $Z=lin(P_1, Q_1)$ or $Z=lin(P_2, Q_2)$, never both.

Likewise, either $H_1=lin(P_1, Q_1)$ or $H_1=lin(P_2, Q_2)$, never both.

It's not possible that $(Z=lin(P_1, Q_1)$ and $H_1=lin(P_2, Q_2))$, now we prove it.
Using the evidence (*) from the above, $(Z+r_1H_1)=a(P_1+c_{11}Q_1)+b(c_{12}P_2+c_{13}Q_2)$:

$(Z=lin(P_1, Q_1)$ and $H_1=lin(P_2, Q_2)) \Rightarrow$

Prover knows $z_1, z_2, h_1, h_2$: $(Z=z_1P_1+z_2Q_1$ and $H_1=h_1P_2+h_2Q_2) \Rightarrow$

$z_1P_1+z_2Q_1+r_1(h_1P_2+h_2Q_2)=a(P_1+c_{11}Q_1)+b(c_{12}P_2+c_{13}Q_2) \Rightarrow$

by the OrtUniqueRepresentation lemma: $(z_1=a$ and $z_2=ac_{11}) \Rightarrow z_2/z_1=c_{11}$

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 $(Z=lin(P_2, Q_2)$ and $H_1=lin(P_1, Q_1))$ is not possible.

Hence, either $(Z=lin(P_1, Q_1)$ and $H_1=lin(P_1, Q_1))$ or $(Z=lin(P_2, Q_2)$ and $H_1=lin(P_2, Q_2))$, never both. That is, either of a) and b).

## Corollary of Lin2-Xor lemma:
If the protocol of the Lin2-Xor lemma is successfully completed, then exactly one of the following a) or b) holds:

   a)   $(Z+r_1H_1)\sim(P_1+c_{11}Q_1)$ and $(Z+r_1H_1)!\sim(c_{12}P_2+c_{13}Q_2)$

   b)   $(Z+r_1H_1)\sim(c_{12}P_2+c_{13}Q_2)$ and $(Z+r_1H_1)!\sim(P_1+c_{11}Q_1)$

**Proof**: If $(Z=lin(P_1, Q_1)$ and $H_1=lin(P_1, Q_1))$, then by definition of $lin()$:
$$(Z+r_1H_1)=lin(P_1, Q_1).$$
At the same time, Verifier has the (*) evidence:
$$(Z+r_1H_1)=lin(P_1+c_{11}Q_1, c_{12}P_2+c_{13}Q_2).$$
Combining both, by the OrtUniqueRepresentation lemma, definition of $lin()$ and definition of '$\sim$':
$$(Z+r_1H_1)\sim(P_1+c_{11}Q_1).$$
   Suppose, $(Z+r_1H_1)\sim(c_{12}P_2+c_{13}Q_2)$ holds simultaneously with the above. This is a contradiction to the OrtUniqueRepresentation lemma, as the $(Z+r_1H_1)$ gets two representations: $a(P_1+c_{11}Q_1)$ and $b(c_{12}P_2+c_{13}Q_2)$, where $a$ and $b$ are known to Prover. Hence, $(Z+r_1H_1)!\sim(c_{12}P_2+c_{13}Q_2)$.

   Thus, we have proven that the case a) of the Lin2-Xor lemma implies the case a) of this corollary:
$$(Z+r_1H_1)\sim(P_1+c_{11}Q_1)\text{ and }(Z+r_1H_1)!\sim(c_{12}P_2+c_{13}Q_2).$$

   Likewise, the case b) of the Lin2-Xor lemma implies the case b) of this corollary:
$$(Z+r_1H_1)\sim(c_{12}P_2+c_{13}Q_2)\text{ and }(Z+r_1H_1)!\sim(P_1+c_{11}Q_1).$$

# 5. Lin2-Selector lemma

## 5.1. Preliminary definitions and lemmas

### 5.1.1. Rsum

Let's rewrite the $R = P_1 + c_{11}Q_1 + c_2 c_{12}P_2 + c_2 c_{13}Q_2$ sum that we considered in the Lin2-Xor lemma as the following tree structure (see Figure 1):
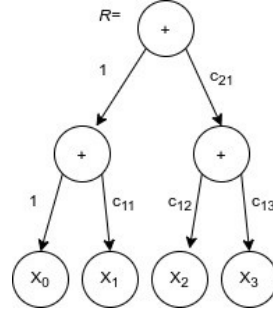


Figure 1. Rsum for four elements.

We have renamed the $P_1, Q_1, P_2, Q_2$ as $X_0, X_1, X_2, X_3$.

Informally, this tree structure is evaluated to $R$ recursively, each node perform summation and each arrow performs multiplication by its tag. Also, 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 $[X_j]_{j=0}^{15}$ it will look as on Figure 2:
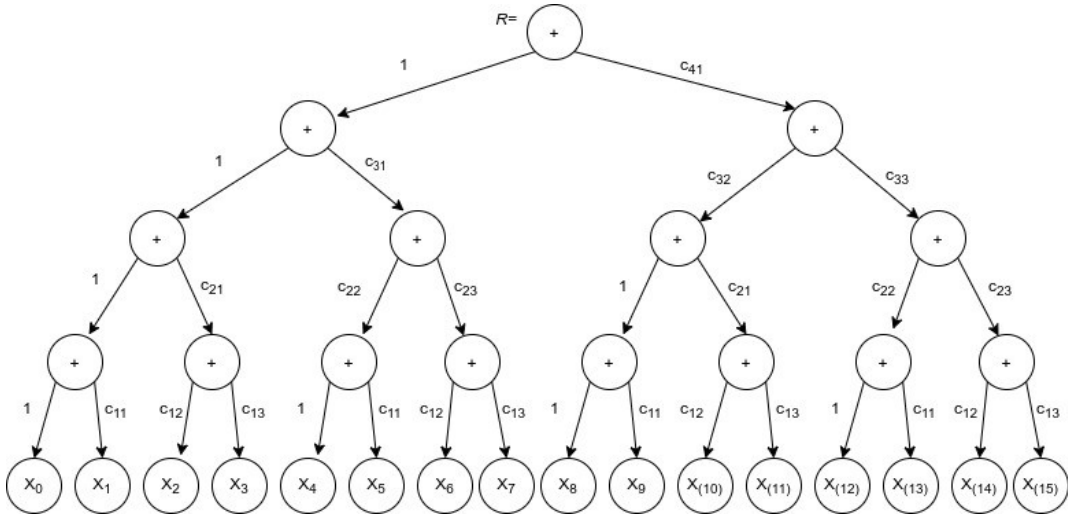


Figure 2. Rsum for sixteen elements.

This is the sum $R=$

$$
\begin{aligned}
&X_0 + &&c_{11}X_1 + &&c_{21}c_{12}X_2 + &&c_{21}c_{13}X_3 + \\
&c_{31}c_{22}X_4 + &&c_{31}c_{22}c_{11}X_5 + &&c_{31}c_{23}c_{12}X_6 + &&c_{31}c_{23}c_{13}X_7 + \\
&c_{41}c_{32}X_8 + &&c_{41}c_{32}c_{11}X_9 + &&c_{41}c_{32}c_{21}c_{12}X_{(10)} + &&c_{41}c_{32}c_{21}c_{13}X_{(11)} + \\
&c_{41}c_{32}c_{22}X_{(12)} + &&c_{41}c_{32}c_{22}c_{11}X_{(13)} + &&c_{41}c_{33}c_{23}c_{12}X_{(14)} + &&c_{41}c_{33}c_{23}c_{13}X_{(15)}
\end{aligned}
$$

14

**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 $[X_j]_{j=0}^{N-1}$, a vector of 3-tuples of scalars $[(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}$, a pair of scalars $(c_{n0}, c_{n1})$, let $\mathrm{Rsum}(n, N, [X_j]_{j=0}^{N-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}, (c_{n0}, c_{n1}))$ be an element, such that:

$$
\begin{aligned}
&\mathrm{Rsum}(n, N, [X_j]_{j=0}^{N-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}, (c_{n0}, c_{n1}))= \\
&\qquad c_{n0}\mathrm{Rsum}(n\text{-}1, N/2, [X_j]_{j=0}^{N/2-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-2}, (1, c_{(n-1),1}))+ \\
&\qquad c_{n1}\mathrm{Rsum}(n\text{-}1, N/2, [X_j]_{j=N/2}^{N-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-2}, (c_{(n-1),2}, c_{(n-1),3})) \\
&\mathrm{Rsum}(1, 2, [X_j]_{j=2k}^{2k+1}, [], (c_{10}, c_{11}))=c_{10}X_{(2k)}+c_{11}X_{(2k+1)}, \text{ where } k\in[0,(N/2)\text{-}1].
\end{aligned}
$$

Informally, for $n>1$, $\mathrm{Rsum}(n, N, [X_j]_{j=0}^{N-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}, (c_{n0}, c_{n1}))$ is a weighted sum of its left and right subtrees, with the weights $c_{n0}$ 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 $c_{10}$, $c_{11}$.

