Lin2-Xor Lemma and Log-size Linkable Ring Signature

Anton A. Sokolov

acmxddk@gmail.com

Abstract

In this paper we introduce a novel approach to constructing efficient linkable ring signatures without 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 signers threshold number. 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 Bunz, et. al. In contrast, we construct our signature from scratch using Lin2-Xor lemma that we formulate and prove here. Next, we generalize it to an *n*-move public coin special honest verifier zero-knowledge Lin2-Selector protocol and, consequently, instantiate the protocol in a form of linkable ring signature in the standard model.

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

19th May, 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.

The formal notion of ring signatures and the early yet efficient schemes are presented in the works of Rivest, Shamir, and Tauman [12], Abe, Ohkubo, and Suzuki [4], Liu, Wei, and Wong [5], an example of a system that uses linkable ring signatures is, for instance, CryptoNote [8]. 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 [4, 5, 8] and other linkable ring signature schemes can be instantiated with a prime-order cyclic group where discrete logarithm problem (DL) is hard. Scheme security and the signer anonymity are usually, e.g., as in [5], reduced to one of the common computational hardness assumptions, for instance, to the decisional Diffie-Hellman assumption (DDH) in the random oracle model or in the standard model.

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 [6], Sarang Noether [7], Benjamin E. Diamond [9], Russell WF Lai, Viktoria Ronge, Tim Ruffing, Dominique Schroder, Sri Aravinda Krishnan Thyagarajan, and Jiafan Wang [15], William Black and Ryan Henry [16], 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 [13], Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox [14] and some others it is shown that signer-ambigous signatures with asymptotically lower sizes and verification complexities can be built at the cost of requiring a trusted setup and 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 log-size linkable ring signature scheme over a prime-order cyclic group without bilinear-pairings under the DDH assumption in the standard model.

1.1. Contribution

- 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 as a disjunction unit, we prove Lin2-Selector lemma and construct an n-move public coin L2S identification protocol that allows for committing to a selected pair of elements from an arbitrary set of element pairs (anonymity set for this case), without revealing the selected pair itself. The unknown discrete logarithm relationship between the elements of the anonymity set is required.
- We prove the L2S id protocol is complete and sound under the DL, special honest verifier zero-knowledge under the DDH.
- Using the L2S id protocol we construct a non-interactive zero-knowledge mL2SHPoM proof of membership scheme and, consequently, construct a many-out-of-many mL2SLnkSig logarithmic-size linkable ring signature, that is signer-ambiguous under the DDH in the standard model. The signature verification complexity is linear.
- Compared to the setup-free log-size linkable ring signature schemes proposed in [6, 7, 9, 15], that originate from the ideas of Jens Groth and Markulf Kohlweiss [1], Benedikt Bunz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell [2], our scheme is constructed on a basis different from [1, 2]. A parallel can be drawn to the work of Jens Groth and Markulf Kohlweiss [1]: our Lin2-Xor and Lin2-Selector lemmas play a role similar to the role of the Kronecker's delta in [1]. The difference is in the anonymity set: in [1] it lays in a plain built over the homomorphic commitment generators, whereas for the Lin2-Xor and Lin2-Selector it is a set of orthogonal generators.

- We present our mL2SLnkSig signature scheme as a straightforward log-size solution for the linkable ring signature problem, when the anonymity set is easier to represent as a set of orthogonal generators, for instance, when linking tag contains a hash to a group element.

1.2. Method overview

In a nutshell, firstly we consider a linear combination 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 or 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 independent 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 in the chapter 5, such that: $wR=Z+\sum_{i=1\ldots n} r_i H_i$, where $Z=k_0P_s+k_1Q_s$ for some $s\in[1,2^{n-1}]$ and for some known a, b. The actual s is made indistinguishable by keeping the scalars k_0 and k_1 in secret. This is the Lin2-Selector lemma protocol.

We construct the L2S id protocol on top of the Lin2-Selector lemma protocol and prove that the L2S id protocol is complete and sound. Then, we prove the L2S id protocol is sHVZK following method by R.Cramer et. al. [10].

The protocol is efficient, it requires to transmit one Z and n (r_i, H_i) pairs, and to compute one multi-exponentiation for 2^n summands for R during verification.

Using the Fiat-Shamir heuristic, we turn the L2S id protocol to the mL2SHPoM non-intaractive many-out-of-many proof of membership and to the mL2SLnkSig ring signature. R is calculated only once for all signers in the proof of membership and in the signature. We add a linking tag to the signature as $x^{-1}\boldsymbol{H}_{\boldsymbol{point}}(P)$.

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 the 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, …, except for letters N, L and M, 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 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 a Prover and a Verifier, unless otherwise stated.

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

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

A protocol can be translated to non-interactive proofs using Fiat-Shamir heuristic in the standard model. We start with proving the lemmas in the interactive setting, next they are turned into 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 $S=\{B_1, B_2, ..., B_n\}$, then the sequence B_1, B_2, ..., B_n 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 that a polynomial-time algorithm 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 $A{\sim}B$ and $A!{\sim}B$ never hold. It's not required for the statements $A{\sim}B$ and $A!{\sim}B$ to obey the law of excluded middle anywhere below, the only assumed law and implication are:

- (not ($A{\sim}B$ and $A!{\sim}B$)), meaning that it's not possible simultaneously to know and not to know x in the $A=xB$.
- (not $A!{\sim}B$) implies $A{\sim}B$, meaning that if solving $A=xB$ is shown not hard, then $A{\sim}B$. That is, showing to be not hard means a demonstration of a polynomial-time algorithm for finding x from A and B.

No implication from (not $A{\sim}B$) to $A!{\sim}B$ is used. In general, the statements $A{\sim}B$ and $A!{\sim}B$ can hold, not hold and be undetermined. Each of $A{\sim}B$ and $A!{\sim}B$ is determined, if it's obtained by a premise or by implication. If both are determined, then they can't hold simultaneously. If $A!{\sim}B$ is determined and doesn't hold, then $A{\sim}B$ is determined and holds.

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 the 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 the ones for $A{\sim}B$ and $A!{\sim}B$:

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

In general, for an element set S the statements $A=lin(S)$ and $A!=lin(S)$ can hold, not hold and to be undetermined. Each of $A=lin(S)$ and $A!=lin(S)$ is determined if it's obtained by a premise or by implication. If both are determined, they can't hold simultaneously. If $A!=lin(S)$ is determined and doesn't hold, then $A=lin(S)$ is determined and holds.

For any elements A and B, the statement $A=lin(B)$ is equivalent to $A{\sim}B$, and $A!=lin(B)$ is equivalent to $A!{\sim}B$.

Due to the constructive nature of proofs, quantified statements for scalars "for all x …" and "there exist y …" are to be read as "for any provided x …" and "provided or 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 the unknown discrete logarithm of each element in the set to the other elements in the set: for each element $B_i \in S$ holds: $B_i!=lin(S \setminus \{B_i\})$.

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 the Prover to convince the Verifier of facts using evidences.

For instance, an evidence of $A{\sim}B$ can be simply x, such that Verifier can check $A=x*B$, 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 called exited by error. For some protocols we define function Verif instead, that returns 0 or 1. If 0 is returned, it means that a protocol immediately exits by error. If 1 is returned, it means the protocol continues to its successful completion.

