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

Anton A. Sokolov

acmxddk@gmail.com

Full version

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

Keywords: Ring signature, linkable ring signature, log-size signature, threshold, membership proof, anonymity, zero-knowledge, disjunctive proof, unforgeability, non-frameability, witness-extended emulation.

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 an anonymity set or, interchangeably, 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 the stated problem plays a role in cryptographic applications, for instance, in the telecommunication and peer-to-peer distributed systems.

A formal notion of ring signatures and the early yet efficient schemes are presented in the works of Rivest, Shamir, and Tauman [22], Abe, Ohkubo, and Suzuki [1], Liu, Wei, and Wong [19], an example of a system that uses linkable ring signatures is, for instance, CryptoNote [25]. The nice features of these schemes are those there is no trusted setup process and no selected entities in them, an actual signer is allowed to form a ring in an ad hoc manner without notifying the other participants about this. All these signatures have sizes that grow linearly in the signer anonymity set size, their verification complexities are linear, too.

The schemes in [1, 19] and other linkable ring signature schemes can be instantiated with a prime-order cyclic group under the discrete logarithm problem hardness (DL) assumption. The scheme security and signer anonymity are usually, e.g. as in [19], reduced to one of the stronger hardness assumptions, e.g. to the decisional Diffie-Hellman (DDH) assumption in the random oracle model (ROM).

Recent works by Tsz Hon Yuen, Shi feng Sun, Joseph K. Liu, Man Ho Au, Muhammed F. Esgin, Qingzhao Zhang, and Dawu Gu [26], Sarang Noether and Brandon Goodell [20], Benjamin E. Diamond [3], Russell W. F. Lai, Viktoria Ronge, Tim Ruffing, Dominique Schröder, Sri Aravinda Krishnan Thyagarajan, and Jiafan Wang [17], William Black and Ryan Henry [4], and others show that under the common assumptions for a prime-order cyclic group where DL is hard 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 [15], and in some others it is shown that signer-ambiguous signatures with asymptotically lower than logarithmic sizes and lower than linear verification complexities can be built at the cost of requiring a trusted setup or bilinear pairings to an underlying prime-order group. However, this line of solutions is out of the scope of our current work.

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

1.1 CONTRIBUTION

1.1.1 LIN2-XOR AND LIN2-SELECTOR LEMMAS

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

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

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

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

With the Lin2-Selector lemma, no additional proof is required that the commitment Z has the form $k_0 P_s + k_1 Q_s$. After the lemma's n-round public coin protocol has been successfully completed, the verifier is convinced both in the form $Z = k_0 P_s + k_1 Q_s$, and in the prover's knowledge of (k_0, k_1, s). The Lin2-Xor and Lin2-Selector lemmas are proven for a prime-order group under the DL hardness assumption. The amount of data transmitted from a prover to a verifier during the Lin2-Selector protocol execution is logarithmic in the size of the element pair set $\left\{(P_j, Q_j)\right\}_{j=0}^{N-1}$, which we consider as a decoy set.

1.1.2 L2S SET MEMBERSHIP PROOF PROTOCOL AND MRL2SLNKSIG LINKABLE RING SIGNATURE

By defining prover's behavior for the Lin2-Selector lemma pure protocol, we create an interactive n-round public coin set membership proof protocol, called L2S. The L2S protocol inherits the properties of the Lin2-Selector lemma pure protocol and thus convinces the verifier that the commitment $Z = k_0 P_s + k_1 Q_s$ is built over a member (P_s, Q_s) of a set of element pairs with unknown discrete logarithmic relationship between the elements from all the pairs. We prove the L2S protocol is complete and sound under DL, special honest verifier zero-knowledge (sHVZK) under DDH.

Using the L2S protocol we construct a non-interactive sHVZK many-out-of-many MRL2SPoM membership proof scheme and, consequently, construct a many-out-of-many MRL2SLnkSig logarithmic-size linkable ring signature, which appears to be anonymous, unforgeable, and non-frameable under DDH in ROM. Moreover, under these assumptions the MRL2SLnkSig signature remains anonymous, unforgeable, and non-frameable even when its ring is composed of unevenly distributed or partially corrupted public keys. Therefore, we present MRL2SLnkSig as a general-purpose log-size solution for the linkable threshold ring signature problem.

1.1.3 NOVEL METHOD FOR CONSTRUCTING A LINKABLE RING SIGNATURE

In comparision to the setup-free log-size linkable ring signature schemes proposed in [26, 20, 3, 17], that originate from the ideas of Jens Groth and Markulf Kohlweiss [14] or from the ideas of Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell [6], our signature scheme is constructed on a basis different from [14, 6].

A parallel can be drawn with the work [14], which introduced a mechanism similar to the Kronecker's delta to select a member of the anonymity set without revealing it. Our signature uses the Lin2-Xor and, consequently, Lin2-Selector lemmas in exactly the same role. There is a difference in the anonymity sets: the anonymity sets in [14] are located in a plane of two orthogonal generators, while the anonymity sets for the Lin2-Selector lemma protocol are themselves orthogonal generator sets.

Thus, the Lin2-Xor lemma provides a new cryptographic primitive that can be used to construct a ring signature and possibly to construct other schemes. We also formulate and prove a somewhat stronger version of the Lin2-Xor lemma, Lin2-Xor-WEE, which includes witness extraction and which we use to prove the signature unforgeability.

1.2 METHOD OVERVIEW

1.2.1 LIN2 LEMMA

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

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

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

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

$$Z = aP + bQ$$

are certainly known to the prover.