**Rsum property:**

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

$$\mathrm{Rsum}(n, N, [X_j]_{j=0}^{N-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}, (c_{n0}, c_{n1}))=lin([X_j]_{j=0}^{N-1})$$

**RsumOne lemma:**

For any $n>0$, for $N=2^n$, for a vector of $N$ elements $[X_j]_{j=0}^{N-1}$, a vector of 3-tuples of scalars $[(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}$, a pair of scalars $(c_{n0}, c_{n1})$ such that $c_{n0}\neq0$, the following holds:

$$
\begin{aligned}
&\mathrm{Rsum}(n, N, [X_j]_{j=0}^{N-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}, (c_{n0}, c_{n1}))= \\
&\qquad c_{n0}\,\mathrm{Rsum}(n, N, [X_j]_{j=0}^{N-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}, (1, c_{n1}/c_{n0}))
\end{aligned}
$$

**Proof**: By definition of the Rsum, the conclusion follows from the equalities:

For $n=1$:   $\mathrm{Rsum}(1, 2, [X_{(j+2k)}]_{j=0}^{1}, [], (c_{10}, c_{11}))=c_{10}X_{(2k)}+c_{11}X_{(2k+1)}=c_{10}(X_{(2k)}+(c_{11}/c_{10})X_{(2k+1)})=$

$$c_{10}\mathrm{Rsum}(1, 2, [X_{(j+2k)}]_{j=0}^{1}, [], (1, c_{11}/c_{10}))$$

For $n>1$:   $\mathrm{Rsum}(n, N, [X_j]_{j=0}^{N-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}, (c_{n0}, c_{n1}))=$

$$
\begin{aligned}
&c_{n0}\mathrm{Rsum}(n\text{-}1, N/2, [X_j]_{j=0}^{N/2-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-2}, (1, c_{(n-1),1}))+ \\
&c_{n1}\mathrm{Rsum}(n\text{-}1, N/2, [X_j]_{j=N/2}^{N-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-2}, (c_{(n-1),2}, c_{(n-1),3}))= \\
&c_{n0}(\,\mathrm{Rsum}(n\text{-}1, N/2, [X_j]_{j=0}^{N/2-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-2}, (1, c_{(n-1),1}))+ \\
&(c_{n1}/c_{n0})\mathrm{Rsum}(n\text{-}1, N/2, [X_j]_{j=N/2}^{N-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-2}, (c_{(n-1),2}, c_{(n-1),3}))\,)= \\
&c_{n0}\mathrm{Rsum}(n, N, [X_j]_{j=0}^{N-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}, (1, c_{n1}/c_{n0}))
\end{aligned}
$$

Simply stated, according to this lemma, we can extract $c_{n0}$ multiplier from Rsum:

$$\mathrm{Rsum}(\_, (c_{n0}, c_{n1}))=c_{n0}\mathrm{Rsum}(\_, (1, c_{n1}/c_{n0}))$$

## 5.2. Lin2-Selector lemma

**Lin2-Selector lemma:**

For any *n>1* and *N=2ⁿ*, any vector of non-zero fixed elements $[X_j]_{j=0}^{N-1}$, such that $ort([X_j]_{j=0}^{N-1})$ holds, for any non-zero fixed element *Z*, a vector of *n* non-zero elements $[H_i]_{i=1}^{n}$, where $H_1$ is non-zero and fixed, and for a vector of non-zero scalars $[r_i]_{i=1}^{n}$, the following protocol (Table 3.) is an evidence of $Z=lin(X_{(2s)}, X_{(2s+1)})$ for some $s\in[0,N/2-1]$:
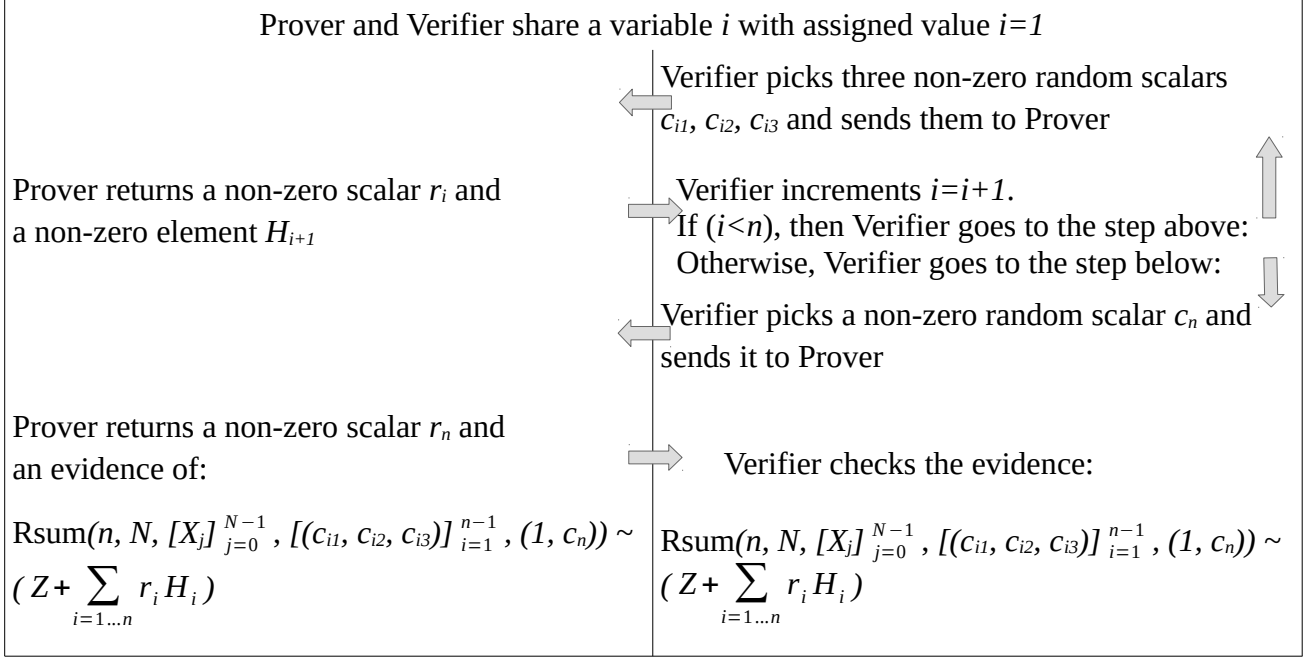
| Prover and Verifier share a variable *i* with assigned value *i=1* | |
|---|---|
| | Verifier picks three non-zero random scalars $c_{i1}$, $c_{i2}$, $c_{i3}$ and sends them to Prover |
| Prover returns a non-zero scalar $r_i$ and a non-zero element $H_{i+1}$ | Verifier increments *i=i+1*. If (*i<n*), 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: $Rsum(n, N, [X_j]_{j=0}^{N-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}, (1, c_n)) \sim$ $(Z+\sum_{i=1...n} r_i H_i)$ | Verifier checks the evidence: $Rsum(n, N, [X_j]_{j=0}^{N-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}, (1, c_n)) \sim$ $(Z+\sum_{i=1...n} r_i H_i)$ |

Table 3. Lin2-Selector lemma protocol.

**Proof**: We prove this lemma by induction for every *n* starting from *2*, recalling *n* is an integer equal to the logarithm of the $[X_j]_{j=0}^{N-1}$ 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 $c_{i1}, c_{i2}, c_{i3}$ triplet generation, where *i=1*.

As

$Rsum(2, 4, [X_j]_{j=0}^{3}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{1}, (1,c_n))=X_0+c_{11}X_1+c_{21}c_{12}X_2+c_{21}c_{13}X_3$,

Verifier has an evidence of

$(X_0+c_{11}X_1+c_{21}c_{12}X_2+c_{21}c_{13}X_3)\sim(Z+r_1H_1+r_2H_2)$

in the last step of the protocol.
By the conclusion of the Lin2-Xor lemma, thus, Verifier has an evidence of either one of

$Z=lin(X_0, X_1)$ and $Z=lin(X_2, X_3)$.

That is, $Z=lin(X_{(2s)}, X_{(2s+1)})$ for some $s\in[0,1]$. 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 $c_{i1}, c_{i2}, c_{i3}$ challenge triplet 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 $[X_j]_{j=0}^{2M-1}$, such that $ort([X_j]_{j=0}^{2M-1})$ 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 4.) is an evidence of $Z=lin(X_{(2s)}, X_{(2s+1)})$ for some $s\in[0,M-1]$:

| Prover and Verifier share a variable $i$ with assigned value $i=1$ | |
|---|---|
| | Verifier picks three non-zero random scalars $c_{i1}$, $c_{i2}$, $c_{i3}$ and sends them to Prover |
| Prover returns a non-zero scalar $r_i$ and a non-zero element $H_{i+1}$ | Verifier increments $i=i+1$. If ($i<m$), then Verifier goes to the step above: Otherwise, Verifier goes to the step below: |
| | Verifier picks three non-zero random scalars $c_{m1}$, $c_{m2}$, $c_{m3}$ and sends them to Prover |
| Prover returns a non-zero scalar $r_m$ and a non-zero element $H_{m+1}$ | |
| | 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: Rsum$(m+1, 2M, [X_j]_{j=0}^{2M-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{m}, (1, c_{m+1})) \sim ( Z + \sum_{i=1\dots(m+1)} r_i H_i )$ | Verifier checks the evidence: Rsum$(m+1, 2M, [X_j]_{j=0}^{2M-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{m}, (1, c_{m+1})) \sim ( Z + \sum_{i=1\dots(m+1)} r_i H_i )$ |

Table 4. Lin2-Selector lemma protocol for n=(m+1).

Let the Rsum$(m+1, 2M, [X_j]_{j=0}^{2M-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{m}, (1, c_{m+1}))$ be rewritten by the definition of the Rsum as a sum of four Rsum's $Y_0, Y_1, Y_2, Y_3$:

Rsum$(m+1, 2M, [X_j]_{j=0}^{2M-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{m}, (1, c_{m+1}))=$

$\qquad$ Rsum$(m, M, [X_j]_{j=0}^{M-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{m-1}, (1, c_{m1}))+$

$\qquad c_{m+1}$Rsum$(m, M, [X_j]_{j=M}^{2M-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{m-1}, (c_{m2}, c_{m3}))=$

$\qquad$ Rsum$(m-1, M/2, [X_j]_{j=0}^{M/2-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{m-2}, (1, c_{(m-1),1}))+$

$\qquad c_{m1}$Rsum$(m-1, M/2, [X_j]_{j=M/2}^{M-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{m-2}, (c_{(m-1),2}, c_{(m-1),3}))+$

$\qquad c_{m+1}c_{m2}$Rsum$(m-1, M/2, [X_j]_{j=M}^{3M/2-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{m-2}, (1, c_{(m-1),1}))+$

$\qquad c_{m+1}c_{m3}$Rsum$(m-1, M/2, [X_j]_{j=3M/2}^{2M-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{m-2}, (c_{(m-1),2}, c_{(m-1),3}))=$

$\qquad Y_0+c_{m1}Y_1+c_{m+1}c_{m2}Y_2+c_{m+1}c_{m3}Y_3$, where:

$\begin{cases} Y_0=\text{Rsum}(m-1, M/2, [X_j]_{j=0}^{M/2-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{m-2}, (1, c_{(m-1),1})) \\ Y_1=\text{Rsum}(m-1, M/2, [X_j]_{j=M/2}^{M-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{m-2}, (c_{(m-1),2}, c_{(m-1),3})) \\ Y_2=\text{Rsum}(m-1, M/2, [X_j]_{j=M}^{3M/2-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{m-2}, (1, c_{(m-1),1})) \\ Y_3=\text{Rsum}(m-1, M/2, [X_j]_{j=3M/2}^{2M-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{m-2}, (c_{(m-1),2}, c_{(m-1),3})) \end{cases}$