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 a uniform random choice of scalars over $\mathbb{F}$ everywhere below and call it simply 'random choice'.

2.2.7. Negligible probability and contradictions

We assume probability to be negligible if its inverse is of the 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) => ($c=c'$, where c and c' are chosen uniformly and independently at random) => Contradiction.

2.2.8. Decoy sets and their cardinality

We call the anonymity set as a decoy set. One entry of the decoy set belongs to the 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 entries in the decoy set, usually except for one that the actual signer signs with.

The cardinality of the 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 terms ‘ring’ and simply ‘set’ as synonyms to the ‘decoy set’, assuming the following technical 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 sums of elements multiplied by scalars, are interchangeable, they both mean a sum like: $a_1B_1+a_2B_2+...+a_nB_n$. 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, 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) The both hold: $A!\sim B$ and $A!\sim C$

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.

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.

c) Suppose $A\sim B$, then by definition of $A\sim B$ there is provided x, such that $A=xB$. That is, by defenition of $lin()$, $A=lin(B, C)$ => Contradiction. The same for $A!\sim C$.

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 vector $\vec{x}=[x_i]_{i=1}^n$ of scalars, such that $A=\sum_{i=1...n} x_i B_i$, is unique.

Proof: Suppose $\vec{x}$ is not unique, that is, A has one more representation $\vec{y}$, then subtracting both representations we get $0=\sum_{i=1...n} z_i B_i$, where $\vec{z}=\vec{x}-\vec{y}$ has at least one non-zero scalar. Suppose z_j is non-zero, then moving z_jB_j to the left part and dividing by z_j we get $B_j=\sum_{i=1...n, i\neq j} (z_i/z_j) B_i$. This means that $B_j=lin(\vec{B} \backslash \{B_j\})$, however $B_j!=lin(\vec{B} \backslash \{B_j\})$ 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: $ort(S) \Rightarrow ort(\{(aB_j+bB_k)\} \cup (S \backslash (\{B_j\} \cup \{B_k\})))$.

Proof: Suppose the opposite, that means $(aB_j+bB_k)=lin(S\backslash(\{B_j\}\cup\{B_k\}))$ ⇒ moving B_k to the right: $aB_j=lin(S\backslash\{B_j\})$ ⇒ dividing by a: $B_j=lin(S\backslash\{B_j\})$ ⇒ 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 $0=\sum_{i=1\dots n} x_i B_i$, then $\vec{x}=\vec{0}$.

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

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 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 $(P+cQ){\sim}(Z+rH)$) ⇒ (Prover knows a, b, x, y, such that: $Z=aP+bQ$ and $H=xP+yQ$).

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

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 $(P+c'Q)\sim(Z+r'H)$.

Eliminating Z: $(P+cQ)/t-rH=(P+c'Q)/t'-r'H \Rightarrow (1/t-1/t')P+(c/t-c'/t')Q+(r'-r)H=0$.
Suppose $(r'-r)=0$. We have two possibilities with this assumption: $(1/t-1/t')=(c/t-c'/t')=0$ or $(1/t-1/t')P+(c/t-c'/t')Q=0$.

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

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

Finding H from the equation with the eliminated Z: $H=(1/t-1/t')/(r'-r)P+(c/t-c'/t')/(r'-r)Q \Rightarrow$ $H=lin(P, Q)$. By the OrtUniqueRepresentation lemma: $x=(1/t-1/t')/(r'-r)$ and $y=(c/t-c'/t')/(r'-r)$.

$Z=(P+cQ)/t-rH=(1/t)P+(c/t)Q-r(1/t-1/t')/(r'-r)P-r(c/t-c'/t')/(r'-r)Q \Rightarrow Z=lin(P, Q)$.

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 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, $ort(P_1, Q_1\ P_2, Q_2) \Rightarrow ort((P_1+c_{11}Q_1), (c_{12}P_2+c_{13}Q_2)) \Rightarrow$ by definition of $ort()$, $(P_1+c_{11}Q_1)!\sim(c_{12}P_2+c_{13}Q_2)$.

With this, we get exactly the premise, protocol and conclusion of the Lin2 lemma with the following substitution:

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:

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

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) $(Z+r_1H_1)\sim(P_1+c_{11}Q_1)$

b) $(Z+r_1H_1)\sim(c_{12}P_2+c_{13}Q_2)$

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 '~': $(Z+r_1H_1)\sim(P_1+c_{11}Q_1)$.

If $(Z=lin(P_2, Q_2)$ and $H_1=lin(P_2, Q_2))$, then, likewise, $(Z+r_1H_1)\sim(c_{12}P_2+c_{13}Q_2)$.

5. Lin2-Selector lemma

5.1. Preliminary definitions and lemmas

5.1.1. Rsum

Let's rewrite the $R=P_1+c_{11}Q_1+c_2c_{12}P_2+c_2c_{13}Q_2$ sum, that we considered in the Lin2-Xor lemma, as the following tree structure (see Figure 1):

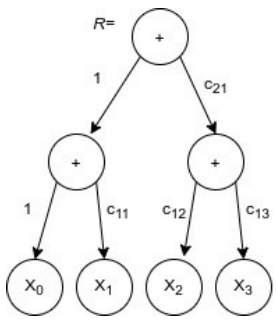

Figure 1. Rsum for four elements.

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 like as on Figure 2:

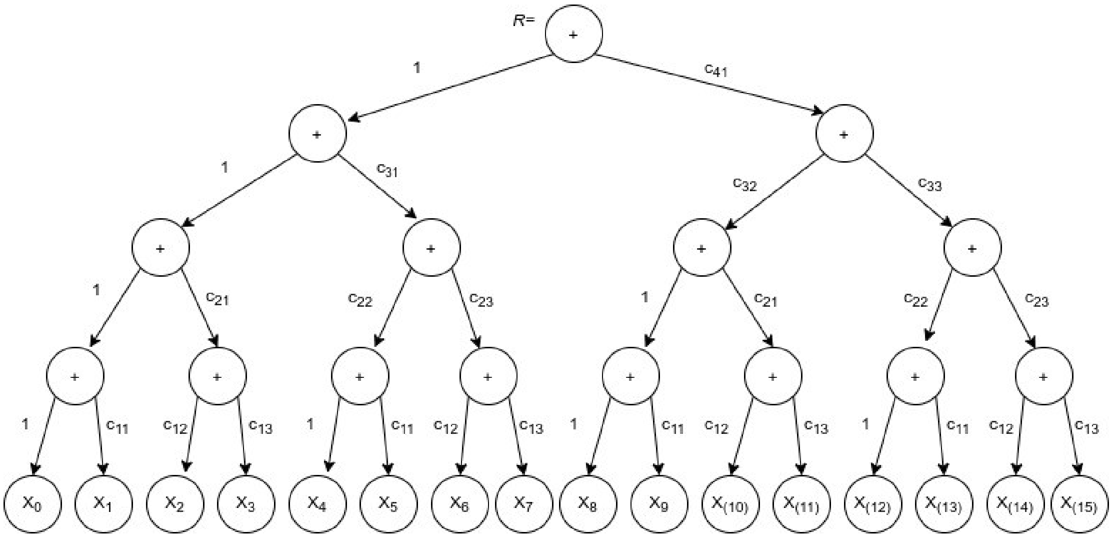

Figure 2. Rsum for sixteen elements.