1.2.2 LIN2-XOR LEMMA AND ITS COROLLARIES

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

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

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

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

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

$$wR = Z + r_1H_1 + r_2H_2$$

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

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

$$(Z = aP_1 + bQ_1) \oplus (Z = aP_2 + bQ_2).$$

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

Also, as a corollary, after the protocol successful completion the verifier is convinced that H_1 is a linear combination of either (P_1, Q_1) or (P_2, Q_2), that is, H_1 has a similar property

$$(H_1 = fP_1 + gQ_1) \oplus (H_1 = fP_2 + gQ_2)$$

for some known to the prover f, g. Moreover, as another corollary, if this game succeeds, then there is some known to the prover x such that the element $Z + r_1H_1$ has the following property

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

1.2.3 LIN2-SELECTOR LEMMA

It turns out that the Lin2-Xor lemma can be 'stacked', i.e. applied several times as an n-round game to an arbitrary number of fixed orthogonal elements. We assume the number of elements is a power of 2. For instance, for eight fixed orthogonal elements $P_1, Q_1, P_2, Q_2, P_3, Q_3, P_4, Q_4$, and for two fixed elements Z, H_1, the game will contain

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

where (c_{11}, c_{13}) is the first challenge and (r_1, H_2) is the first reply, (c_{21}, c_{23}) and (r_2, H_3) are the second challenge and reply, c_{31} and r_3 are the third challenge and reply.

In this game, Lin2-Selector lemma convinces the verifier that Z is exactly one of $(aP_1 + bQ_1)$, $(aP_2 + bQ_2)$, $(aP_3 + bQ_3)$, $(aP_4 + bQ_4)$ for some known to the prover a, b. Namely, by applying one of the Lin2-Xor lemma corollaries, it becomes proven that exactly one equality of the following two

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

holds for some known to the prover x. Applying the Lin2-Xor lemma to the equality that holds, suppose, to the first one, it becomes proven that Z is exactly one of $(aP_1 + bQ_1)$, $(aP_2 + bQ_2)$ for some a, b known to the prover. The same happens for the case when the second equality holds.

For a set of 2^{n-1} pairs $\left\{(P_j, Q_j)\right\}_{j=0}^{2^{n-1}-1}$, the Lin2-Selector lemma provides a general n-round reduction method that resembles the n-round reduction by B. Bünz et al. [6], although slightly different, for constructing R such that

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

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

1.2.4 LEMMA PROOFS, PURE PROTOCOLS, AND SOUNDNESS

Overall, the Lin2, Lin2-Xor, and Lin2-Selector lemmas have similar structure of their premises and conclusions in our work. The structure is this: a premise declares the necessary assumptions about the publicly seen values and defines what we call a pure protocol. The conclusion is that if the assumptions hold and the pure protocol is successfully completed, then the verifier is convinced that the prover knows some secret values.

A pure protocol specifies in detail what the verifier should do, however it does not specify the same for the prover. It only describes what the prover has to reply to the verifier, without specifying how to prepare the replies. With this minimum of information, we can prove soundness of the pure protocol, namely, that the protocol successful completion implies that the prover knows the secret values. The Lin2, Lin2-Xor, and Lin2-Selector lemmas provide proofs of soundness for their pure protocols. In other words, these lemmas state that for any prover strategy, including a dishonest one, the verifier is convinced that the prover knows certain secret values after the successful completion of the corresponding pure protocol.

We don't consider completeness and zero-knowledge for the pure protocols, since these properties depend on how the prover prepares the responses. If a pure protocol is proven to be sound, then a derived protocol that specifies prover's behavior in detail inherits the soundness. When the prover's behavior is fully defined in a derived protocol, we begin to consider its completeness and zero-knowledge.

As should be already seen, we use the term 'soundness' in the sense in which it is more often used in deductive logic rather than in cryptography, where this term is usually meant a shorthand for 'special soundness'. We distinguish between these two terms and use the term 'soundness' in the sense of the basic relationship between knowledge of secret values and successful completion of the protocol, while the term 'special soundness' in the stricter sense of witness extraction from a series of runs, as it is defined in cryptography, keeping in mind the latter always implies the former.

We assume that the prover and verifier are probabilistic polynomial-time Turing machines (PPT) equipped with a common tape on which they record their conversation transcript. When we prove soundness of a pure protocol, the verifier is assumed honest, while the prover is assumed to have a dishonest subroutine that gives with overwhelming probability acceptable replies to the uniformly random challenges such that the protocol succeeds.

To prove soundness of the Lin2 lemma protocol, we suppose that the secret values in question are not known to the prover. We consider two successful Lin2 lemma protocol transcripts, one of which is that of the prover-verifier conversation, and the other one is that the prover gets itself by calling the dishonest subroutine for another set of challenges taken from its random tape. We demonstrate a polynomial-time algorithm that extracts the secret values in question from these two transcripts using known to the prover information. Thus, we show that even without knowing these values, once the prover is able to successfully complete the protocol, it is able to obtain them in a polynomial time, and therefore the protocol is sound.

We use the same method for the Lin2-Xor lemma protocol with the only difference that we do not immediately demonstrate a polynomial time algorithm that finds the secret values if the protocol succeeds. Instead, we prove the possibility of such an algorithm, namely, we gradually find what values can be obtained by the prover in polynomial time, and finally show that the secret values in question are among them. For the Lin2-Selector lemma protocol we do the same using the Lin2 and Lin2-Xor lemmas. Thus we prove soundness of the lemma protocols.

1.2.5 SOUNDNESS, UNFORGEABILITY, AND WITNESS-EXTENDED EMULATION

So, we have proven that the protocols given in the Lin2, Lin2-Xor, and Lin2-Selector lemmas are sound. In their proofs we first and foremost used the logical inference based on the impossibility of circumventing the DL assumption. This inference is non-trivial, so, to make it easier to write down the inference chains, we introduce a tiny symbolic logic system that repeats the usual way of getting system properties from DL.

Having proven soundness of the Lin2-Selector lemma protocol, it is easy to construct a zero-knowledge membership proof protocol and a signature based on it. However, there still exists the question of how to prove unforgeability of such a signature. Here we have to start using the canvas of modern cryptography, where the methods for proving unforgeability have been already developed. To do this, first of all, we reformulate the three mentioned above lemmas as Lin2-WEE, Lin2-Xor-WEE, and Lin2-Selector-WEE, and prove the witness-extended emulation (WEE) property for each of their protocols. The WEE property can be thought of as a slightly increased soundness, the signature unforgeability appears to be provable with it.