By the Rsum property, $Y_0=lin([X_j]_{j=0}^{M/2-1})$, $Y_1=lin([X_j]_{j=M/2}^{M-1})$,
$$Y_2=lin([X_j]_{j=M}^{3M/2-1}), \quad Y_3=lin([X_j]_{j=3M/2}^{2M-1}).$$

As the subsets $[X_j]_{j=0}^{M/2-1}$, $[X_j]_{j=M/2}^{M-1}$, $[X_j]_{j=M}^{3M/2-1}$, $[X_j]_{j=3M/2}^{2M-1}$ of the set $[X_j]_{j=0}^{2M-1}$ don't intersect pairwise, and as $ort([X_j]_{j=0}^{2M-1})$ by the premise, we have $ort(Y_0, Y_1, Y_2, Y_3)$ by the OrtDisjunction lemma. Thus, the evidence in the last step of the protocol rewrites as:
$$Y_0+c_{m1}Y_1+c_{m+1}c_{m2}Y_2+c_{m+1}c_{m3}Y_3 \sim ( Z+ \sum_{i=1...(m+1)} r_i H_i )$$

Defining element $F$ as: $F= Z+ \sum_{i=1...(m-1)} r_i H_i$ , the evidence is
$$Y_0+c_{m1}Y_1+c_{m+1}c_{m2}Y_2+c_{m+1}c_{m3}Y_3 \sim (F+r_m H_m+r_{m+1}H_{m+1})$$

Now, let's look at the step where Verifier picks the challenges $c_{m1}, c_{m2}, c_{m3}$. At that moment, all $c_{i1}, c_{i2}, c_{i3}$ 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 $(Y_0+c_{m1}Y_1+c_{m+1}c_{m2}Y_2+c_{m+1}c_{m3}Y_3) \sim (F+r_m H_m+r_{m+1}H_{m+1})$ 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 $ort(Y_0, Y_1, Y_2, Y_3)$. Verifier picks the challenges $c_{m1}, c_{m2}, c_{m3}$, 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 $(Y_0+c_{m1}Y_1+c_{m+1}c_{m2}Y_2+c_{m+1}c_{m3}Y_3) \sim (F+r_m H_m+r_{m+1}H_{m+1})$.

Hence, if Verifier successfully completes the protocol for $n=(m+1)$, that is, if Verifier accepts that
$$Rsum(m+1, 2M, [X_j]_{j=0}^{2M-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{m}, \ (1, c_{m+1})) \sim ( Z+ \sum_{i=1...(m+1)} r_i H_i ) ,$$

then, by the above, he accepts that
$$Y_0+c_{m1}Y_1+c_{m+1}c_{m2}Y_2+c_{m+1}c_{m3}Y_3 \sim (F+r_m H_m+r_{m+1}H_{m+1}) ,$$
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) or b) holds:

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

b) $(F+r_m H_m)\sim(c_{m2}Y_2+c_{m3}Y_3)$

Here we can rewrite $Y_0+c_{m1}Y_1$ and $c_{m3}Y_2+c_{m3}Y_{12}$ using the definitions of $Y_0, Y_1, Y_2, Y_3$, the definition of Rsum and the RsumOne lemma as
$$Y_0+c_{m1}Y_1=Rsum(m-1, M/2, [X_j]_{j=0}^{M/2-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{m-2}, (1, c_{(m-1),1}))+$$
$$c_{m1}Rsum(m-1, M/2, [X_j]_{j=M/2}^{M-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{m-2}, (c_{(m-1),2}, c_{(m-1),3}))=$$
$$Rsum(m, M, [X_j]_{j=0}^{M-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{m-1}, (1, c_{m1}))$$
$$c_{m2}Y_2+c_{m3}Y_3=c_{m2}Rsum(m-1, M/2, [X_j]_{j=M}^{3M/2-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{m-2}, (1, c_{(m-1),1}))+$$
$$c_{m3}Rsum(m-1, M/2, [X_j]_{j=3M/2}^{2M-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{m-2}, (c_{(m-1),2}, c_{(m-1),3}))=$$
$$Rsum(m, M, [X_j]_{j=M}^{2M-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{m-1}, (c_{m2}, c_{m3}))=$$
$$c_{m2}Rsum(m, M, [X_j]_{j=M}^{2M-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{m-1}, (1, c_{m3}/c_{m2}))$$

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:

a) $(Z+ \sum_{i=1...m} r_i H_i ) \sim \text{Rsum}(m, M, [X_j]_{j=0}^{M-1} , [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{m-1} , (1, c_{m1}))$

b) $(Z+ \sum_{i=1...m} r_i H_i ) \sim c_{m2}\text{Rsum}(m, M, [X_j]_{j=M}^{2M-1} , [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{m-1} , (1, c_{m3}/c_{m2}))$

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

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

If b) holds, then, by definition of '$\sim$', as $c_{m2}$ is a known non-zero scalar, the following holds:

$$(Z+ \sum_{i=1...m} r_i H_i ) \sim \text{Rsum}(m, M, [X_j]_{j=M}^{2M-1} , [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{m-1} , (1, c_{m3}/c_{m2}))$$

As both $c_{m3}$ and $c_{m2}$ are picked uniformly at random, the $c_m=(c_{m3}/c_{m2})$ is also random uniform. Hence, the premise and protocol of this lemma for the case $n=m$ are met, and, by the induction hypothesis, Verifier has an evidence of:

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

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:

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

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

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

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 $n>1$.

## 5.3. Informal explanation of the Lin2-Selector lemma:

Let's start from an informal look at the Lin2-Xor lemma corollary.

The Corollary of Lin2-Xor lemma states that given four orthogonal nodes as leaves and a binary tree composed with random challenges over them, if the tree root node $R$ is proportional to $Z+r_1H_1+r_2H_2$, where $Z$ and $H_1$ are predefined, and $r_1H_1+r_2H_2$ is a sum of the replies, then one of the two tree nodes at depth 1 from the root is proportional to the same sum minus the last reply $r_2H_2$, i.e., to $Z+r_1H_1$.

Note, at the same time, the corollary says that the other of the two tree nodes at depth 1 doesn't have the property to be proportional to $Z+r_1H_1$. So, according to the Corollary of Lin2-Xor lemma, upon completion of its protocol, the property of being proportional to $Z+r_1H_1$ appears to be assigned to exactly one of the two nodes at depth 1.

In this explanation we use the term 'proportional' in the informal sense. More formally speaking, Verifier is convinced in that Prover knows a scalar, such that a node multiplied by that scalar lets the equality to hold.

This is a raw picture, just to represent the idea we use. To be precise, there is a number of details like the order of challenges and the structure of replies to be met too.

Anyway, using this idea, the statement of the Lin2-Selector lemma is as follows: given $2^n$ orthogonal nodes and a binary tree of height $n$ composed with challenges over them, if the tree root

node $R$ is proportional to $Z+r_1H_1+...+r_nH_n$, then $Z$ is proportional to a neighbor pair in the given $2^n$ nodes.

Here we call a node of a binary tree at height 1 from its leaves as a neighbor pair. Using this term, the Corollary of Lin2-Xor lemma states that one of the two neighbor pairs of a binary tree with four leaves is proportional to $Z+r_1H_1$.

In the Lin2-Selector lemma, Prover is not required to be honest or dishonest. The lemma states, that if a Prover, even dishonest, is able to reply to the challenges with some $r_i$'s and $H_1$'s and, finally, to provide an evidence of knowledge of a linear relationship for two elements calculated on the Verifier's side with these challenges and replies, then with the overwhelming probability the Prover knows $k_1$, $k_2$ in the equation $Z=k_0X_{(2s)}+k_1X_{(2s+1)}$ for some $s \in [0,N/2-1]$.

That is, the lemma states, that the probability of generating a sequence of replies satisfying the final evidence check is negligible, unless Prover knows the two scalars $k_0$, $k_1$ and index $s$.

From the lemma follows, that once the sum $Z+r_1H_1+...+r_nH_n$ is built according to the lemma protocol and the evidence check is passed, the possible known decomposition forms for $Z$ appear to be limited to the single one. A decomposition like, for instance, $Z=k_0X_{(2s)}+k_1X_{(2s+1)}+k_2X_{2t}$ is proven unfeasible.

To prove the Lin2-Selector lemma, we start with proving that four nodes at depth 2 from $R$ are orthogonal.

Next, we consider the following substitution: $(Z+r_1H_1+...+r_{n-2}H_{n-2}) \rightarrow Z$,

$$r_{n-1} \rightarrow r_1, r_n \rightarrow r_2,$$
$$H_{n-1} \rightarrow H_1, H_n \rightarrow H_2,$$

we find that the new $Z$ and $H_1$ are fixed, and we apply the Corollary of Lin2-Xor lemma to the subtree of depth 2 from the root.

After that, we have the initial tree split into its left and right subtrees, each composed over the left and right halves of the $2^n$ initial nodes. From the Corollary of Lin2-Xor lemma, we have that one of roots of these subtrees has the property of being proportional to the initial $Z+r1H1+...+r_{n-1}H_{n-1}$.

Proceeding *(n-2)* times splitting the subtrees, we get to a neighbor pair in the given $2^n$ nodes, that is, we get to the conclusion of the Lin2-Selector lemma.

Actually, in the formal proof above using induction we prove this a bit differently, in the reverse order. Although, splitting the subtrees is more illustrative.

The Lin2-Selector lemma doesn't specify what neighbor pair we get to, it states that we certainly get to one of the $2^n/2$ pairs only, as each split guarantees that exactly one of the left and right subtrees has the necessary property to be split further.

In other words, the Lin2-Selector lemma paves a hidden path that goes from the root of the challenge tree to one of the $2^n/2$ neighbor leave pairs, such that all nodes along the path are proportional to an incremental reply from Prover.

# 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 that no possibility exists for identifying a pair in the set the element $Z$ corresponds to.

## 6.1. Com2 commitment

**Com2 definition:**

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

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

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

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 the Com2 commitment $Z$ a commitment to a member-pair. The set of member-pairs $[X_j]_{j=0}^{N-1}$ is called a decoy set.