This is the sum $R=$

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

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 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:

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

Informally, for $n>1$, 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 $c_{n0}\neq 0$, the following holds:

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

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

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 $N=2^n$, 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 is an evidence of $Z=lin(X_{(2s)}, X_{(2s+1)})$ for some $s \in [0,N/2\text{-}1]$:

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: $\mathrm{Rsum}(n, N, [X_j]_{j=0}^{N-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}, (1, c_n)) \sim (Z+\sum_{i=1\ldots n} r_i H_i)$	Verifier checks the evidence: $\mathrm{Rsum}(n, N, [X_j]_{j=0}^{N-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}, (1, c_n)) \sim (Z+\sum_{i=1\ldots n} r_i H_i)$

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 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$(m+1, 2M, [X_j]_{j=0}^{2M-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{m}, (1, c_{m+1}))\sim(Z+\sum_{i=1...(m+1)} r_i H_i)$	Verifier checks the evidence: Rsum$(m+1, 2M, [X_j]_{j=0}^{2M-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{m}, (1, c_{m+1}))\sim(Z+\sum_{i=1...(m+1)} r_i H_i)$

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

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

$$\text{Rsum}(m+1, 2M, [X_j]_{j=0}^{2M-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{m}, (1, c_{m+1}))=$$
$$\text{Rsum}(m, M, [X_j]_{j=0}^{M-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{m-1}, (1, c_{m1}))+$$
$$c_{m+1}\text{Rsum}(m, M, [X_j]_{j=M}^{2M-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{m-1}, (c_{m2}, c_{m3}))=$$
$$\text{Rsum}(m\text{-}1, M/2, [X_j]_{j=0}^{M/2-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{m-2}, (1, c_{(m-1),1}))+$$
$$c_{m1}\text{Rsum}(m\text{-}1, M/2, [X_j]_{j=M/2}^{M-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{m-2}, (c_{(m-1),2}, c_{(m-1),3}))+$$
$$c_{m+1}c_{m2}\text{Rsum}(m\text{-}1, M/2, [X_j]_{j=M}^{3M/2-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{m-2}, (1, c_{(m-1),1}))+$$
$$c_{m+1}c_{m3}\text{Rsum}(m\text{-}1, M/2, [X_j]_{j=3M/2}^{2M-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{m-2}, (c_{(m-1),2}, c_{(m-1),3}))=$$
$$Y_0+c_{m1}Y_1+c_{m+1}c_{m2}Y_2+c_{m+1}c_{m3}Y_3, \text{ where:}$$

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

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

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 an element F as: $F= Z+\sum_{i=1...(m-1)} r_i H_i$, the evidence is

$$Y_0+c_{m1}Y_1+c_{m+1}c_{m2}Y_2+c_{m+1}c_{m3}Y_3 \sim (F+r_mH_m+r_{m+1}H_{m+1})$$

Now, let's look at the step where Verifier picks 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 $(Y_0+c_{m1}Y_1+c_{m+1}c_{m2}Y_2+c_{m+1}c_{m3}Y_3) \sim (F+r_mH_m+r_{m+1}H_{m+1})$ 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 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 $(Y_0+c_{m1}Y_1+c_{m+1}c_{m2}Y_2+c_{m+1}c_{m3}Y_3) \sim (F+r_mH_m+r_{m+1}H_{m+1})$.

Hence, by the Corollary of Lin2-Xor lemma, if the Verifier successfully completes the protocol for $n=(m+1)$, that is, if Verifier successfully checks that

$$\text{Rsum}(m+1, 2M, [X_j]_{j=0}^{2M-1}, [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{m}, \quad (1, c_{m+1})) \sim (Z+\sum_{i=1...(m+1)} r_i H_i),$$

then, by this, he successfully checks that

$$Y_0+c_{m1}Y_1+c_{m+1}c_{m2}Y_2+c_{m+1}c_{m3}Y_3 \sim (F+r_mH_m+r_{m+1}H_{m+1}),$$

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

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

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

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

$$Y_0+c_{m1}Y_1=\mathrm{Rsum}(m\text{-}1,\ M/2,\ [X_j]_{j=0}^{M/2-1},\ [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{m-2},\ (1, c_{(m-1),1}))+$$
$$c_{m1}\mathrm{Rsum}(m\text{-}1,\ M/2,\ [X_j]_{j=M/2}^{M-1},\ [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{m-2},\ (c_{(m-1),2}, c_{(m-1),3}))=$$
$$\mathrm{Rsum}(m,\ M,\ [X_j]_{j=0}^{M-1},\ [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{m-1},\ (1, c_{m1}))$$

$$c_{m2}Y_2+c_{m3}Y_3=c_{m2}\mathrm{Rsum}(m\text{-}1,\ M/2,\ [X_j]_{j=M}^{3M/2-1},\ [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{m-2},\ (1, c_{(m-1),1}))+$$
$$c_{m3}\mathrm{Rsum}(m\text{-}1,\ M/2,\ [X_j]_{j=3M/2}^{2M-1},\ [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{m-2},\ (c_{(m-1),2}, c_{(m-1),3}))=$$
$$\mathrm{Rsum}(m,\ M,\ [X_j]_{j=M}^{2M-1},\ [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{m-1},\ (c_{m2}, c_{m3}))=$$
$$c_{m2}\mathrm{Rsum}(m,\ M,\ [X_j]_{j=M}^{2M-1},\ [(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{m-1},\ (1, c_{m3}/c_{m2}))$$

Thus, using the definition of F, the two above equalities and inserting r_mH_m in the sum, exactly one of the following a) or b) holds:

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

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

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

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

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

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

As both c_{m3} and c_{m2} are picked uniformly at random, $c_m=(c_{m3}/c_{m2})$ is also uniformly random. 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:

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

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 successfully met, then Verifier has exactly one of the two evidences: ($Z=lin(X_{(2s)}, X_{(2s+1)})$ for some $s\in[0,M/2\text{-}1])$
or $(Z=lin(X_{(2s)}, X_{(2s+1)})$ for some $s\in[M/2,M\text{-}1])$.

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

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

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 the sum: $Z+r_1H_1+r_2H_2$, 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 the $Z+r_1H_1$. So, according 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 let the equality holds.

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 carefully 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 the sum: $Z+r_1H_1+...+r_nH_n$, 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, the 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 the 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: $(Z+r_1H_1+...+r_{n-2}H_{n-2}) \rightarrow Z$, $r_{n-1} \rightarrow r_1$, $r_n \rightarrow r_2$, $H_{n-1} \rightarrow H_1$, $H_n \rightarrow H_2$, find that the new Z and H_1 are fixed and 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 halfs 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 $Z+r_1H_1+...+r_{n-1}H_{n-1}$.

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 the induction we prove this a bit differently, in 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 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 identification protocol

We construct an identification protocol called **L2S**, where Verifier is provided with an element Z, and, upon successful completion of all steps of the protocol, Verifier is convinced in Z is a commitment built upon a publicly known set of element pairs, such that Prover knows an opening for Z.