The Lin2-WEE, Lin2-Xor-WEE, and Lin2-Selector-WEE lemmas are similar to their former counterparts, with the main difference that the logical inference used in the Lin2, Lin2-Xor, and Lin2-Selector lemmas is replaced by witness extraction algebra in them, although their witness extraction algebra often resembles the former logical inference. Another question is why did we leave the Lin2, Lin2-Xor, and Lin2-Selector lemmas, if there are their WEE counterparts providing what we need instead of them. The answer is that we think it makes sense to show the way that led us to the signature scheme, and then show a way to verify its unforgeability. To do the latter, we adapt the methods by Jens Groth and Markulf Kohlweiss [14] and by Joseph K. Liu, Victor K. Wei, and Duncan S. Wong [19].

1.2.6 L2S MEMBERSHIP PROOF, MRL2SLNKSIG SIGNATURE

We construct L2S set membership proof protocol on top of the Lin2-Selector lemma pure protocol, although, to be precise, we construct it on top of the Lin2-Selector-WEE lemma pure protocol. We prove that the L2S protocol is complete and sound, obtaining the soundness directly from the Lin2-Selector lemma, and also prove that it has witness-extended emulation, obtaining it from the Lin2-Selector-WEE lemma.

Then we analyze the L2S protocol transcript and show that all its records have distributions indistinguishable from independent and uniform randomness, except for one record, which is a linear combination of other records in the transcript. On this basis, we show that the L2S protocol is sHVZK and therefore doesn't reveal any information other than the fact of membership. This allows us to build an anonymous signature based on it.

The L2S protocol is efficient, it requires transmitting one point Z plus n scalar-point pairs (r_i, H_i), and computing one multi-exponentiation for N summands during verification. Here N translates to $2S_{sz}$, where S_{sz} is the anonymity set size, and $n = \log_2(N)$. Overall, in all schemes in this paper the value R (introduced in 1.2.3) is calculated as a multi-exponent only once during the verification, thus each scheme verification takes time about $(2 \ldots 4)S_{sz}/\log_2(S_{sz})$ plus $O(\log_2(S_{sz}))$, plus maybe not a big $O(S_{sz})$ related to the signature linking tags.

With a couple of auxiliary steps, using the Fiat-Shamir heuristic, we turn the L2S protocol into a non-interactive many-out-of-many proof of membership scheme MRL2SPoM and into a linkable threshold ring signature MRL2SLnkSig with a linking tag in the form $x^{-1}\mathbf{H_{point}}(P)$, where $P = xG$, and $\mathbf{H_{point}}$ is a hash to curve function. While the MRL2SPoM proof of membership scheme requires all elements of its anonymity set to be orthogonal to each other, the MRL2SLnkSig scheme removes this limitation by lifting the anonymity set to an orthogonal set of an $\mathbf{H_{point}}$-based hash function images, and then applying the MRL2SPoM to this orthogonal set.

2 PRELIMINARIES

- Let $\mathbb{G}$ be a cyclic group of prime order in which the discrete logarithm problem is hard, and let $\mathbb{F}$ be a scalar field of $\mathbb{G}$. The field $\mathbb{F}$ is finite, of the same order as $\mathbb{G}$.
- Let lowercase italic letters and words a, b, sum, ... denote scalars in $\mathbb{F}$. Sometimes indices and apostrophes are appended: a_{12}, b', s_1^p, 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 uppercase italic letters and words A, B, X, P, H, ... denote the elements of $\mathbb{G}$. Indices and apostrophes can be appended: A_1, B', X_{12}, P_{11}, Z_0^p, ... Also, uppercase italic letters denote sets and, sometimes, integers, that is clear from the context. The letters N and M are reserved for integer powers of 2.
- Let 0 denote the zero element of $\mathbb{G}$ and also denote the zero scalar in $\mathbb{F}$, it's easy to distinguish its meaning from the context.
- Let G be a generator of $\mathbb{G}$. As $\mathbb{G}$ is a prime-order group, any non-zero element A is a generator of $\mathbb{G}$, hence we assume G is an a-priory chosen element.

2.1 A NOTE ABOUT CONTEXT

All definitions and lemmas below are given in the context of a game between Prover and Verifier, unless otherwise stated. During the game Prover tries to convince Verifier that certain facts are true. For the sake of this, Prover may disclose some information to Verifier, the latter may pick some, e.g., random, challenges, send them to Prover and get some values back from it.

The game can contain multiple protocols that prescribe who provides what. Thus, playing the game Prover and Verifier execute protocols among themselves, so that Verifier gradually becomes convinced of the facts. A protocol can be translated into corresponding non-interactive scheme using the Fiat-Shamir heuristic in ROM. We start by proving our lemmas in the interactive setting, then they are translated into the non-interactive setting using the Fiat-Shamir heuristic.

2.2 DEFINITIONS

2.2.1 SECURITY PARAMETER AND CRS

We assume security parameter λ is equal to the logarithm of cardinality of $\mathbb{F}$. The cardinalities of $\mathbb{F}$ and $\mathbb{G}$ are equal to each other, so λ is equal to the logarithm of cardinality of $\mathbb{G}$ We omit mentioning λ in the protocols, implying polynomial time is the polynomial time in λ everywhere.

The same is about common reference string (CRS) that contains parameters of $\mathbb{G}$, G, and is implied silently passed to all the protocols and hash functions.

2.2.2 SETS AND VECTORS

Sets are assumed having cardinalities that are polynomial in λ everywhere, of course, excluding $\mathbb{G}$ and $\mathbb{F}$. Vectors are ordered sets.

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

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

2.2.3 KNOWN AND UNKNOWN DISCRETE LOGARITHM RELATION

We say that a discrete logarithm relation between any element A and a non-zero element B is known iff scalar x in the equation

$$A = xB$$

is known or can be efficiently calculated.