## 6.2. L2S membership proof protocol

We define **L2S** PoM protocol as four procedures
**L2S**={**DecoySetGen**, **ComGen**, **InteractionProcedure**, **Verif**},
where:

- **DecoySetGen**($n$) is an arbitrary function that returns an element vector $\vec{X} = [X_j]_{j=0}^{N-1}$ of $N=2^n$ elements, such that $ort(\vec{X})$ holds. Each element in the generated $\vec{X}$ has an independent of the other elements in the same $\vec{X}$ distribution indistinguishable from 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 elements distributions within the vector and their uniform randomness are to be guaranteed.

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

- **InteractionProcedure** is depicted in Table 5. 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 $([(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}, (1, c_n), Z, [(r_i, H_i)]_{i=1}^{n}, c, T, t)$, that contains $Z$ together with all the challenges and replies occurred during the Prover and Verifier interaction.

| Prover and Verifier common parameters: | |
|---|---|
| • $n$, $N=2^n$, | |
| • a set of elements $[X_j]_{j=0}^{N-1} = \textbf{DecoySetGen}(n)$ | |
| • a shared variable $i$ with assigned value $i=1$ | |

| Prover: | Verifier: |
|---|---|
| $(k_0, k_1, s)$, where: $k_0 \neq 0$, $s \in [0, N/2-1]$ | $Z$ |
| | |
| $w=k_0$ | |
| $k=k_1/w$ | |
| $M=N$ | |
| $[Y_j]_{j=0}^{M-1} = [X_j]_{j=0}^{N-1}$ | |
| $(z, h)=(2s, InvertLastBit(2s))$ | |
| $a=1$ | |
| $q \leftarrow$ random, non-zero | |
| $H_1=wY_h/q$ $\qquad H_1 \Rightarrow$ | |
| | $(c_{i1}, c_{i2}, c_{i3}) \leftarrow$ random, non-zero |
| $c_{i0}=1$ | |
| $r_i=q((c_{i,(h\%4)}/c_{i,(z\%4)})-k)$ $\qquad r_i \Rightarrow$ | |
| $k=0$ | |
| $M=M/2$ | |
| $[Y_j]_{j=0}^{M-1} = [(c_{i,(2j\%4)}Y_{(2j)}+c_{i,((2j+1)\%4)}Y_{(2j+1)})/c_{i,(z\%4)}]_{j=0}^{M-1}$ | |
| $a=c_{i,(z\%4)}a$ | |
| $(z, h)=(z//2, InvertLastBit(z//2))$ | |
| $q \leftarrow$ random, non-zero | |
| $H_{i+1}=wY_h/q$ $\qquad H_{i+1} \Rightarrow$ | |
| | $i=i+1$ |
| | If $(i<n)$, then: |
| | Otherwise: |
| | $c_n \leftarrow$ random, non-zero |
| $(c_{n0}, c_{n1})=(1, c_n)$ | |
| $r_n=q(c_{n,h}/c_{n,z})$ $\qquad r_n \Rightarrow$ | |
| $a=c_{n,z}a$ | |
| $x=a/w$ | |
| $q \leftarrow$ random, non-zero | |
| $W=(k_0X_{2s}+k_1X_{2s+1}) + \sum\limits_{i=1...n} r_iH_i$ | |
| $T=qW$ $\qquad T \Rightarrow$ | |
| | $c \leftarrow$ random, non-zero |
| $t=q-xc$ $\qquad t \Rightarrow$ | |
| | Verifier has a tuple: |
| | $([(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}, (1, c_n), Z, [(r_i, H_i)]_{i=1}^{n}, c, T, t)$ |

Table 5. **L2S.InteractionProcedure**.

- **Verif** function is shown in Table 6. It takes the tuple that Verifier has upon completion of the **InteractionProcedure** procedure together with the decoy set from the **DecoySetGen**. It returns 1 or 0, meaning the verification is completed successfully or failed.

| Input: $n$, $[X_j]_{j=0}^{N-1}$, $([(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}, (1, c_n), Z, [(r_i, H_i)]_{i=1}^{n}, c, T, t)$, where $N=2^n$ |
|---|
| $R=\text{Rsum}(n, N, [X_j]_{j=0}^{N-1}, [(1, c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}, (1, c_n))$ <br> $W= Z+ \sum_{i=1...n} r_i H_i$ <br> If $(tW+cR)==T$ then return 1 <br> Else return 0. |

<div align="center">Table 6. <strong>L2S.Verif</strong> function.</div>

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 $([(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}, (1, c_n), Z, [(r_i, H_i)]_{i=1}^{n}, c, T, t)$.

- 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 protocol completion means $Z==\text{Com2}(k_0, k_1, s, \vec{X})$.

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)*. We use the *InvertLastBit* for indexes to switch between the left and right subtrees of a binary tree node.

## 6.2.1. Proof for the equality Rsum$(n, N, [X_j]_{j=0}^{N-1}, [(1, c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}, (1, c_n))=xW$

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

The expression
$$[Y_j]_{j=0}^{M-1} =[X_j]_{j=0}^{N-1} \text{, where } M=N,$$
in the beginning of the Prover's part of the **L2S.InteractionProcedure** lets all $Y_j$'s to be $X_j$'s.

Next, down the protocol execution flow, when $i=1$, the expression
$$[Y_j]_{j=0}^{M-1} =[(c_{i,(2j\%4)}Y_{(2j)}+c_{i,((2j+1)\%4)}Y_{(2j+1)})/c_{i,(z\%4)}]_{j=0}^{M-1} \text{, where } M=N/2,$$
lets the $Y_j$'s vector to contain $N/2$ Rsum's:

Rsum$(1, 2, [X_t]_{t=2j}^{2j+1}, [], (c_{1,(2j\%4)}, c_{1,((2j+1)\%4)}))$, each divided by the common factor $c_{1,(2s\%4)}$.

In the next line, the variable $a$ becomes that common factor from the above: $a=c_{1,(2s\%4)}$.

When $i=2$, the expression
$$[Y_j]_{j=0}^{M-1} =[(c_{i,(2j\%4)}Y_{(2j)}+c_{i,((2j+1)\%4)}Y_{(2j+1)})/c_{i,(z\%4)}]_{j=0}^{M-1} \text{, where } M=N/4,$$
lets the $Y_j$'s vector to contain $N/4$ Rsum's:

Rsum$(2, 4, [X_t]_{t=4j}^{8j-1}, [(c_{d,0}, c_{d,1}, c_{d,2}, c_{d,3})]_{d=1}^{1}, (c_{2,(2j\%4)}, c_{2,((2j+1)\%4)}))$ divided

by the common factor $c_{1,(2s\%4)}c_{2,(s\%4)}$.

Note, for all $d$ the $c_{d,0}$ is always 1.

In the next line, the variable $a$ accumulates the common factor: $a=c_{1,(2s\%4)}c_{2,(s\%4)}$.

When $i=3$, the expression

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

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

Rsum$(3, 8, [X_t]_{t=8j}^{16j-1}, [(c_{d,0}, c_{d,1}, c_{d,2}, c_{d,3})]_{d=1}^{2}, (c_{3,(2j\%4)}, c_{3,((2j+1)\%4)}))$ divided

by the common factor $c_{1,(2s\%4)}c_{2,(s\%4)}c_{3,((s//2)\%4)}$.

In the next line, the variable $a$ accumulates the common factor: $a=c_{1,(2s\%4)}c_{2,(s\%4)}c_{3,((s//2)\%4)}$.

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 a 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:

$H_1=wX_{(2s+1)}/q$ , where $q$ is random

$r_1=q((c_{1,((2s+1)\%4)}/c_{1,(2s\%4)})-k)$ , where $q$ is the same and $k=k_1/w$, so that

$$(wX_{2s}+r_1H_1)=w\text{Rsum}(1, 2, [X_t]_{t=2s}^{2s+1}, [], (c_{1,(2s\%4)}, c_{1,((2s+1)\%4)}))/c_{1,(2s\%4)}$$

Next, Prover reshuffles q, sets $h=InvertLastBit(s)$ and sends:

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

When $i=2$, Prover has $k$ set to zero forever and sends:

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

$$(wX_{2s}+r_1H_1+r_2H_2)=w\text{Rsum}(2, 4, [X_t]_{t=4(s/2)}^{8(s/2)-1}, [(1, c_{d,1}, c_{d,2}, c_{d,3})]_{d=1}^{1},$$
$$(c_{2,(2(s/2)\%4)}, c_{2,((2(s/2)+1)\%4)}))/(c_{1,(2s\%4)}c_{2,(s\%4)})$$

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

$H_3=w\text{Rsum}(2, 4, [X_t]_{t=4h}^{8h+1}, [(1, c_{d,1}, c_{d,2}, c_{d,3})]_{d=1}^{1}, (c_{2,(2h\%4)}, c_{2,((2h+1)\%4)}))/c_{1,(2s\%4)}/q$

When $i=3$, Prover sends:

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

$$(wX_{2s}+r_1H_1+r_2H_2+r_3H_3)=w\text{Rsum}(2, 4, [X_t]_{t=8(s/4)}^{16(s/4)-1}, [(1, c_{d,1}, c_{d,2}, c_{d,3})]_{d=1}^{2},$$
$$(c_{3,(2(s/4)\%4)}, c_{3,((2(s/4)+1)\%4)}))/(c_{1,(2s\%4)}c_{2,(s\%4)}c_{3,((s/2)\%4)})$$

And so on, until $i=n$ and

$$W=(wX_{2s}+r_1H_1+r_2H_2+...+r_nH_n)=w\text{Rsum}(n, N, [X_j]_{j=0}^{N-1}, [(1, c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}, (1, c_n))/a$$

Thus, Rsum$(n, N, [X_j]_{j=0}^{N-1}, [(1, c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}, (1, c_n))=xW$.

## 6.2.2. Proof that $Z=Com2(k_0, k_1, s, [X_j]_{j=0}^{N-1})$ implies L2S.Verif returns 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 $R=xW$.