We prove the **L2S** protocol is complete, sound and special honest verifier zero-knowledge.

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 $(k_0X_{2s}+k_1X_{2s+1})$. That is,

$$\text{Com2}(k_0, k_1, s, \vec{X}) = k_0X_{2s}+k_1X_{2s+1}$$

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 the opening, it's possible to efficiently calculate the index s by iterating through $\vec{X}$ and checking if $Z=k_0X_{2s}+k_1X_{2s+1}$.

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

6.2. L2S id protocol

We define **L2S** identification protocol as four procedures **L2S**={**DecoySetGen**, **KeyGen**, **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. The distribution of elements in $\vec{X}$ is to be indistinguishable from the independent random uniformity.
  For any **DecoySetGen** implementation choice, the returned vector $\vec{X}$ orthogonality and independent random uniformity are to be guaranteed.

- **KeyGen**($\vec{X}$) is an arbitrary function, that returns a private-public key-pair $((k_0, k_1, s), Z)$, where (k_0, k_1, s) is the private-key with $k_0 \neq 0$ having an uniform random distribution, with arbitrary k_1, s, and Z is the public-key such that: $Z=$Com2$(k_0, k_1, s, \vec{X})$.
  As $k_0 \neq 0$ holds with overwhelming probability, this doesn't affect the uniformity of k_0.
  For any **KeyGen** implementation choice, the random uniformity of k_0 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 a private-key (k_0, k_1, s), $k_0 \neq 0$, and Verifier having an element 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 $[X_j]_{j=0}^{N-1}$ =**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 $H_1=wY_h/q$	H_1	
		$(c_{i1}, c_{i2}, c_{i3}) \leftarrow$ random, non-zero
$c_{i0}=1$ $r_i=q((c_{i,(h\%4)}/c_{i,(z\%4)})-k)$	r_i	
$k=0$		
$M=M/2$ $[Y_j]_{j=0}^{M-1}=[(c_{i,(2j\%4)}Y_{(2j)}+c_{i,((2j+1)\%4)}Y_{(2j+1)})/c_{i,(z\%4)}]_{j=0}^{M-1}$		
$a=c_{i,(z\%4)}a$		
$(z, h)=(z//2, InvertLastBit(z//2))$		
$q \leftarrow$ random, non-zero $H_{i+1}=wY_h/q$	H_{i+1}	
		$i=i+1$ If ($i<n$), then: Otherwise:
		$c_n \leftarrow$ random, non-zero
$(c_{n0}, c_{n1})=(1, c_n)$ $r_n=q(c_{n,h}/c_{n,z})$	r_n	
$a=c_{n,z}a$		
$x=a/w$		
$q \leftarrow$ random, non-zero $W=(k_0X_{2s}+k_1X_{2s+1})+\sum_{i=1\ldots n} r_i H_i$		
$T=qW$	T	
		$c \leftarrow$ random, non-zero
$t=q-xc$	t	
		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 the 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 $N=2^n$
$R=\text{Rsum}(n, N, [X_j]_{j=0}^{N-1}, [(1, c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}, (1, c_n))$
$W=Z+\sum_{i=1\ldots n} r_i H_i$
If $(tW+cR)==T$ then return 1
Else return 0.

Table 6. **L2S.Verif** function.

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

- A decoy set is generated using same implementation of the **L2S.DecoySetGen** at both Prover and Verifier sides.
- Prover obtains a private-key (k_0, k_1, s) from the **L2S.KeyGen**. At the same time, Verifier obtains 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.

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:

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

$\text{Rsum}(3, 8, [X_t]_{t=8j}^{16j-1}, [(c_{d,0}, c_{d,1}, c_{d,2}, c_{d,3})]_{d=1}^{2}, (c_{3,(2j\%4)}, c_{3,((2j+1)\%4)}))$ 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, $\text{Rsum}(n, N, [X_j]_{j=0}^{N-1}, [(1, c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}, (1, c_n))=xW$.

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 [17] initial message, challenge and reply for the relation $R=xW$.

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 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 id protocol properties

6.3.1. Completeness

As proven in 6.2.2, if Z on Verifier's input is equal to Com2$(k_0, k_1, s, [X_j]_{j=0}^{N-1})$, where (k_0, k_1, s) is the Prover's input, then the **L2S.Verif** returns 1. Thus, if a public-key corresponds to its private-key, then the **L2S.Verif** returns 1.

That means the **LS2** id protocol is complete.

6.3.2. Soundness

The **L2S.InteractionProcedure** with the subsequent call to the **L2S.Verif** meet 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 $s\in[0,N/2-1]$, that is, by the definitions of $lin()$ and Com2, Verifier is convinced that Prover knows k_0, k_1, s, such that Z=Com2$(k_0, k_1, s, [X_j]_{j=0}^{N-1})$.

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

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

The Prover-Verifier public transcript is exactly the tuple $([(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}, (1, c_n), Z, [(r_i, H_i)]_{i=1}^{n}, c, T, t)$.

The T, t in it are related to the Schnorr id scheme, they are distributed uniformly at random.

All the challenges are random uniform. All r_i's are random uniform too, as each r_i is obfuscated by the private multiplier q, that 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. All these elements are obfuscated by the multiplier w.

The multiplier w is private random uniform, as $w=k_0$, where k_0 is random uniform by definition of the **L2S.KeyGen**.

By the definition of Rsum, each r_iH_i is a linear combination of $[X_j]_{j=0}^{N-1}$ with known, or, at least, efficiently computable, scalar coefficients. Moreover, all r_iH_i's in a proof depend on different subsets of the $[X_j]_{j=0}^{N-1}$.

Using the terms introduced in [18], the r_iH_i's are linearly independent degree 2 polynomials of a private set of the independent and random uniform variables: the discrete logarithms of $[X_j]_{j=0}^{N-1}$, w, all private random q's. Thus, reducing the question of the r_iH_i's distribution to the (P,Q)-DDH problem [18], we have:

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

By the (P,Q)-DDH assumption, the distributions of all the r_iH_i's are independent random uniform, indistinguishable from e_iG's, where all e_i's are independent random uniform. In addition to this, the distributions of the r_iH_i's are independent from the distributions of Z and T.

As the DDH assumption implies the (P,Q)-DDH [18] for our polynomials in P and Q, we have all the r_iH_i's are distributed independently and uniformly at random under the DDH.

6.3.4. Special Honest Verifier Zero-knowledge

Let's build a simulator that, given an arbitrary Z', generates a statistically indistinguishable from an honest Prover-Verifier one simulated transcript that the honest Verifier accepts.

This means that the simulator, knowing all the Verifier's challenges beforehand, has to generate the simulated transcripts for different Z''s, such that the **L2S.Verif** always returns 1 on these transcripts. The Verifier is honest, i.e., it acts completely in accordance with the **L2S** protocol.