For any element A and for any finite set of non-zero elements $S = \{B_1, B_2, \ldots, B_n\}$, we say a discrete logarithm relation of A to S is known iff the scalars x_1, x_2, …, x_n in the equation

$$A = x_1 B_1 + x_2 B_2 + \ldots + x_n B_n.$$

can be efficiently calculated.

The term "efficiently calculated" means that a probabilistic polynomial-time algorithm (PPT) that solves the problem with a non-negligible probability can be demonstrated. Since all the sets in our paper have polynomial cardinality (excluding $\mathbb{G}$ and $\mathbb{F}$) and since all the proofs have polynomial numbers of steps, we consider the terms "efficiently calculated" and "known" as having the same meaning throughout.

If it's proven that it's infeasible to build a PPT for calculating x in $A = xB$, then we say that a discrete logarithm relation between A and B is unknown or, equivalently, that finding it is hard. The same is about the discrete logarithm relation of A to an element set S.

If we can't say that a discrete logarithm relation between A and B is known and, at the same time, if we don't have any proof about that it is unknown, then we say nothing. The same is about the relation of A to a set S.

2.2.4 DL AND DDH ASSUMPTIONS

The discrete logarithm assumption (DL) is defined as: for any non-zero element A, for a randomly and uniformly chosen scalar x, it is hard to find x from the pair (A, xA).

The decisional Diffie–Hellman assumption (DDH) is defined as: for any non-zero element H, for randomly and uniformly chosen scalar series $\{a\}$, $\{b\}$, $\{c\}$, it is hard to distinguish the series of triplets $\{(aH, bH, abH)\}$ and $\{(aH, bH, cH)\}$.

DDH implies DL. We assume DL holds for $\mathbb{G}$ everywhere. When we prove zero-knowledge, we assume that DDH holds for $\mathbb{G}$.

2.2.5 SHORTHANDS FOR THE KNOWN AND UNKNOWN DISCRETE LOGARITHM RELATIONS

To simplify reasoning about the discrete logarithm relation, we introduce several shorthands, which in turn form a tiny symbolic logic system.

For any two elements A and B such that $B \neq 0$, the simbol '$\sim$' in statement

$$A \sim B$$

denotes the fact of knowing the discrete logarithm relation between A and B.

If a discrete logarithm relation between A and B is unknown, we write

$$A \mathrel{!\sim} B.$$

Although the statement $A \mathrel{!\sim} B$ may look as an inverted $A \sim B$, it is not. These statements don't obey the law of excluded middle, the only assumed law and inference rule for them are:

- (not ($A \sim B$ and $A \mathrel{!\sim} B$)), meaning that it's not possible for a discrete logarithm relationship to be simultaneously known and unknown.
- (not $A \sim B$) $\Rightarrow A \mathrel{!\sim} B$, meaning that if knowing a discrete logarithm relationship between A and B leads to a contradiction, then it is assumed to be unknown.

Thus, the denotations $A \sim B$ and $A \mathrel{!\sim} B$ together with the above law and inference rule provide us with a symbolic logic system, which is a shorthand way for the common way of reasoning about knowledge of the discrete logarithm.

That is, instead of writing, e.g. *"suppose, x in $A = xB$ is known, then . . . logical chain . . . this is a contradiction, hence, solving $A = xB$ is hard"*, we write

$$(A \sim B \Rightarrow \ldots \textit{logical chain} \ldots \Rightarrow \text{Contradiction}) \Rightarrow A \mathrel{!\sim} B.$$

We will not go deeper into the properties of this symbolic logic system now. A typical way of obtaining new statements using it is to make a supposition $A \sim B$, see if it leads to a contradiction, and, if a contradiction is found, obtain the statements (not $A \sim B$) and $A \mathrel{!\sim} B$. Note that nothing can be obtained from supposition of $A \mathrel{!\sim} B$.

Likewise, for any element A and any finite number of non-zero elements $B_1, B_2, \ldots, B_n$, let's denote as

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

the fact of knowing the discrete logarithm relation of A to $\{B_1, B_2, \ldots, B_n\}$.

If finding a discrete logarithm relation of A to a set of non-zero elements $\{B_1, B_2, \ldots, B_n\}$ is hard, we write

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

The law and inference rule for these statements are similar to those for $A \sim B$ and $A \mathrel{!\sim} B$:

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

Also, for any elements A and B such that $B \neq 0$

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

2.2.6 ORTHOGONAL SETS

For any set $S = \{B_1, B_2, \ldots, B_n\}$ of non-zero elements, if for each element $B_i \in S$ holds $B_i \mathrel{!=} \operatorname{lin}(S \setminus \{B_i\})$, then we denote this fact as

$$\operatorname{ort}(S)$$

and call it an unknown discrete logarithm of each element in a set to the other elements in the set.

For any S, $\operatorname{ort}(S)$ 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.

2.2.7 EVIDENCE

Let's call a valid proof that Prover provides to Verifier and thereby convinces the latter of some fact as evidence of that fact. Thus, the game's goal is for Prover to convince Verifier of the facts using evidences.

For instance, if x in the relation $A = xB$ is known to Prover, we write this fact as

$$A \sim B \text{ for Prover.}$$

Evidence of this fact can simply be the x that Prover provides to Verifier so that the latter can verify that $A = xB$ (assuming A and B are already shared between them).

In general, any acceptable way to convince Verifier of the Prover's knowledge of x in $A \sim B$ can be considered as evidence of the above fact. For example, it can be 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 in order to distinguish the proofs of statements and lemmas from the proofs of facts that Prover provides to Verifier and the latter checks and accepts. For instance, we write

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

We call a protocol an evidence of a fact if successful completion of the protocol means that Verifier is convinced that the fact holds on the Prover's side with overwhelming probability. In this case the protocol is also called sound.