24

The $Z=Com2(k_0, k_1, s, [X_j]_{j=0}^{N-1})$ makes the $W$ calculated on the Prover's side and in the **L2S.Verif** 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, Rsum$(n, N, [X_j]_{j=0}^{N-1}, [(1, c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}, (1, c_n))=xW$. 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, $Z=Com2(k_0, k_1, s, [X_j]_{j=0}^{N-1})$ 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 $Com2(k_0, k_1, s, [X_j]_{j=0}^{N-1})$, 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

$$\text{Rsum}(n, N, [X_j]_{j=0}^{N-1}, [(1, c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}, (1, c_n)) \sim ( Z + \sum_{i=1...n} r_i H_i )$$

Thus, by the Lin2-Selector lemma, if the **L2S.Verif** returns 1, then Verifier is convinced that $Z=lin(X_{(2s)}, X_{(2s+1)})$ for some member-pair $(X_{(2s)}, X_{(2s+1)})$, 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 the opening $(k_0, k_1, s)$ of the commitment $Z=Com2(k_0, k_1, s, [X_j]_{j=0}^{N-1})$, where $s$ corresponds to a member-pair in the decoy set.

We have proven the **LS2** protocol is sound.

## 6.3.3. Structure and view of the L2S Prover-Verifier public transcript

The Prover-Verifier public transcript is the following tuple

$$([(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}, (1, c_n), Z, [(r_i, H_i)]_{i=1}^{n}, c, T, t).$$

Of course, the constant *1* in $(1, c_n)$ is not a part of the transcript, we keep it here for coherence and assume it is not written into the transcript.

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

$$(Z, [(r_i, H_i)]_{i=1}^{n}) \quad \text{and} \quad ([X_j]_{j=0}^{N-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}, (1, c_n))$$

respectively. Thus, the element $T$ is a linear combination of the variables seen for anyone. Hence, we exclude $T$ from our consideration: for any accepted by Verifier transcript 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 $r_iH_i$. These products represent Rsum's, i.e., the subtree sums at heights $i$. That is, for each height $i$, the element $(Z+r_1H_1+...+r_{i-1}H_{i-1})$ corresponds to a subtree that the index $s$ belongs to. At the same time, the element $r_iH_i$ 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, however this difference has no effect on the transcript items independence and uniformity.

All the elements $Z, r_1H_1, \dots , r_iH_i$ 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 $r_iH_i$ is a linear combination of the elements from the $[X_j]_{j=0}^{N-1}$ with efficiently computable scalar coefficients. Moreover, all $r_iH_i$'s depend on the different non-intersecting subsets of the $[X_j]_{j=0}^{N-1}$.

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

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

The coefficients of these polynomials are efficiently computable from the $[(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}$, $c_n$, and $k_1$. Thus, reducing the question of the $r_iH_i$'s distribution to the (P,Q)-DDH problem [4], we have

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

By the (P,Q)-DDH assumption, the distributions of all the $r_iH_i$'s and $Z$ are indistinguishable from $\{e_iG\}_{i=1}^{n+1}$, 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 $r_iH_i$'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 $[X_j]_{j=0}^{N-1}$ generated by the **L2S.DecoySetGen**. For clarity, we are omitting the word 'indistinguishable', implying it for the distributions.

For the 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

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

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

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

By requiring $w$ be different for each transcript, meaning same $Z$ is never used in any two different conversations, we obtain that all the $r_iH_i$'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 the honest transcripts, they

serve only to distinguish the 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 [5]. We use the natural extrapolation of the sHVZK definition to the $n$-round public-coin protocols.

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 the honest conversation transcripts with the same challenges.

The simulator acts as follows:
- It takes an empty L2S transcript placeholder and puts the given input $Z$ and the given challenges $[(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}$, $(1, c_n)$ in the placeholder, recalling the $1$ is not actually placed into the placeholder.
- It independently generates and puts random uniform scalars 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 $[(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}$, $(1, c_n)$, $Z$, $[(r_i, H_i)]_{i=1}^{n}$, $c$, $t$ from the already filled in places of the placeholder, obtains $[X_j]_{j=0}^{N-1}$ by calling **L2S.DecoySetGen**, calculates

$$R = \text{Rsum}(n, N, [X_j]_{j=0}^{N-1}, [(1, c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}, (1, c_n))$$
$$W = Z + \sum_{i=1\ldots n} r_i H_i,$$

and puts the 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**.

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 that have the items of the simulated transcript, except for the item $T$. However, $T$ is calculated the same way from the same sources for the honest and for the simulated transcripts, hence the algorithms 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, as long as 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 $s$

Here we prove, that the $s$ 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 the 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 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 contradiction.

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

### 6.3.6. Side note about special soundness

We have already proven the properties that we need in for the further. Anyway, interestingly, there exists a possibility to prove special soundness for the **L2S** protocol extrapolating the definition of special soundness by R.Cramer et al. [5] to *n*-round protocols.

The extrapolation is that 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 the same witness, with different challenges, and with the same first Prover's message represent two conversations where the Prover's random tape and witness are fixed and the Verifier's random tapes is reshuffled. For the 3-round protocols the extrapolated definition reduces exactly to the special soundness definition in [5].

To comply to this definition, the L2S protocol is extended with the first message containing Prover's random tape commitment, that Prover sends to Verifier in 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 witness $(k_0, k_1, s)$.

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 $k_0$.
  - Knowing $k_0$ and *x* it tries to find $k_1$ from the *Z*, $r_1H_1$, $X_{2s}$, $X_{2s+1}$ and corresponding challenges.
- One of the parallel guesses ends up with $(k_0, k_1)$ successfully found. Thus, the witness is extracted.

# 7. L2S protocol extensions

## 7.1. iL2S protocol, sHVZK for not-random input

As shown in 6.3.4., the **L2S** is sHVZK under the DDH, as long as the scalar $k_0$ in the Prover's input $(k_0, k_1, s)$ has independent and random 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 to track the known relationships between them, however with the sHVZK the adversary is not able to obtain from the transcripts any information beyond that.

The idea of the input randomization is that right in 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 provides to Verifier 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:

**iL2S**={**DecoySetGen=L2S.DecoySetGen**, **ComGen**, **InteractionProcedure**, **Verif**},

where

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

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

- **iL2S.InteractionProcedure** is depicted in Table 7. It starts with Prover having $(k_0, k_1, s)$, $k_0 \neq 0$, and Verifier having $Z$.

  On completion of the **iL2S.InteractionProcedure**, Verifier has two tuples: $(Z_0, c_0, T_0, t_0)$ and $([(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}, (1, c_n), Z, [(r_i, H_i)]_{i=1}^{n}, c, T, t)$, 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.

<div style="border:1px solid">

Prover and Verifier common parameters:
- $n$, $N=2^n$,
- a set of elements $[X_j]_{j=0}^{N-1} = $ **DecoySetGen**$(n)$

| Prover: | Verifier: |
|---|---|
| $(k_0, k_1, s)$, where: $k_0 \neq 0$ | $Z$ |
| $s$ – secret index, $s \in [0, N/2-1]$ | |
| $Z_0 = \text{Com2}(k_0, k_1, s, [X_j]_{j=0}^{N-1})$ | $Z_0 = Z$ |
| $f \leftarrow$ random, non-zero | |
| $Z = fZ_0$     $\quad Z \Longrightarrow$ | Verifier assigns the received value to $Z$ |
| $(k_0, k_1, s) = (fk_0, fk_1, s)$ | |
| $q \leftarrow$ random, non-zero | |
| $T_0 = qZ_0$     $\quad T_0 \Longrightarrow$ | |
| $\Longleftarrow c_0 \leftarrow$ random, non-zero | |
| $t_0 = q-fc_0$     $\quad t_0 \Longrightarrow$ | |

Run **L2S.InteractionProcedure** for the new $(k_0, k_1, s)$ and $Z$

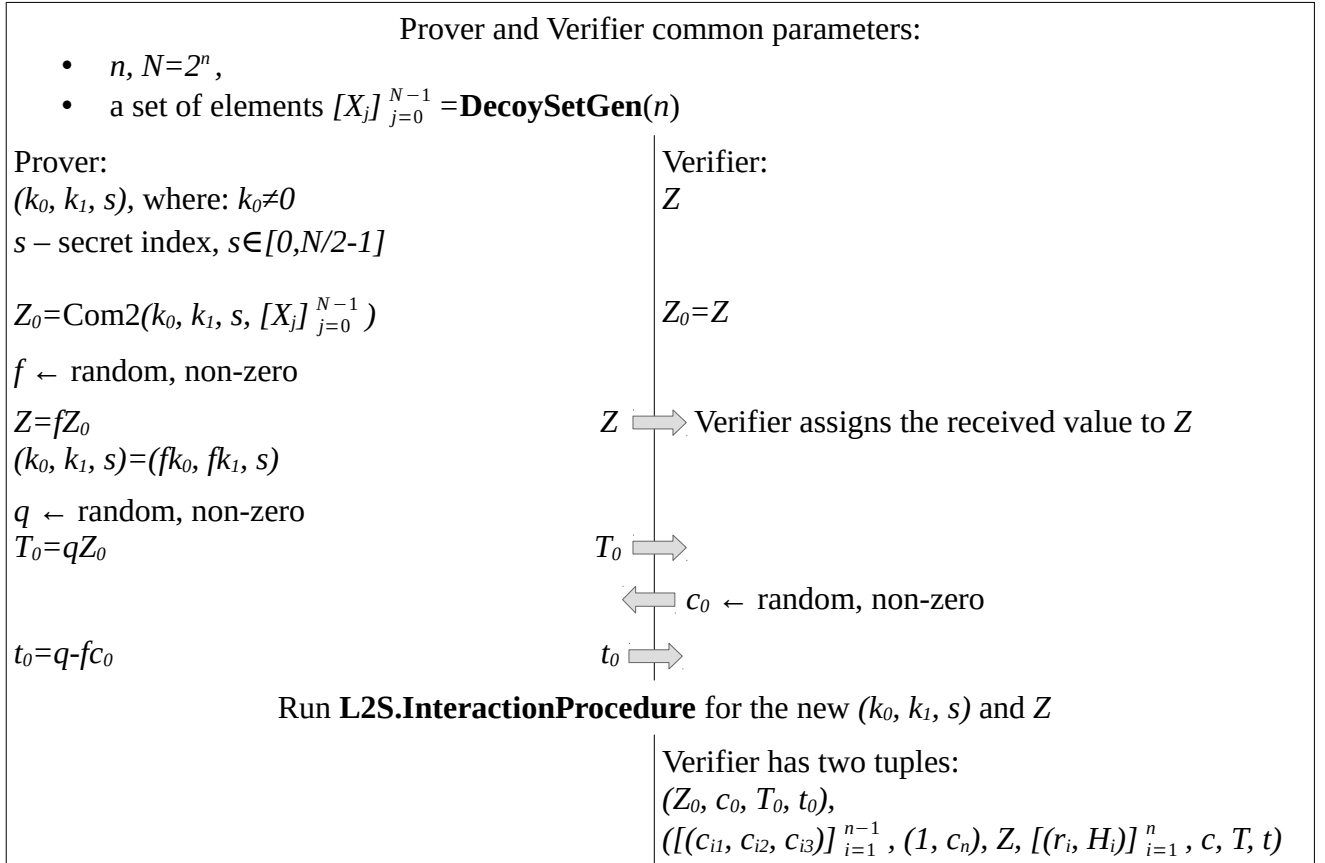| | Verifier has two tuples: |
|---|---|
| | $(Z_0, c_0, T_0, t_0)$, |
| | $([(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}, (1, c_n), Z, [(r_i, H_i)]_{i=1}^{n}, c, T, t)$ |

</div>

Table 7. **iL2S.InteractionProcedure**.