Let's define the simulator. For any given Z' it proceeds as follows:

- Picks a random uniform k_0 for some fixed s and sets Z^{HT}=Com2$(k_0, 0, s)$
- Runs the honest Prover-Verifier **L2S** protocol and obtains its transcript:
  HT=$([(c_{i1}^{HT}, c_{i2}^{HT}, c_{i3}^{HT})]_{i=1}^{n-1}, (1, c_n^{HT}), Z^{HT}, [(r_i^{HT}, H_i^{HT})]_{i=1}^{n}, c^{HT}, T^{HT}, t^{HT})$
  As the protocol is complete, the **L2S.Verif**(HT) returns 1.
- Substitutes $Z'\rightarrow Z^{HT}$ and $(H_1^{HT}+(Z^{HT}-Z')/r_1^{HT})\rightarrow H_1^{HT}$ in the HT:
  ST=$([(c_{i1}^{HT}, c_{i2}^{HT}, c_{i3}^{HT})]_{i=1}^{n-1}, (1, c_n^{HT}), Z', [(r_1^{HT}, H_1^{HT}+(Z^{HT}-Z')/r_1^{HT})\cup(r_i^{HT}, H_i^{HT})]_{i=2}^{n}, c^{HT}, T^{HT}, t^{HT})$

All the challenges, t^{HT}, T^{HT} and the sum $Z^{HT}+\sum_{i=1...n} r_i^{HT} H_i^{HT}$ remain intact after this substitution, so the **L2S.Verif**(ST) returns 1.

As shown in 6.3.3., all elements and scalars in an honest Prover-Verifier conversation transcript with a random uniform input Z are indistinguishable from the independent random uniform.

All elements and scalars in the ST's simulated with the random uniform input Z' are indistinguishable from the independent random uniform, too. The only two elements in them that differ in their calculation from the honest Prover-Verifier case are Z' and $(r_1^{HT}H_1^{HT}+Z^{HT}-Z')$.

Formally, using the (P,Q)-DDH method applied in 6.3.3., it's possible to prove, that the distributions of Z' and $(r_1^{HT}H_1^{HT}+Z^{HT}-Z')$ are indistinguishable from the independent random uniform. However, it's easier to see this from that Z^{HT} is a private independent random uniform element generated inside the simulator. Hence, as the distribution of $r_1^{HT}H_1^{HT}$ is independent of the distribution of Z^{HT} according to 6.3.3., and is independent of the distribution of Z', the distribution of $(r_1^{HT}H_1^{HT}+Z^{HT}-Z')$ is independent of the distribution of Z'. At the same time, the distribution of $(r_1^{HT}H_1^{HT}+Z^{HT}-Z')$ is independent of the distributions of all other elements in the ST, as it contains Z'.

Thus, all elements in an ST have distributions indistinguishable from independent random uniform.

Suppose, there exists a PPT adversary that statistically distinguish the ST's generated by the simulator for some uniform random input stream of Z''s from the honest Prover-Verifier conversation transcripts generated for another uniform random input stream of Z's. This implies that the adversary is able to distinguish two streams of tuples, each containing only independent random uniform elements. This contradicts to the uniform randomness.

Thus, the simulator generates simulated transcripts indistinguishable from the honest ones.

Having provided this simulator, we have proven the **L2S** id protocol is sHVZK under the DDH.

7. L2S id protocol extensions

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

As shown in 6.3.4., the **L2S** is sHVZK under the DDH, as long as the input public-keys Z distribution is indistinguishable from the independent random uniform.

To remove this restriction and to allow the protocol to keep the sHVZK property for any input public-keys, including the cases when a linear relationship of different public-keys is publicly known or known to particular adversaries, we extend the **L2S** protocol with an input randomization. This allows to keep the sHVZK for any input public-keys.

The idea of input randomization is: right in the beginning of the **L2S.InteractionProcedure** Prover multiplies the private-public key-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 private-public key-pair:
$((k_0, k_1, s), Z) \leftarrow ((fk_0, fk_1, s), fZ)$.

We define **iL2S** id protocol as four procedures:
iL2S={**DecoySetGen=L2S.DecoySetGen**, **KeyGen=L2S.KeyGen**, **InteractionProcedure**, **Verif**}, where:

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

Prover and Verifier common parameters:	
• n, $N=2^n$, • a set of elements $[X_j]_{j=0}^{N-1}$ =**DecoySetGen**(n)	
Prover: (k_0, k_1, s), where: $k_0 \neq 0$ s – secret index, $s \in [0,N/2-1]$	Verifier: Z
Z_0=Com2$(k_0, k_1, s, [X_j]_{j=0}^{N-1})$	$Z_0=Z$
$f \leftarrow$ random, non-zero	
$Z=fZ_0$ $(k_0, k_1, s)=(fk_0, fk_1, s)$ Z ⇨	Verifier assigns the received value to Z
$q \leftarrow$ random, non-zero $T_0=qZ_0$ T_0 ⇨	
⇦	$c_0 \leftarrow$ random, non-zero
$t_0=q-fc_0$ t_0 ⇨	
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), $([(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}, (1, c_n), Z, [(r_i, H_i)]_{i=1}^{n}, c, T, t)$

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 $(t_0Z_0+c_0Z)==T_0$ then continue   Else return 0
Run **L2S.Verif** for the 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** id protocol are the same as for the **L2S** id protocol.

7.1.1. iL2S id protocol completeness, soundness and sHVZK

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

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

- As the (Z_0, c_0, T_0, t_0) tuple is put to the beginning of the public Prover-Verifier transcripts, it's necessary to determine the distributions of T_0 and t_0. The scalar t_0 is random uniform, as it's obfuscated by the private random uniform q. The distribution of the element T_0 is independent indistinguishable from the random uniform. These are the properties of the Schnorr id scheme.
- The simulator starts with simulating the (Z_0, c_0, T_0, t_0) part of the transcript by multiplying the input by some private factor f, so that the first check in the **iL2S.Verif** passes. Then, the simulator proceeds the same way as for the **L2S**.

7.1.2. iL2S id protocol public transcript entries independence

A nice property acquired after the input randomization is that all elements in the public Prover-Verifier transcript appear to be multiplied by the private random uniform scalar f. As for the single element T_0 not multiplied by f, it is independent random uniform, as it is multiplied by its own private random uniform q.

Therefore, if we put in a row a number of arbitrary public Prover-Verifier transcripts, even generated for linear dependent public-keys, then we find all their elements to be independent of each other and indistinguishable from the random uniformity. The same holds for their scalars.

7.2. mL2S id protocol

A natural extension to the **iL2S** id protocol is a protocol, that runs multiple instances of the **iL2S.InteractionProcedure** in parallel and thus ensures identities for multiple public-keys at once. We call this extension **mL2S** id protocol.

mL2S id protocol is four procedures:
mL2S={**DecoySetGen=L2S.DecoySetGen**, **KeyGen=L2S.KeyGen**,
MapInteractionProcedure, **JoinVerif**},
where:

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

Prover and Verifier common parameters:

- L
- n, $N=2^n$

Prover:	Verifier:
$[(k_0^p, k_1^p, s^p) \mid k_0^p \neq 0]_{p=1}^{L}$	$[Z^p]_{p=1}^{L}$

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.

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:

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

Table 9. **mL2S.MapInteractionProcedure**.