The term "evidence" resembles the term "argument of knowledge" defined in [6] or in [18]. However, the "argument of knowledge" is a stricter term, as e.g. by definition in [6] it requires the perfect completeness and witness-extended emulation, whereas "evidence" has only to be valid, i.e. to be sound. Also, we use the term "soundness" to denote the property that successful protocol completion implies overwhelming probability for a particular fact to hold on the Prover's side. The term "soundness" differs from the term "knowledge soundness" defined in [18] in that the latter requires existence of a knowledge extractor. In any case, as shown, e.g. in [6], "knowledge soundness" implies "soundness".

When an evidence is sent from Prover to Verifier, if it fails verification on the Verifier's side, then the protocol terminates with an error status. For some protocols we define function Verif that checks provided evidences, and a protocol immediately exits by error if Verif returns 0.

2.2.8 FIXED ELEMENTS

An element A is said to be fixed for a protocol if it remains unchanged during execution of the protocol. For example, A is fixed for a protocol if it is revealed at the beginning of the protocol and don't change later, or, when $A = xB$, if x and B are revealed at the beginning and aren't changed until the end of the protocol.

2.2.9 RANDOM CHOICE

We use only uniformly random choice of scalars over $\mathbb{F}$ everywhere and call it simply 'random choice'. It is assumed that the probability that a randomly chosen scalar will be equal to zero is negligible.

2.2.10 NEGLIGIBLE PROBABILITY AND CONTRADICTIONS

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

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

2.2.11 DECOY SETS AND THEIR CARDINALITY

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

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

It is assumed that the cardinality of any decoy set is polynomial in $\boldsymbol{\lambda}$. That is, it is assumed that the cardinality of a decoy set is much less than the cardinality of $\mathbb{F}$. An algorithm that looks through all the entries in a decoy set is assumed to run in a polynomial time.

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

2.2.12 LINEAR COMBINATIONS

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

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

The scalars in the sum are sometimes called “weights”, although they don’t carry any additional meaning except for being multipliers for the elements, i.e. the weights aren’t required to be comparable.

2.2.13 INDEX PAIRS

Index pairs for the scalars and elements are usually written without separating commas, like

$$a_{12}, \quad c_{i1}, \quad c_{ij}.$$

To avoid ambiguity, when a two-digit number is used as a single index, it is put into curly brackets, e.g.

$$X_{(12)}.$$

The separating comma and brackets are used for the case when an index pair is a compound expression, e.g.

$$c_{1,(j+1)}, c_{i,(2j+1)}, c_{(2i),(2j+1)}.$$

2.2.14 UNIQUENESS

Two vectors are called different if they have at least one position with different items.

We call a vector unique under certain conditions when it is possible to efficiently calculate exactly one vector that satisfies these conditions. Namely, when it is hard to calculate a different vector that satisfies these conditions.

For instance, the statement

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

means that the scalar vector $\vec{x}$ is efficiently computable and it’s hard to calculate a different vector $\vec{y}$ such that the expression $A = \sum_{i=1\ldots n} y_i B_i$ holds for it.

2.2.15 WITNESS

If $\mathcal{R}$ is a binary polynomial-time-decidable relation, and if $(u, w) \in \mathcal{R}$, then we call w a witness for the statement u. As a primer, the statement u can be defined as a commitment and the witness w as a corresponding opening.

Regarding evidences, the statement u can be viewed as an assertion about that a particular fact holds for Prover, and the witness w as the fact itself. The relation $\mathcal{R}$ is a relation that connects facts that take place on the Prover’s side with statements about them. From this angle of view, evidence is a proof of $(u, w) \in \mathcal{R}$.

2.2.16 WITNESS-EXTENDED EMULATION

To say that a protocol has computational witness-extended emulation (also called simply witness-extended emulation, abbreviated as WEE) we use definition of WEE from [6]. In a nutshell, by this definition, a pure protocol has WEE if there exists a PPT emulator that finds a witness from a Prover-Verifier successful conversation. The emulator is assumed equipped with an oracle that permits rewinding the conversation to a specific move and resuming it with fresh randomness for the Verifier from that move onwards.

It should be noted that the definition in [6] is for protocols, whereas we have adopted it for pure protocols without any loss. In fact, the definition in [6] requires the emulator to be able to elicit witness from any PPT Prover that provides acceptable responses, whereas we require the emulator to do this for a pure protocol, where Prover is defined only to the extent that it gives acceptable responses, which is essentially the same.

According to the Forking lemma, which is also provided in [6], a protocol has WEE if there exists a PPT extractor that finds a witness from a polynomially bounded tree of the Prover-Verifier successful conversation transcripts.

If a protocol has WEE, then it is sound, as the extraction of witness from a conversation or from a tree of transcripts indicates that the witness was somehow put there by Prover and, thus, indicates that Prover knows it. Therefore, a protocol having WEE is an evidence of the fact that Prover knows a witness.

2.2.17 SPECIAL HONEST VERIFIER ZERO-KNOWLEDGE

For special honest verifier zero-knowledge (sHVZK) we use definition from [6]. A protocol is sHVZK if there exists a PPT simulator capable of producing successful transcripts of the protocol, which are statistically indistinguishable from the space of honest Prover-Verifier conversation transcripts with the same challenges.

This definition can be regarded as a natural extension of sHVZK definition by R. Cramer et al. for Σ-protocols [7] to the n-round protocols.

3 PRELIMINARY LEMMAS

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

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

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

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

Proof:

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

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

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

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

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

is unique.

Proof: Suppose, $\vec{x}$ is not unique, i.e. A has one more representation, with different vector $\vec{y}$. Subtracting these two representations of A from each other we get

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

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

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

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

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

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, the following holds

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

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

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

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

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

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

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

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

By definition of lin(), each Y_i is a weighted sum of elements from S, and, as $S_j \cap S_k = \emptyset$, each element from S participates in no more than one of these sums. Hence, we have a representation of the zero element as a weighted sum of elements from S, where at least one weight is non-zero. This contradicts the ZeroRepresentation lemma. Thus, $\mathrm{ort}\,\left([Y_i]_{i=0}^{n}\right)$.