- **iL2S.Verif** function is shown in Table 8. It takes the two tuples from the **iL2S.InteractionProcedure** procedure together with the decoy set from the **DecoySetGen** and returns *1* or *0*.

| |
|---|
| Input: $n$, $[X_j]_{j=0}^{N-1}$, where $N=2^n$, $(Z_0, c_0, T_0, t_0)$, $([(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}, (1, c_n), Z, [(r_i, H_i)]_{i=1}^{n}, c, T, t)$ |
| If $(t_0 Z_0 + c_0 Z) == T_0$ then continue <br> Else return 0 <br><br> Run **L2S.Verif** for the <br> $n$, $[X_j]_{j=0}^{N-1}$, $([(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}, (1, c_n), Z, [(r_i, H_i)]_{i=1}^{n}, c, T, t)$ |

Table 8. **iL2S.Verif** function.

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 into the beginning of the public Prover-Verifier transcripts and as the $Z$ in the transcript becomes $Z=fZ_0$, it's necessary to determine the distributions of them:
    - $c_0$ is an independent and random 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 random uniform scalar $q$ in $t_0=q-fc_0$.
    - $Z_0$, which is the input commitment, 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 the 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 items, except for $T_0$, distinguishably dependent on $Z_0$ in the transcript. All the other items are independent and uniformly random.

- The **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 $T$.

## 7.2.  mL2S protocol

A parallel variant 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:

**mL2S**={**DecoySetGen=L2S.DecoySetGen**, **ComGen=iL2S.ComGen**,

    **MapInteractionProcedure**, **JoinVerif**},

where:

- **mL2S.MapInteractionProcedure** is depicted in Table 9. It starts with Prover having $L$ openings $[(k_0^p, k_1^p, s^p) \mid k_0^p \neq 0]_{p=1}^{L}$, and Verifier having $L$ commitments $[Z^p]_{p=1}^{L}$.

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

  $((Z_0^p, c_0, T_0^p, t_0^p), ([(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}, (1, c_n), Z^p, [(r_i^p, H_i^p)]_{i=1}^{n}, c, T^p, t^p))_{p=1}^{L}$, that contain the outputs of $L$ **iL2S.InteractionProcedure** parallel runs with the common decoy set and challenges.

| Prover and Verifier common parameters: | |
|---|---|
| • $L$ <br> • $n$, $N=2^n$ | |
| Prover: <br> $[(k_0^p, k_1^p, s^p) \mid k_0^p \neq 0]_{p=1}^{L}$ | Verifier: <br> $[Z^p]_{p=1}^{L}$ |
| For each $p \in [1,L]$: run **iL2S.InteractionProcedure** using $n$, $(k_0^p, k_1^p, s^p)$ as arguments for Prover, and $n$, $Z^p$ as arguments for Verifier. <br> All the parallel **iL2S.InteractionProcedure** instances share the same decoy set $[X_j]_{j=0}^{N-1} =$ **DecoySetGen**$(n)$ and same Verifier's challenges $c_0$, $[(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}$, $(1, c_n)$, $c$ | |
| | Verifier has $L$ tuples: <br> $[((Z_0^p, c_0, T_0^p, t_0^p), ([(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}, (1, c_n), Z^p, [(r_i^p, H_i^p)]_{i=1}^{n}, c, T^p, t^p))]_{p=1}^{L}$ |

Table 9. **mL2S.MapInteractionProcedure**.

- **mL2S.JoinVerif** function is shown in Table 10. It takes the $L$ tuples from the **mL2S.InteractionProcedure** procedure together with the decoy set from the **DecoySetGen** and returns 1 or 0.

| Input: $L$, $n$, $[X_j]_{j=0}^{N-1}$, where $N=2^n$, <br> $((Z_0^p, c_0, T_0^p, t_0^p), ([(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}, (1, c_n), Z^p, [(r_i^p, H_i^p)]_{i=1}^{n}, c, T^p, t^p))_{p=1}^{L}$ |
|---|
| $R = $Rsum$(n, N, [X_j]_{j=0}^{N-1}, [(1, c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}, (1, c_n))$ <br><br> For each $p \in [1,L]$: run **iL2S.Verif** using $n$, $[X_j]_{j=0}^{N-1}$ and <br> $(Z_0^p, c_0, T_0^p, t_0^p), ([(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}, (1, c_n), Z^p, [(r_i^p, H_i^p)]_{i=1}^{n}, c, T^p, t^p)$ as arguments. <br><br> Inside each **iL2S.Verif** call, within nested **L2S.Verif** call, use the calculated above $R$ for the **iL2S.Verif.L2S.Verif**.$R$ <br><br> Return 0 if one of the **iL2S.Verif** calls returns 0. Otherwise, return 1. |

Table 10. **mL2S.JoinVerif** function.

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, in the beginning, and uses the calculated value $R = $Rsum$(n, N, [X_j]_{j=0}^{N-1}, [(1, c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}, (1, c_n))$ for them.

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:

**MapInteractionProcedure** → **InteractionProcedure**,

**JoinVerif** → **Verif**

### 7.2.1.  mL2S protocol completeness, soundness and sHVZK

The **mL2S** protocol completeness and soundness immediately follows 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 one addition below.

The space of the honest **mL2S** transcripts is the space of the honest **iL2S** transcripts with the only difference in that it is partitioned by the transcripts belonging to the same **mL2S** proof. Each partition contains the **iL2S** transcripts with the same challenges. Nevertheless, all their items, except for these challenges and discussed above $Z_0$, $T_0$, $T$, 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.

### 7.2.2.  mL2S protocol complexities

Recalling the **mL2S** transcript

$$((Z_0^p, c_0, T_0^p, t_0^p), ([(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}, (1, c_n), Z^p, [(r_i^p, H_i^p)]_{i=1}^{n}, c, T^p, t^p))_{p=1}^{L},$$

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 11.

|          | $\mathbb{G}$ | $\mathbb{F}$ |
|----------|:--------:|:--------:|
| **mL2S** | $L(n+3)$ | $L(n+2)$ |

Table 11. **mL2S** transmitted data amount.

The $R=\mathrm{Rsum}(n, N, [X_j]_{j=0}^{N-1}, [(1, c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}, (1, c_n))$ 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 $[X_j]_{j=0}^{N-1}$ are calculated as scalar-scalar multiplications and, after that, a single multi-exponentiation of the elements from the $[X_j]_{j=0}^{N-1}$ to their respective coefficients is performed.

The **mL2S** verification complexity is shown in Table 12, where $N=2^n$:

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

Table 12. **mL2S** verification complexity.

# 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 indistinguishable from independent random uniform distribution of elements, we employ a 'point-to-point' hash function $H_{point}(...)$ 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 $\mathbb{G}$ thereafter. Thus, we use the term 'points' instead of the 'elements', they become equivalent after this.

**Any to scalar hash function $H_{scalar}(...)$ definition:**

We call $H_{scalar}(...)$ 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 $H_{point}(...)$ definition:**

We call $H_{point}()$ 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 $H_{scalar}$ it can be, for instance, SHA-3. For the $H_{point}$ it can be, for instance, a function described in [7].

**Integers $n, N, L$:**

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

- $N>1$ is a number of decoys, $N$ is a power of $2$, and $N/2$ 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 $[X_j]_{j=0}^{N-1}$. We reshape this vector to be a vector of pairs $[(P_j, Q_j)]_{j=0}^{N/2-1}$ below.

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

**Procedure substitution and lambda function:**

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

# 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: $[B_j]_{j=0}^{N/2-1}$ and $[A^p]_{p=1}^{L}$, Prover knows a vector of scalar-integer pairs:

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

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 $[\mathbf{H_{point}}(B_j)] \, _{j=0}^{N/2-1}$.

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:

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

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 $Z_0$ that the **mL2S** Verifier knows beforehand.

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

## 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 $[B_j] \, _{j=0}^{N/2-1}$ 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 it is an ideal point-to-point hash function.

```
Input: B
Output: A point-hash of B

Procedure:
    Return Hpoint(B)
```

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

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

```
Input: none
Output: image set [Pj] N/2-1 j=0 , HashPoint mapped to the pre-images

Procedure:
    [Bj] N/2-1 j=0 =PreimageSetGen()
    [Pj] N/2-1 j=0 =[HashPoint(Bj)] N/2-1 j=0
    Return [Pj] N/2-1 j=0
```

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

- **mL2SHPoM.MemberSetGen** returns a vector $[A^p] \, _{p=1}^{L}$ of points that are going to be proven to be members of the image set returned by the **GetImageSet** multiplied by some known to Prover scalar coefficients.

- **mL2SHPoM.GetDecoySet** returns a decoy set $[X_j]_{j=0}^{N-1}$ for use in the proof. Even elements of the $[X_j]_{j=0}^{N-1}$ 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 the Listing 3.

```
Input: none
Output: decoy set [Xⱼ] ᴺ⁻¹ⱼ₌₀
Procedure:
    [Pⱼ] ᴺ/²⁻¹ⱼ₌₀ =GetImageSet()
    [Aᵖ] ᴸₚ₌₁ =MemberSetGen()
    Qshift=Hscalar([Aᵖ] ᴸₚ₌₁ ,[Pⱼ] ᴺ/²⁻¹ⱼ₌₀ )G
    [Qⱼ] ᴺ/²⁻¹ⱼ₌₀ =[Hpoint(Qshift+Pⱼ)] ᴺ/²⁻¹ⱼ₌₀
    [Xⱼ] ᴺ⁻¹ⱼ₌₀ =Flatten([(Pⱼ, Qⱼ)] ᴺ/²⁻¹ⱼ₌₀ )
    Return [Xⱼ] ᴺ⁻¹ⱼ₌₀
```

Listing 3. **mL2SHPoM.GetImageSet** implementation.