- **mL2S.JoinVerif** function is shown in the 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 $N=2^n$, $((Z_0^p, c_0, T_0^p, t_0^p), ([(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}, (1, c_n), Z^p, [(r_i^p, H_i^p)]_{i=1}^{n}, c, T^p, t^p))_{p=1}^{L}$ $R=\mathrm{Rsum}(n, N, [X_j]_{j=0}^{N-1}, [(1, c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}, (1, c_n))$ For each $p\in[1,L]$: run **iL2S.Verif** using n, $[X_j]_{j=0}^{N-1}$ and $(Z_0^p, c_0, T_0^p, t_0^p), ([(c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}, (1, c_n), Z^p, [(r_i^p, H_i^p)]_{i=1}^{n}, c, T^p, t^p)$ as arguments. Inside each **iL2S.Verif** call, within nested **L2S.Verif** call, use the calculated above R for the **iL2S.Verif.L2S.Verif**.R Return 0 if one of the **iL2S.Verif** calls returns 0. Otherwise, return 1.

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 $R=\mathrm{Rsum}(n, N, [X_j]_{j=0}^{N-1}, [(1, c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}, (1, c_n))$ for them.

The steps for the **mL2S** id protocol are similar to the steps of the **iL2S** id protocol, with the only difference that the parallel procedure versions are used in place of the sequential ones:
MapInteractionProcedure → **InteractionProcedure**,
JoinVerif → **Verif**

7.2.1. mL2S id protocol completeness, soundness and sHVZK

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

The **mL2S** id protocol is sHVZK, as all the **iL2S** id protocol instances run in parallel are sHVZK and, due to the public transcript entries independence property established in 7.1.2., all entries of the unified public transcript, except for the L input Z_0's, are independent indistinguishable from the random uniform.

That is, a simulator for the **mL2S** id protocol runs L **iL2S** id protocol simulators in parallel, and, after completion, the simulated transcript contains L indistinguishable from honest **iL2S** simulated transcripts, that have no correlation between each other due to the 7.1.2. Thus, the **mL2S** simulated transcript is indistinguishable from an honest **mL2S** transcript.

7.2.2. mL2S id protocol complexities

Keeping in mind 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 is transferred, the amount of data transferred from Prover to Verifier is shown in Table 11.

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

Table 11. **mL2S** transferred data amount.

The R=Rsum$(n, N, [X_j]_{j=0}^{N-1}, [(1, c_{i1}, c_{i2}, c_{i3})]_{i=1}^{n-1}, (1, c_n))$ 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 element 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 signature

Having an interactive honest verifier zero-knowledge interactive id protocol, it's possible to creating the non-interactive proof of membership and signature schemes on its base using the Fiat-Shamir heuristic [19].

We create a non-interactive zero-knowledge proof of membership and a linkable signature schemes on the base of the **mL2S**.

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 $\boldsymbol{H_{point}}(...)$ defined below.

8.1. Preliminaries

Elliptic curve points and elements, point definition:

We assume a 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 $\mathbb{G}$ everywhere below. Thus, we use the term 'points' instead of the 'elements', they are equivalent.

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

We call $\boldsymbol{H_{scalar}}(...)$ 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 $\boldsymbol{H}_{point}(...)$ an ideal hash function that accepts any number of points in $\mathbb{G}$ and returns a point in $\mathbb{G}$.

Integers *n, N, L*:

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

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

Decoy vector as a vector of pairs:

The procedure **mL2S.DecoySetGen** in 7.2. returns a decoy vector $[X_j]_{j=0}^{N-1}$. 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 $[(P_j, Q_j)]_{j=0}^{N/2-1}$,
and 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**={**ProcedureA, ProcedureB**}, where the **ProcedureB** is defined as taking X and returning $\boldsymbol{H}_{point}(X)$, then, if we use the **Scheme** within another scheme and want the **ProcedureB** to be returning $(X+\boldsymbol{H}_{point}(X))$ after some point, we write: **ProcedureB**=$\lambda(X).(X+\boldsymbol{H}_{point}(X))$.

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:

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

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

Note, the s^p's are not required to be different, that is, only membership is going to be proved. The different A^p's are allowed to have a known linear relationship between each other.

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 transferred from Prover to Verifier for each of L parallel identifications. 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 proof 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 five 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 **mHashPoint** is that any its implementation is an ideal point-to-point hash function.

```
Input: B
Output: A point-hash of B
Procedure:
    Return H_point(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 $[P_j]_{j=0}^{N/2-1}$, **HashPoint** mapped to the pre-images
Procedure:
$[B_j]_{j=0}^{N/2-1}$=**PreimageSetGen**()
$[P_j]_{j=0}^{N/2-1}$=[**HashPoint**$(B_j)]_{j=0}^{N/2-1}$
Return $[P_j]_{j=0}^{N/2-1}$

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 is excluded. Implementation is in the Listing 3.

Input: *none*
Output: decoy set $[X_j]_{j=0}^{N-1}$
Procedure:
$[A^p]_{p=1}^{L}$=**MemberSetGen**()
$Qshift=\boldsymbol{H}_{point}([A^p]_{p=1}^{L})$
$[P_j]_{j=0}^{N/2-1}$=**GetImageSet**()
$[Q_j]_{j=0}^{N/2-1}=[\boldsymbol{H}_{point}(Qshift+P_j)]_{j=0}^{N/2-1}$

```
    [X_j]_{j=0}^{N-1} = Flatten([(P_j, Q_j)]_{j=0}^{N/2-1})
    Return [X_j]_{j=0}^{N-1}
```

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:   [(v^p, s^p)]_{p=1}^{L}   --private keys
         e                        --scalar seed
Output:  [σ^p]_{p=1}^{L} or 0     --proof, vector of σ's on success,
                                  --      0 on failure

Procedure:
  •  Let [X_j]_{j=0}^{N-1} =GetDecoySet()
  •  Let [A^p]_{p=1}^{L} =MemberSetGen()
  •  Ensure the private keys correspond to the member set elements:
     For p=1...L:
          If A^p ≠ v^p X_{2s^p} then Return 0
  •  Let [(k_0^p, k_1^p, s^p)]_{p=1}^{L} =[(v^p, 0, s^p)]_{p=1}^{L}
  •  [Z_0^p]_{p=1}^{L} =[A^p]_{p=1}^{L}
  •  Run all L iL2S.InteractionProcedure's in parallel with the
     [(k_0^p, k_1^p, s^p)]_{p=1}^{L} and [Z_0^p]_{p=1}^{L} as arguments. Stop all them at
     the point, where the first challenge c_0 is to be obtained.
     At that moment the values [(Z_0^p, T_0^p, Z^p)]_{p=1}^{L} have already been
     calculated.
  •  Calculate e=H_scalar(e, [X_j]_{j=0}^{N-1}, [(Z_0^p, T_0^p, Z^p)]_{p=1}^{L})
  •  Let c_0=e
  •  Continue all the L parallel procedures to the point, where
     the challenge tuple (c_11, c_12, c_13) is to be obtained.
     At that moment the [t_0^p]_{p=1}^{L} and [H_1^p]_{p=1}^{L} have already been
     calculated.
  •  Calculate e=H_scalar(e, [t_0^p]_{p=1}^{L}, [H_1^p]_{p=1}^{L})
  •  Let (c_11, c_12, c_13)=(e, H_scalar(e), H_scalar(e+1))
  •  Continue all the L parallel procedures to the point, where
     the challenge tuple (c_21, c_22, c_23) is to be obtained.
     At that moment the [r_1^p]_{p=1}^{L} and [H_2^p]_{p=1}^{L} have already been
     calculated.
  •  Calculate e=H_scalar(e, [r_1^p]_{p=1}^{L}, [H_2^p]_{p=1}^{L})
  •  Let (c_21, c_22, c_23)=(e, H_scalar(e), H_scalar(e+1))
  •  And so on...,
     until all values [(Z_0^p, T_0^p, Z^p, t_0^p, [(r_i^p, H_i^p)]_{i=1}^{n}, T^p, t^p)]_{p=1}^{L} and
     (c_0, [(c_i1, c_i2, c_i3)]_{i=1}^{n-1}, c_n, c) are calculated.
  •  Let [σ^p]_{p=1}^{L} =[(Z_0^p, T_0^p, Z^p, t_0^p, [(r_i^p, H_i^p)]_{i=1}^{n}, T^p, t^p)]_{p=1}^{L}
  •  Return [σ^p]_{p=1}^{L}
```