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

OrtHalfShift lemma:
For any two vectors of non-zero elements $[X_i]_{i=1}^{m}$ and $[Y_i]_{i=1}^{n}$ such that $m \geq 0$, $n \geq 0$, $(m + n) > 0$, and $S = ([X_i]_{i=1}^{m} \cup [Y_i]_{i=1}^{n})$, for any non-zero element F such that $F \mathrel{!}= \mathrm{lin}(S)$, the following holds

$$\mathrm{ort}(S) \Rightarrow \mathrm{ort}([X_i]_{i=1}^{m} \cup [Y_i + F]_{i=1}^{n})$$

Proof: Suppose the opposite, that is, by definitions of ort() and lin() there are two vectors of known scalars $[x_i]_{i=1}^{m}$ and $[y_i]_{i=1}^{n}$ such that there is at least one non-zero scalar in them and the following holds

$$0 = \sum_{i=1 \ldots m} x_i X_i + \sum_{i=1 \ldots n} y_i (Y_i + F) = \sum_{i=1 \ldots m} x_i X_i + \sum_{i=1 \ldots n} y_i Y_i + \left(\sum_{i=1 \ldots n} y_i \right) F.$$

Suppose, the $(\sum_{i=1 \ldots n} y_i)F$ summand is zero, then the rest of the above sum is also zero that contradicts $\mathrm{ort}(S)$. Hence, the $(\sum_{i=1 \ldots n} y_i)F$ summand is not zero. Dividing all the above sum by $(\sum_{i=1 \ldots n} y_i)$, we obtain $F = \mathrm{lin}(S)$ that contradicts $F \mathrel{!}= \mathrm{lin}(S)$. Thus, $\mathrm{ort}([X_i]_{i=1}^{m} \cup [Y_i + F]_{i=1}^{n})$.

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

$$Z = \mathrm{lin}\,(P, Q)$$

.

Table 1: Lin2 lemma protocol.

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

Proof: Note the protocol is not claimed to be a Σ-protocol. We have to prove only the following statement: if Verifier succeeds in checking $(P + cQ) \sim (Z + rH)$, where $(Z + rH) \neq 0, r \neq 0$, then Prover knows a, b such that $Z = aP + bQ$.

After the protocol (Table 1) successful completion Verifier is convinced that $(P + cQ) \sim (Z + rH)$ for Prover, where $(Z + rH) \neq 0, r \neq 0$. Hence, it is convinced that Prover knows t such that

$$P + cQ = tZ + trH \qquad (1)$$

Suppose, $t = 0 \Rightarrow P + cQ = 0 \Rightarrow P \sim Q \Rightarrow$ Contradiction to $P \mathrel{!\sim} Q$. Hence, $t \neq 0$.

Finding Z from the equality (1)

$$Z = (P + cQ)\,/t - rH. \qquad (2)$$

For another challenge c':

$$Z = (P + c'Q)/t' - r'H, \tag{3}$$

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

Eliminating Z from the equations (2) and (3): $(P + cQ)/t - rH = (P + c'Q)/t' - r'H \Rightarrow$

$$(1/t - 1/t')P + (c/t - c'/t')Q + (r' - r)H = 0. \tag{4}$$

Suppose, $(r' - r) = 0$. We have two possibilities with this supposition: $(1/t - 1/t') = (c/t - c'/t') = 0$ 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 (4):

$$H = (1/t - 1/t')/(r' - r)P + (c/t - c'/t')/(r' - r)Q. \tag{5}$$

Thus,

$$H = xP + yQ, \tag{6}$$

where

$$\begin{aligned} x &= (1/t - 1/t')/(r' - r), \\ y &= (c/t - c'/t')/(r' - r). \end{aligned} \tag{7}$$

Prover is able to efficiently calculate these x and y from the two successful transcripts.

Finding $Z = aP + bQ$ from (2) and (5):

$$Z = (1/t)P + (c/t)Q - r(1/t - 1/t')/(r' - r)P - r(c/t - c'/t')/(r' - r)Q \tag{8}$$

$\Rightarrow$

$$\begin{aligned} a &= 1/t - r(1/t - 1/t')/(r' - r), \\ b &= c/t - r(c/t - c'/t')/(r' - r). \end{aligned}$$

$\Rightarrow Z = \mathrm{lin}(P, Q)$ for Prover. Thus, the lemma is proven.

Corollary of Lin2 lemma:
Under the conditions of the Lin2 lemma, its protocol (Table 1) is an evidence of

$$H = \mathrm{lin}(P, Q) \;\wedge\; (Z + rH) = \mathrm{lin}(P, Q)$$

.

Proof: In the course of proving the Lin2 lemma, we have already shown that the element H is represented by the formula (6) with the known coefficients (7). Hence, by definition of lin(), $H = \mathrm{lin}(P, Q)$ for Prover.

Also, by definition of lin(), there are known to Prover scalars a, b, x, y such that $Z = aP + bQ$ and $H = xP + yQ$. Hence, $(Z + rH) = (a + rx)P + (b + ry)Q$ and, thus, $(Z + rH) = \mathrm{lin}(P, Q)$ for Prover.

4 LIN2-XOR LEMMA AND ITS COROLLARIES

Here we formulate and prove the main lemma of this paper using the auxiliary lemmas given above. We also prove two useful corollaries of the lemma. Moreover, we show that under a slightly stronger premise the lemma protocol admits witnesses extraction.

4.1 LIN2-XOR LEMMA

Lin2-Xor lemma:
For any four non-zero fixed elements $P_1, Q_1 P_2, Q_2$ such that $\mathrm{ort}(P_1, Q_1 P_2, Q_2)$, for any two non-zero fixed elements Z, H_1, the following protocol (Table 2) is an evidence of

$$Z = \mathrm{lin}(P_1, Q_1) \;\oplus\; Z = \mathrm{lin}(P_2, Q_2)$$

.

Table 2: Lin2-Xor lemma protocol.