- **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 $[\sigma^p]_{p=1}^{L}$, meaning a non-interactive proof, or $0$ on error. The **GetProof** is the **mL2S.MapInteractionProcedure** translated to non-interactive setting. Specification is in Listing 4.

```
Input:    [(vᵖ, sᵖ)] ᴸₚ₌₁    --private keys
          e                 --scalar seed
Output:   [σᵖ] ᴸₚ₌₁  or 0    --proof, vector of σ's on success,
                            --0 on failure

Procedure:
  • Let [Xⱼ] ᴺ⁻¹ⱼ₌₀ =GetDecoySet()
  • Let [Aᵖ] ᴸₚ₌₁ =MemberSetGen()
  • Ensure the private keys correspond to the member set elements:
    For p=1...L:
        If Aᵖ≠ vᴾ X₂ₛᵖ then Return 0
  • Let [(k₀ᵖ, k₁ᵖ, sᵖ)] ᴸₚ₌₁ =[(vᵖ, 0, sᵖ)] ᴸₚ₌₁
  • [Z₀ᵖ] ᴸₚ₌₁ =[Aᵖ] ᴸₚ₌₁
  • Run all L iL2S.InteractionProcedure's in parallel with the
    [(k₀ᵖ, k₁ᵖ, sᵖ)] ᴸₚ₌₁ and [Z₀ᵖ] ᴸₚ₌₁ as arguments. Stop all them at
    the point, where the first challenge c₀ is to be obtained.
    At that moment the values [(Z₀ᵖ, T₀ᵖ, Zᵖ)] ᴸₚ₌₁ have already been
    calculated.
  • Calculate e=Hscalar(e, [Xⱼ] ᴺ⁻¹ⱼ₌₀ , [(Z₀ᵖ, T₀ᵖ, Zᵖ)] ᴸₚ₌₁ )
  • Let c₀=e
  • Continue all the L parallel procedures to the point, where
    the challenge tuple (c₁₁, c₁₂, c₁₃) is to be obtained.
    At that moment the [t₀ᵖ] ᴸₚ₌₁ and [H₁ᵖ] ᴸₚ₌₁ have already been
    calculated.
  • Calculate e=Hscalar(e, [t₀ᵖ] ᴸₚ₌₁ , [H₁ᵖ] ᴸₚ₌₁ )
  • Let (c₁₁, c₁₂, c₁₃)=(e, Hscalar(e), Hscalar(e+1))
  • Continue all the L parallel procedures to the point, where
    the challenge tuple (c₂₁, c₂₂, c₂₃) is to be obtained.
```

```
        At that moment the [r₁ᵖ] L_{p=1} and [H₂ᵖ] L_{p=1} have already been
        calculated.
    •   Calculate e=Hscalar(e, [r₁ᵖ] L_{p=1}, [H₂ᵖ] L_{p=1} )
    •   Let (c₂₁, c₂₂, c₂₃)=(e, Hscalar(e), Hscalar(e+1))
    •   And so on...,
        until all values [(Z₀ᵖ, T₀ᵖ, Zᵖ, t₀ᵖ, [(rᵢᵖ, Hᵢᵖ)] n_{i=1}, Tᵖ, tᵖ)] L_{p=1} and
        (c₀, [(cᵢ₁, cᵢ₂, cᵢ₃)] n−1_{i=1}, cₙ, c) are calculated.
    •   Let [σᵖ] L_{p=1} =[(Z₀ᵖ, T₀ᵖ, Zᵖ, t₀ᵖ, [(rᵢᵖ, Hᵢᵖ)] n_{i=1}, Tᵖ, tᵖ)] L_{p=1}
    •   Return [σᵖ] L_{p=1}
```

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

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

```
Input:   [σᵖ] L_{p=1}       --proof, a vector of σ's
         e                  --scalar seed, same as used for GetProof call
Output: 0 or 1              --the verification is failed or completed ok

Procedure:
    •   Let [Xⱼ] N−1_{j=0} =GetDecoySet()
    •   Extract the values of [(Z₀ᵖ, T₀ᵖ, Zᵖ)] L_{p=1} from the [σᵖ] L_{p=1}
    •   Calculate e=Hscalar(e, [Xⱼ] N−1_{j=0}, [(Z₀ᵖ, T₀ᵖ, Zᵖ)] L_{p=1} )
    •   Let c₀=e
    •   Extract the values of [t₀ᵖ] L_{p=1} and [H₁ᵖ] L_{p=1} from the [σᵖ] L_{p=1}
    •   Calculate e=Hscalar(e, [t₀ᵖ] L_{p=1}, [H₁ᵖ] L_{p=1} )
    •   Let (c₁₁, c₁₂, c₁₃)=(e, Hscalar(e), Hscalar(e+1))
    •   Extract the values of [r₁ᵖ] L_{p=1} and [H₂ᵖ] L_{p=1} from the [σᵖ] L_{p=1}
    •   Calculate e=Hscalar(e, [r₁ᵖ] L_{p=1}, [H₂ᵖ] L_{p=1} )
    •   Let (c₂₁, c₂₂, c₂₃)=(e, Hscalar(e), Hscalar(e+1))
    •   And so on...,
        until all values (c₀, [(cᵢ₁, cᵢ₂, cᵢ₃)] n−1_{i=1}, cₙ, c) are restored.
        At this moment all values of [(Z₀ᵖ, T₀ᵖ, Zᵖ, t₀ᵖ, [(rᵢᵖ, Hᵢᵖ)] n_{i=1}, Tᵖ, tᵖ)] L_{p=1}
        are extracted from the [σᵖ] L_{p=1} .
    •   For p=1...L:
            If (t₀ᵖZ₀ᵖ+c₀Zᵖ)≠T₀ᵖ then Return 0
    •   Calculate R=Rsum(n, N, [Xⱼ] N−1_{j=0}, [(1, cᵢ₁, cᵢ₂, cᵢ₃)] n−1_{i=1}, (1, cₙ))
    •   For p=1...L:
            Calculate W= Zᵖ+ Σ_{i=1...n} rᵢᵖ Hᵢᵖ
            If (tᵖW+cR)≠Tᵖ then Return 0
    •   Return 1
```

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

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

- Prover and Verifier agree on the scheme implementation, particularly, on the set returned by the **PreimageSetGen** and on the **HashPoint** function.

- 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 $[P_j]_{j=0}^{N/2-1}$ 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 $[Z_0^p]_{p=1}^{L}$ from the $[\sigma^p]_{p=1}^{L}$. The set $[Z_0^p]_{p=1}^{L}$ 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 in that Prover knows the private keys, that connect each element of the set $[Z_0^p]_{p=1}^{L}$ to an element of the $[P_j]_{j=0}^{N/2-1}$.

### 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 13. The scalar seed is not accounted, as it can have an agreed between Prover and Verifier value, e.g., be fixed as $e=0$.

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

Table 13. **mL2SHPoM** proof size.

The **mL2SHPoM** verification complexity is shown in Table 14, where $N=2^n$. We use the same optimization for the Rsum calculation, as in the **mL2S**. The scalar-scalar multiplications and $H_{scalar}$ calls are assumed taking negligible amount of the computational time.

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

Table 14. **mL2SHPoM** verification complexity.

## 8.3. Linkable ring signature based on the mL2SHPoM

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

### 8.3.1. Realization idea

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

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

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

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

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

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

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

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

Thus, Verifier is convinced in

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

From this, Verifier is convinced in: $(\ b'(bG+bI) = (bB'+b\boldsymbol{H_{point}}(B'))\ ) \Rightarrow$

$(\ b'(bG+\boldsymbol{H_{point}}(B)) = (bB'+b\boldsymbol{H_{point}}(B'))\ ) \Rightarrow (\ b'\boldsymbol{H_{point}}(B) - b\boldsymbol{H_{point}}(B') = (bB'-bb'G)\ ).$

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

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

$(\ B = bG\ \text{and}\ I = \boldsymbol{H_{point}}(B)/b\ ).$

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 set, such that Prover knows $b$ in the equality $B=bG$, and $I$ is equal to $\boldsymbol{H_{point}}(B)/b$.

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

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

For the sake of this, we separate $G$ from $I$ in the member set and $B$ from $\boldsymbol{H_{point}}(B)$ 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 $\boldsymbol{H_{point}}(B)$'s by it in the proof. Next, with a single run of the **mL2SHPoM** we convince Verifier that

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

From this, Verifier is convinced in

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

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 $z\boldsymbol{H_{point}}(B)$ in the image set.

We optimize the $N/2$ exponentiations for $z\boldsymbol{H_{point}}(B)$'s in the image set further: we redefine the **mL2SHPoM.PointHash** as $\lambda(B).(B+z\boldsymbol{H_{point}}(B))$ and let the returned point $(B+z\boldsymbol{H_{point}}(B))$ be lazily evaluated.

That is, internally, the **mL2SHPoM.PointHash**$(B)$ becomes returning a 3-tuple $(B, z, \boldsymbol{H_{point}}(B))$ that evaluates to $(B+z\boldsymbol{H_{point}}(B))$ 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 following:

- a 3-tuple *(B, z, $H_{point}(B)$)* doesn't evaluate to *(B+z$H_{point}(B)$)* when it is moved to or from a vector or other data structure.

- a 3-tuple *(B, z, $H_{point}(B)$)* doesn't evaluate to *(B+z$H_{point}(B)$)* when the latter participates, directly or within a vector, as an argument to the *$H_{scalar}$*. The *$H_{scalar}$* takes a hash of the 3-tuple in this case.

- a 3-tuple *(B, z, $H_{point}(B)$)* evaluates in a special way to *(B+z$H_{point}(B)$)* 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, $H_{point}(B)$)* entry, instead of an immediate evaluation of the *(B+z$H_{point}(B)$)* the weights for the *B* and *$H_{point}(B)$* are calculated and, then, a single multi-exponentiation for all entries is performed. Of course, *z* contributes to a weight for *$H_{point}(B)$*.