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 a non-interactive setting. Specification is provided in Listing 5.

```
Input:  [σ^p]_{p=1}^{L}      --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 [X_j]_{j=0}^{N-1} =GetDecoySet()
  • Extract the values of [(Z_0^p, T_0^p, Z^p)]_{p=1}^{L} from the [σ^p]_{p=1}^{L}
  • Calculate e=H_scalar(e, [X_j]_{j=0}^{N-1}, [(Z_0^p, T_0^p, Z^p)]_{p=1}^{L})
  • Let c_0=e
  • Extract the values of [t_0^p]_{p=1}^{L} and [H_1^p]_{p=1}^{L} from the [σ^p]_{p=1}^{L}
  • Calculate e=H_scalar(e, [t_0^p]_{p=1}^{L}, [H_1^p]_{p=1}^{L})
  • Let (c_11, c_12, c_13)=(e, H_scalar(e), H_scalar(e+1))
  • Extract the values of [r_1^p]_{p=1}^{L} and [H_2^p]_{p=1}^{L} from the [σ^p]_{p=1}^{L}
  • Calculate e=H_scalar(e, [r_1^p]_{p=1}^{L}, [H_2^p]_{p=1}^{L})
  • Let (c_21, c_22, c_23)=(e, H_scalar(e), H_scalar(e+1))
  • And so on...,
    until all values (c_0, [(c_i1, c_i2, c_i3)]_{i=1}^{n-1}, c_n, c) are restored.
    At this moment all values of [(Z_0^p, T_0^p, Z^p, t_0^p, [(r_i^p, H_i^p)]_{i=1}^{n}, T^p, t^p)]_{p=1}^{L}
    are extracted from the [σ^p]_{p=1}^{L}.
  • For p=1...L:
      If (t_0^p Z_0^p + c_0 Z^p) ≠ T_0^p then Return 0
  • Calculate 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 p=1...L:
      Calculate W= Z^p + Σ_{i=1...n} r_i^p H_i^p
      If (t^p W + cR) ≠ T^p 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 a proof and, thus, is not able to cheat, is zero-knowledge.

8.2.4. mL2SHPoM complexities

The **mL2SHPoM** proof size, recalling the proof is $[\boldsymbol{\sigma}^p]_{p=1}^{L}$, is shown in Table 13. The scalar seed is not accounted, as it can have any 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 $\boldsymbol{H}_{scalar}$ calls are assumed taking negligible amount of the computational time.

	multi-exp(N)	single-exp	$\boldsymbol{H}_{point}$
mL2SHPoM	1	$nL+3L+1$	$N+1$

Table 14. **mL2S** verification complexity.

8.3. Linkable ring signature based on the mL2SHPoM

We construct a non-interactive linkable ring signature **mL2SLnkSig** on the base of **mL2SHPoM**.

The idea is following: 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 $\boldsymbol{H}_{point}(B)/b$, we have a set $[I^p \mid b^pI^p=\boldsymbol{H}_{point}(B_{s^p}), s^p\in[0,N/2-1], \forall\ i,j: s^i\neq s^j]_{p=1}^{L}$. 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 that $\forall\ I\in[I^p]_{p=1}^{L}\ \exists\ B\in[B_j]_{j=0}^{N/2-1}, B'\in[B_j]_{j=0}^{N/2-1}, b, b'$: $bI=\boldsymbol{H}_{point}(B)$ and, at the same time, $(\ b'(G+I)=(B'+\boldsymbol{H}_{point}(B'))\ ) \Rightarrow (\ b'(bG+bI)=(bB'+b\boldsymbol{H}_{point}(B'))\ ) \Rightarrow (\ b'(bG+\boldsymbol{H}_{point}(B))=(bB'+b\boldsymbol{H}_{point}(B'))\ ) \Rightarrow (\ b'\boldsymbol{H}_{point}(B)-b\boldsymbol{H}_{point}(B')=(bB'-bb'G)\ )$.

This equility, by definition of ideal hash function, can hold only if $B=B'$ and $b=b'$.

Hence, Verifier is convinced in that $\forall\ I\in[I^p]_{p=1}^{L}\ \exists\ B\in[B_j]_{j=0}^{N/2-1}, b$: $(\ bI=\boldsymbol{H}_{point}(B)$ and $b(G+I)=(B+\boldsymbol{H}_{point}(B))\ ) \Rightarrow (\ B=bG$ 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 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, it 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. That is, a key-image for B is B's linking tag.

8.3.1. mL2SLnkSig linkable signature

The **mL2SLnkSig** linkable signature scheme is four procedures:
mL2SLnkSig={**RingGen**, **Sign**, **Verif**, **Link**},
where:

- **mL2SLnkSig.RingGen** 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. The procedure contract is the same as for the **mL2SHPoM.PreimageSetGen**.
- **mL2SLnkSig.Sign** takes an actual signer vector of private-keys $[(b^p, s^p) \mid b^pG= B_{s^p}, s^p\in[0,N/2\text{-}1], \forall\ i,j: s^i\neq s^j\ ]_{p=1}^{L}$, a scalar message m and returns a signature $([\sigma_0^p]_{p=1}^{L}, [\sigma^p{}_1]_{p=1}^{L})$ on success or 0 on failure. Implementation is in Listing 6.