	$\leftarrow$	Verifier picks two non-zero random scalars c_{11}, c_{13} and sends them to Prover
Prover returns a non-zero scalar r_1 and a non-zero element H_2	$\rightarrow$	Verifier checks $(Z + r_1 H_1) \neq 0$, $r_1 \neq 0$, $H_2 \neq 0$
	$\leftarrow$	Verifier picks a non-zero random scalar c_2 and sends it to Prover
Prover returns a non-zero scalar r_2 and an evidence of $(P_1 + c_{11}Q_1 + c_2P_2 + c_2c_{13}Q_2) \sim (Z + r_1H_1 + r_2H_2)$	$\rightarrow$	Verifier checks $(Z + r_1H_1 + r_2H_2) \neq 0$, $r_2 \neq 0$ Verifier checks the evidence of $(P_1 + c_{11}Q_1 + c_2P_2 + c_2c_{13}Q_2) \sim (Z + r_1H_1 + r_2H_2)$

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

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

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

Let's move the first two steps of the Lin2-Xor lemma protocol (Table 2) to its premise. After this, we get exactly the premise, protocol (Table 1), and conclusion of the Lin2 lemma with the expression substitution shown in Table 3. Thus, by the Corollary of Lin2 lemma, the Lin2-Xor lemma protocol is also an evidence of

$$(Z + r_1H_1) = \text{lin}(P_1 + c_{11}Q_1, P_2 + c_{13}Q_2) \tag{9}$$

Rewriting the evidence (9) using definition of lin(), we get on the Prover's side

$$(Z + r_1H_1) = a(P_1 + c_{11}Q_1) + b(P_2 + c_{13}Q_2), \tag{10}$$

where Verifier is convinced that the scalars a and b are known to Prover. Also, as $(Z + r_1H_1) \neq 0$, Verifier is convinced that at least one of a and b is non-zero.

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

$$\left(Z + r'_1H_1\right) = a'\left(P_1 + c'_{11}Q_1\right) + b'\left(P_2 + c'_{13}Q_2\right), \tag{11}$$

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

Excluding H_1 from the equations (10), (11), and extracting Z

$$\begin{gathered}(a(P_1 + c_{11}Q_1) + b(P_2 + c_{13}Q_2) - Z)/r_1 = \left(a'\left(P_1 + c'_{11}Q_1\right) + b'\left(P_2 + c'_{13}Q_2\right) - Z\right)/r'_1 \Rightarrow \\ \left(r_1 - r'_1\right)Z = r_1a'\left(P_1 + c'_{11}Q_1\right) + r_1b'\left(P_2 + c'_{13}Q_2\right) - r'_1a(P_1 + c_{11}Q_1) - r'_1b(P_2 + c_{13}Q_2).\end{gathered}$$

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

$$Z = \left(\left(r_1a' - r'_1a\right)P_1 + \left(r_1a'c'_{11} - r'_1ac_{11}\right)Q_1 + \left(r_1b' - r'_1b\right)P_2 + \left(r_1b'c'_{13} - r'_1bc_{13}\right)Q_2\right)/\left(r_1 - r'_1\right)$$

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

$$Z = k_1P_1 + k_2Q_1 + k_3P_2 + k_4Q_2, \tag{12}$$

where

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

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

Moreover, as $\operatorname{ort}(P_1, Q_1, P_2, Q_2)$ and as Z, P_1, Q_1, P_2, Q_2 are fixed by premise, by the OrtUniqueRepresentation lemma Verifier is convinced that the scalars k_1, k_2, k_3, k_4 are constants, i.e. they remain the same regardless of the challenges and replies. Also, at least one of k_1, k_2, k_3, k_4 is non-zero, as the opposite contradicts the premise of non-zero Z.

Now we will prove that the system of equalities (13), where k_1, k_2, k_3, k_4 are constants and at least one of them is non-zero, implies that Verifier is convinced that the following conjunction of four statements holds:

$$\bigwedge \begin{array}{l} ((k_1 \neq 0) \wedge (k_3 \neq 0)) \Rightarrow \text{Contradiction} \\ ((k_1 \neq 0) \wedge (k_4 \neq 0)) \Rightarrow \text{Contradiction} \\ ((k_2 \neq 0) \wedge (k_3 \neq 0)) \Rightarrow \text{Contradiction} \\ ((k_2 \neq 0) \wedge (k_4 \neq 0)) \Rightarrow \text{Contradiction} \end{array} \tag{14}$$

Here is a proof for the first statement in the conjunction (14). Let's suppose

$$k_1 \neq 0 \quad \text{and} \quad k_3 \neq 0. \tag{15}$$

Rewriting the formula for k_1 in the system (13), keeping in mind that $r_1 \neq 0$ and $r_1' \neq 0$

$$\begin{aligned} \left(r_1 - r_1'\right) k_1 = \left(r_1 a' - r_1' a\right) \quad &\Rightarrow \\ r_1 \left(a' - k_1\right) = r_1' \left(a - k_1\right) \quad &\Rightarrow \\ \left(a' - k_1\right) / r_1' = \left(a - k_1\right) / r_1 \end{aligned} \tag{16}$$

As the right-hand side of the equality (16) depends only on the first transcript, and the left-hand side depends only on the second one, Verifier is convinced that both sides are equal to some constant q known to Prover

$$\begin{aligned} \left(a' - k_1\right) / r_1' = q \quad \text{and} \quad \left(a - k_1\right) / r_1 = q \quad &\Rightarrow \\ a' = q r_1' + k_1 \quad \text{and} \quad a = q r_1 + k_1 \end{aligned} \tag{17}$$

Let $t = (k_2 / k_1)$, where $k_1 \neq 0$ by the supposition (15). Taking the formulae for k_2 and k_1 from the system (13) and dividing them

$$\begin{aligned} t \left(r_1 a' - r_1' a\right) = \left(r_1 a' c_{11}' - r_1' a c_{11}\right) \quad &\Rightarrow \\ r_1' a \left(c_{11} - t\right) = r_1 a' \left(c_{11}' - t\right) \quad &\Rightarrow \\ a \left(c_{11} - t\right) / r_1 = a' \left(c_{11}' - t\right) / r_1' \end{aligned} \tag{18}$$