- a 3-tuple *(B, z, $H_{point}(B)$)* evaluates to *(B+z$H_{point}(B)$)* 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 **$H_{scalar}$** hashes of the decoy set *$[X_j]_{j=0}^{N-1}$*, 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$H_{point}(B)$*'s calculation on the Verifier's side move under the single multi-exponentiation for the *R*=Rsum*(n, N, $[X_j]_{j=0}^{N-1}$, $[(1, c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}$, (1, $c_n$))* in the **Verif**. The verification complexity for the updated **mL2SHPoM** is shown in Table 15.

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

Table 15. Optimized **mL2SHPoM** verification complexity.

## 8.3.2. mL2SLnkSig linkable signature

Using the idea from 8.3.1.1. we define **mL2SLnkSig** linkable signature scheme as four procedures:
**mL2SLnkSig**={**RingGen**, **Sign**, **Verif**, **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. The procedure contract is the same as for the **mL2SHPoM.PreimageSetGen**.

- **mL2SLnkSig.Sign** takes an actual signer's vector of private keys $[(b^p, s^p) \mid b^pG= B_{s^p}, s^p \in [0,N/2-1], \forall\ i,j: s^i \neq s^j\ ]\ _{p=1}^{L}$, a scalar message $m$ and returns a signature $(z, [\sigma^p]\ _{p=1}^{L})$ on success or $0$ on failure. Implementation is shown in Listing 6.

```
Input:    [(bᵖ, sᵖ)] ᴸₚ₌₁              -- private keys
          m                            -- message
Output:  (z, [σᵖ] ᴸₚ₌₁ ) or 0         -- signature on success,
                                       -- 0 on failure

Procedure:
    [Bⱼ] ᴺ/²⁻¹ⱼ₌₀ =RingGen()
    [Iᵖ] ᴸₚ₌₁ =[Hpoint(bᵖG)/bᵖ] ᴸₚ₌₁

    z=Hscalar(m, [Bⱼ] ᴺ/²⁻¹ⱼ₌₀ , [Iᵖ] ᴸₚ₌₁ )

    mL2SHPoM.PreimageSetGen=λ.([Bⱼ] ᴺ/²⁻¹ⱼ₌₀ )
    mL2SHPoM.HashPoint=λ(X).(X+zHpoint(X))
    mL2SHPoM.MemberSetGen=λ.([G+zIᵖ] ᴸₚ₌₁ )

    e=Hscalar(z)
    proof=mL2SHPoM.GetProof([(1/bᵖ, sᵖ)] ᴸₚ₌₁ , e)
    If proof==0 then Return 0
    [σᵖ] ᴸₚ₌₁ =proof

    Return (z, [σᵖ] ᴸₚ₌₁ )
```

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

- **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.

```
Input:   m                             -- message
         (z, [σᵖ] ᴸₚ₌₁ )               -- signature
Output:  [Iᵖ] ᴸₚ₌₁  or 0              -- key-images [Iᵖ] ᴸₚ₌₁ on successful,
                                       -- 0 on failed verification

Procedure:
    [Bⱼ] ᴺ/²⁻¹ⱼ₌₀ =RingGen()

    [Z₀ᵖ] ᴸₚ₌₁ =[σᵖ.Z₀] ᴸₚ₌₁      -- extract all Z₀'s from the proof
    [Iᵖ] ᴸₚ₌₁ =[(Z₀ᵖ-G)/z] ᴸₚ₌₁   -- find all key-images [Iᵖ] ᴸₚ₌₁ from Z₀'s

    z'=Hscalar(m, [Bⱼ] ᴺ/²⁻¹ⱼ₌₀ , [Iᵖ] ᴸₚ₌₁ )
    If z≠z' then Return 0   -- check that z was honestly generated

    mL2SHPoM.PreimageSetGen=λ.([Bⱼ] ᴺ/²⁻¹ⱼ₌₀ )
    mL2SHPoM.HashPoint=λ(X).(X+zHpoint(X))
    mL2SHPoM.MemberSetGen=λ.([Z₀ᵖ] ᴸₚ₌₁ )

    e=Hscalar(z)
    If mL2SHPoM.Verif([σᵖ] ᴸₚ₌₁ , e)==0 then Return 0
```

```
        Return [Iᵖ] L_{p=1}
```
Listing 7. **mL2SLnkSig.Verif** implementation.

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

```
Input:    ([I₀ᵖ] L_{p=1}, [I₁ᵖ] L_{p=1})   -- 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 i=1...L:
        If I₀ⁱ ∈ [I₁ᵖ] L_{p=1} then Return 1
    Return 0
```
Listing 8. **mL2SLnkSig.Link** implementation.

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

- Prover and Verifier agree on the **mL2SLnkSig.RingGen** to be returning the same public key ring $[B_j]_{j=0}^{N/2-1}$ on 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 $(z, [\sigma^p]_{p=1}^{L})$.

- 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 in that Prover signed the message *m* with the private keys corresponding to some *L* public keys in the ring and 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 in that two messages were actually signed. Also, Verifier has two vectors $[I_0{}^p]_{p=1}^{L}$ and $[I_1{}^p]_{p=1}^{L}$ 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 the 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 $H_{point}(bG)/b$ reveal no information about the keys used, that 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 to the public keys used in its ring, except for 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 16, 17.

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

<p align="center">Table 16. <strong>mL2SLnkSig</strong> signature size.</p>

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

<p align="center">Table 17. <strong>mL2SLnkSig</strong> verification complexity.</p>

Recalling $N$ commonly denotes a ring size, whereas we use $N$ to denote the internal decoy set size that is two times larger than the ring, in Table 18 we provide the same data as in Tables 16, 17 in the common terms. Also, in Table 18 we assume the size of a point from $\mathbb{G}$ is equal to the size of a scalar from $\mathbb{F}$.

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

<p align="center">Table 18. <strong>mL2SLnkSig</strong> signature size and verification complexity, where:</p>

- $N$ is the ring size
- $L$ is the threshold
- $mexp(3N)$ is the multi-exponentiation of $3N$ summands
- $H_{pt}(2N+1)$ is $2N+1$ calls to the $H_{point}$

### 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.

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 $x^{-1}H_{point}(xG)$, 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 $H_{point}$ 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

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

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 to a particular domain.

We provide a couple of notes below regarding the 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 for big $N$'s. Also, we estimate the **mL2SLnkSig** can be optimized, so that its verification will take asymptotically *$mexp(3D) + H_{pt}(2D)$* 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.

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 *$H_{point}(B)$*.

An intuition is that this will require no additional $N$-size multi-exponentiation, the 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 $R = \text{Rsum}(n, N, [X_j]_{j=0}^{N-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}, (c_{n0}, c_{n1}))$. This calculation reduces to a multi-exponentiation of *$3 \cdot RingN$* summands with weights composed as multiplications of the scalars from the $[(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}, (c_{n0}, c_{n1})$ and a scalar $z$. That is, the verification time is asymptotically *$3 \cdot RingN / \lg(3 \cdot RingN)$*.

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 *$3 \cdot DistinctN / \lg(3 \cdot DistinctN)$* instead of *$3 \cdot d \cdot RingN / \lg(3 \cdot RingN)$* 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 *$3 \cdot DistinctN$* summands. The problem is about the odd part of the internal decoy set, that 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 between the decoy sets, that will allow to combine the *R*'s into *3·DistinctN* summands.

# 10. Conclusion

We have formulated and proven 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 Lin2-Selector lemma as a generalization for the Lin2-Xor lemma.

These two lemmas allowed a novel efficient method for convincing a verifier that a given element is a commitment to a linear combination of a pair of elements 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 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 the elements, under the decisional Diffie-Hellman assumption in the random oracle model.

# References

[1] M. Abe, M. Ohkubo, and K. Suzuki. 1-out-of-n signatures from a variety of keys. In Proc. ASIACRYPT 2002, pages 415–432. Springer-Verlag, 2002. LNCS Vol. 2501.

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

[3] William Black, Ryan Henry. There are 10 Types of Vectors (Polynomials): Efficient Zero-Knowledge Proofs of "One-Hotness" via Polynomials with One Zero. Cryptology ePrint Archive, Report 2019/968, 2019. https://eprint.iacr.org/2020/968.

[4] Emmanuel Bresson, Yassine Lakhnech, Laurent Mazaré, Bogdan Warinschi. A Generalization of DDH with Applications to Protocol Analysis and Computational Soundness. CRYPTO 2007, LNCS 4622, pp 482-499.

[5] Ronald Cramer, Ivan Damgård, Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. CRYPTO '94, LNCS 839, pp. 174– 187. Springer-Verlag, 1994.

[6] Benjamin E. Diamond. "Many-out-of-Many" Proofs with Applications to Anonymous Zether. Cryptology ePrint Archive, Report 2020/293, 2020. https://eprint.iacr.org/2020/293.

[7] Reza R. Farashahi, Pierre-Alain Fouque, Igor E. Shparlinski, Mehdi Tibouchi, and J. Felipe Voloch. Indifferentiable Deterministic Hashing to Elliptic and Hyperelliptic Curves. Cryptology ePrint Archive, Report 2010/539, 2010. https://eprint.iacr.org/2010/539.pdf.

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

[9] 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, pages 253–280. Springer, 2015.

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

[11] Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. Zcash protocol specification. Tech. rep. 2016–1.10. Zerocoin Electric Coin Company, Tech. Rep., 2016.

[12] Russell W. F. Lai, Viktoria Ronge, Tim Ruffing, Dominique Schröder, Sri Aravinda Krishnan Thyagarajan, and Jiafan Wang. Omniring: Scaling private payments without trusted setup. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, pages 31–48, 2019.

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

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

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

[16] C.P. Schnorr. Efficient Signature Generation by Smart Cards. J. Cryptology (1991) 4: 161-174

[17] R. Rivest, A. Shamir, and Y. Tauman. How to leak a secret. Asiacrypt 2001, LNCS 2248, pp. 552–565. Springer-Verlag, 2001.

[18] Tsz Hon Yuen, Shi feng Sun, Joseph K. Liu, Man Ho Au, Muhammed F. Esgin, Qingzhao Zhang, and Dawu Gu. RingCT 3.0 for blockchain confidential transaction: Shorter size and stronger security. Cryptology ePrint Archive, Report 2019/508, 2019. https://eprint.iacr.org/2019/508.