```
Input:   [(b^p, s^p)]_{p=1}^{L}                 -- private-keys
         m                                      -- message
Output:  ([σ0^p]_{p=1}^{L}, [σ1^p]_{p=1}^{L}) or 0   -- signature on success,
                                                -- 0 on failure
Procedure:
    [B_j]_{j=0}^{N/2-1} =RingGen()
    [I^p]_{p=1}^{L} =[H_point(b^pG)/b^p]_{p=1}^{L}

    mL2SHPoM.PreimageSetGen=λ.([B_j]_{j=0}^{N/2-1})

    mL2SHPoM.MemberSetGen=λ.([I^p]_{p=1}^{L})
    e=H_scalar(m)
    FirstProof=mL2SHPoM.GetProof([(b^p, s^p)]_{p=1}^{L}, e)
    If FirstProof==0 then Return 0
    [σ0^p]_{p=1}^{L} =FirstProof

    mL2SHPoM.MemberSetGen=λ.([G+I^p]_{p=1}^{L})
    mL2SHPoM.HashPoint=λ(X).(X+H_point(X))
    SecondProof=mL2SHPoM.GetProof([(b^p, s^p)]_{p=1}^{L}, e)
    If SecondProof==0 then Return 0
    [σ1^p]_{p=1}^{L} =SecondProof

    Return ([σ0^p]_{p=1}^{L}, [σ1^p]_{p=1}^{L})
```

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

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

```
Input:   m                                -- message
         ([σ0^p]_{p=1}^{L}, [σ1^p]_{p=1}^{L})     -- signature
Output:  (retcode, [I^p]_{p=1}^{L})        -- retcode is 0 or 1,
                                          -- [I^p]_{p=1}^{L} contains key-images

Procedure:
    [B_j]_{j=0}^{N/2-1} =RingGen()

    [Z0^p]_{p=1}^{L} =[σ0^p.Z0]_{p=1}^{L}    -- extract all Z0's from the first proof
    [I^p]_{p=1}^{L} =[Z0^p]_{p=1}^{L}        -- save them to [I^p]_{p=1}^{L}
    [Z0^p]_{p=1}^{L} =[σ1^p.Z0]_{p=1}^{L}    -- extract all Z0's from the second proof
    If [G+I^p]_{p=1}^{L} ≠[Z0^p]_{p=1}^{L}  then Return (0, [])
                                          -- Z0's in the second proof have to be
                                          -- equal to Z0's in the first proof plus G

    mL2SHPoM.PreimageSetGen=λ.([B_j]_{j=0}^{N/2-1})

    mL2SHPoM.MemberSetGen=λ.([I^p]_{p=1}^{L})
    e=H_scalar(m)
    If mL2SHPoM.Verif([σ0^p]_{p=1}^{L}, e)==0 then Return (0, [])

    mL2SHPoM.MemberSetGen=λ.([G+I^p]_{p=1}^{L})
    mL2SHPoM.HashPoint=λ(X).(X+H_point(X))
    If mL2SHPoM.Verif([σ0^p]_{p=1}^{L}, e)==0 then Return (0, [])

    Return (1, [I^p]_{p=1}^{L})
```

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:   ([I0^p]_{p=1}^{L}, [I1^p]_{p=1}^{L})  -- two key-image sets from two signatures
Output:  0 or 1                          -- 0 means the signatures are not-linked,
                                         -- 1 means the signatures are linked

Procedure:
    For i=1...L:
        If I0^i ∈[I1^p]_{p=1}^{L} then Return 1
    Return 0
```

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

The usage 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* distinct private-keys $[(b^p, s^p)]_{p=1}^{L}$ by calling the **mL2SLnkSig.Sign** and obtains a signature $([\sigma_0^p]_{p=1}^{L}, [\sigma_1^p]_{p=1}^{L})$.

- Verifier takes the message and the signature and calls **mL2SLnkSig.Verif** for them. If the call returns *(1,* $[I^p]_{p=1}^{L}$ *)*, the Verifier is convinced in that Prover signed the message *m* with the private-keys corresponding to some *L* distinct public-keys in the ring.
- Having repeated 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 in that there is at least one common private-key used for the both signatures.

8.3.2. mL2SLnkSig scheme completeness, soundness and witness-indistinguishably

The **mL2SLnkSig** scheme inherits the completeness and soundness from the **mL2SHPoM**.

As the **mL2SHPoM** scheme is zero-knowledge and the key-images reveal no information about the private-keys, it is not possible to distinguish witnesses, i.e., private-keys, from the signatures.

The only distinguishable thing about the private-keys is the case of linked signatures, i.e., the case when the **mL2SLnkSig.Link** returns *1*. Even revealing the fact of common witnesses, the signatures don't reveal any more information about them.

8.3.3. mL2SLnkSig complexities and some optimizations

The **mL2SLnkSig** signature size is 2 times the amount of data transmitted by a **mL2SHPoM** proof.

The **mL2SLnkSig** verification complexity is 2 times the complexity of the **mL2SHPoM** proof verification plus L^2 for the Z_0's check.

8.3.3.1. Complexities after simple optimizations

We can effectively exclude *L* Z_0's points from the second proof: $[\sigma_1{}^p.Z_0]_{p=1}^{L}$, as they are equal to Z_0's from the first proof plus *G*: $[\sigma_1{}^p.Z_0]_{p=1}^{L} = [\sigma_0{}^p.Z_0+G]_{p=1}^{L}$.

After that, the signature size is shown in Table 15.

	$\mathbb{G}$	$\mathbb{F}$
mL2SLnkSig	$L(2n+7)$	$2L(n+2)$

Table 15. **mL2SLnkSig** signature size.

We can skip calculating the $[Q_j]_{j=0}^{N/2-1}$ for the second proof on both Prover's and Verifier's sides and to use the $[Q_j]_{j=0}^{N/2-1}$ calculated for the first proof instead. Although they are different, they still keep the inner **mL2S** decoy set orthogonal and ensure the Com2 k_1's are zero, so they can be reused.

After that, the complexity is shown in Table 16, where $N=2^n$.

	multi-exp*(N)*	single-exp	H_{point}
mL2SLnkSig	*2*	$2nL+6L+2$	$N+1$

Table 16. **mL2SLnkSig** verification complexity.

[1] 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.
[2] Benedikt Bunz, 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] Shafi Goldwasser, Silvio Micali, Charles Rackoff. The knowledge complexity of interactive proof systems. In SIAM J. COMPUT. Vol 18, No 1. pp 186-208, February 1989
[4] 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.
[5] 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.
[6] 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
[7] Sarang Noether Triptych-2: efficient proofs for confidential transactions. Cryptology ePrint Archive, Report 2020/312, 2020. https://eprint.iacr.org/2020/312.pdf
[8] Nicolas Van Saberhagen. CryptoNote v 2.0, 2013. https://cryptonote.org/whitepaper.pdf
[9] 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
[10] Ronald Cramer, Ivan Damgard, Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. CRYPTO ’94, LNCS 839, pp. 174– 187. Springer-Verlag, 1994.
[11] U. Feige, A. Fiat, and A. Shamir. Zero-knowledge proofs of identity. J. Cryptology, 1:77–94, 1988.
[12] R. Rivest, A. Shamir, and Y. Tauman. How to leak a secret. Asiacrypt 2001, LNCS 2248, pp. 552–565. Springer-Verlag, 2001.
[13] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-S´ebastien Coron, editors, Advances in Cryptology – EUROCRYPT 2016, pages 305–326, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.
[14] Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. Zcash protocol specification. Tech. rep. 2016–1.10. Zerocoin Electric Coin Company, Tech. Rep., 2016.
[15] Russell WF Lai, Viktoria Ronge, Tim Ruffing, Dominique Schr¨oder, 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.
[16] 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
[17] C.P. Schnorr: Efficient Signature Generation by Smart Cards. J. Cryptology (1991) 4: 161-174
[18] E. Bresson, Y. Lakhnech, L. Mazare, B. Warinschi: A Generalization of DDH with Applications to Protocol Analysis and Computational Soundness. CRYPTO 2007, LNCS 4622, pp 482-499.
[19] 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.