As the right-hand side of the equality (18) depends only on the first transcript, and the left-hand side depends only on the second one, Verifier is convinced that both sides are equal to some constant v known to Prover

$$\begin{aligned} a \left(c_{11} - t\right) / r_1 = v \quad \text{and} \quad a' \left(c_{11}' - t\right) / r_1' = v \quad &\Rightarrow \\ v r_1 = a \left(c_{11} - t\right) \quad \text{and} \quad v r_1' = a' \left(c_{11}' - t\right) \end{aligned} \tag{19}$$

The constant v is non-zero, as the opposite immediately leads to $a = 0$ and $a' = 0$, and, consequently, leads to a contradiction with $k_1 \neq 0$. Writing down this,

$$v \neq 0. \tag{20}$$

Using formula for a from the equalities (17), we find r_1 from the equality (19) for $v r_1$

$$\begin{aligned} v r_1 = \left(q r_1 + k_1\right) \left(c_{11} - t\right) \quad &\Rightarrow \\ r_1 \left(v - q \left(c_{11} - t\right)\right) = k_1 \left(c_{11} - t\right) \quad &\Rightarrow \\ r_1 = k_1 \left(c_{11} - t\right) / \left(v - q \left(c_{11} - t\right)\right) \quad &\Rightarrow \\ r_1 = k_1 / \left(\left(v / \left(c_{11} - t\right)\right) - q\right) \end{aligned} \tag{21}$$

Note we have performed division by the expressions $(v - q(c_{11} - t))$ and $(c_{11} - t)$ above, as both they are non-zero with overwhelming probability. It can be seen that both these expressions have random uniform distributions containing only the randomness c_{11} and the constants v, t, q, where $v \neq 0$ according to (20).

Thus, if (15) holds, then according to (21) Verifier is convinced that r_1 has uniform random distribution and is composed of constants known to Prover together with the randomness c_{11}.

Likewise, using the formulae for k_3 and k_4 from (13), Verifier is convinced that if (15) holds, then r_1 is composed of some known to Prover constants u, s, p such that $u \neq 0$ and of the randomness c_{13}

$$r_1 = k_3/((u/(c_{13} - s)) - p). \tag{22}$$

If (15) holds, then according to the equations (21) and (22) the variable r_1 can be calculated from each of the two independent randomnesses c_{11} and c_{13}. Hence, excluding r_1 from (21) and (22), Verifier is convinced that Prover is able to calculate the randomness c_{11} from the constants and from the randomness c_{13}, that contradicts the independence of the randomnesses c_{11} and c_{13}.

Thus, we have proven the first statement in the conjunction (14). Namely, we have proven that on successful completion of the lemma protocol (Table 2) Verifier is convinced that at least one of k_1 and k_3 is zero.

To prove the second statement in (14) we rewrite the system (13) as

$$\left\{\begin{array}{l} k_1 = \left(r_1 a' - r_1' a\right) / \left(r_1 - r_1'\right) \\ k_2 = \left(r_1 a' c_{11}' - r_1' a c_{11}\right) / \left(r_1 - r_1'\right) \\ k_3 = \left(r_1 d' e_{13}' - r_1' d e_{13}\right) / \left(r_1 - r_1'\right) \\ k_4 = \left(r_1 d' - r_1' d\right) / \left(r_1 - r_1'\right) \end{array}\right. , \text{ where } \left\{\begin{array}{l} d = b c_{13} \\ d' = b' c_{13}' \\ e_{13} = (1/c_{13}) \\ e_{13}' = (1/c_{13}') \end{array}\right. \tag{23}$$

The rewritten system (23) is exactly the system (13) where k_3 and k_4 have swapped places. Moreover, the system (23) has the same properties as the system (13). Hence, using the formulae for k_3 and k_4 from (23), Verifier is convinced that

$$r_1 = k_4/((u'/(e_{13} - s')) - p') \tag{24}$$

for some known to Prover constants u', s', p' such that $u' \neq 0$. From the expressions (22) and (24) Verifier obtains the sought contradiction for the case if both k_1 and k_4 are non-zero. Thus, the second statement of the conjunction (14) is proven. Namely, Verifier is convinced that at least one of k_1 and k_4 is zero.

Likewise, swapping k_1 and k_2 in the system (13) the same way as it has been done for k_3 and k_4 in the system (23), the third and fourth statements in the conjunction (14) are proven.

Recalling the linear combination (12) $Z = k_1 P_1 + k_2 Q_1 + k_3 P_2 + k_4 Q_2$, where k_1, k_2, k_3, k_4 are known to Prover, the conjunction (14) implies that, by the definitions of evidence and lin(), Verifier has an evidence of

$$Z = \text{lin}(P_1, Q_1) \oplus Z = \text{lin}(P_2, Q_2).$$

Thus, the lemma is proven.

4.2 COROLLARIES

Corollary 1 of Lin2-Xor lemma:
Under the conditions of the Lin2-Xor lemma, its protocol (Table 2) is an evidence of

$$H_1 = \text{lin}(P_1, Q_1) \ \oplus \ H_1 = \text{lin}(P_2, Q_2)$$

.

Proof: Let's divide the equations (10) and (11) by r_1 and r_1' respectively. It is possible, as $r_1 \neq 0$ and $r_1' \neq 0$. Hence, we rewrite the equations (10) and (11) as

$$\left\{\begin{array}{l} (H_1 + \tilde{r_1} Z) = \tilde{a}(P_1 + c_{11} Q_1) + \tilde{b}(P_2 + c_{13} Q_2) \\ (H_1 + \tilde{r_1'} Z) = \tilde{a'}(P_1 + c_{11}' Q_1) + \tilde{b'}(P_2 + c_{13}' Q_2) \end{array}\right. , \text{ where } \left\{\begin{array}{l} \tilde{r_1} = 1/r_1,\ \tilde{a} = a/r_1,\ \tilde{b} = b/r_1 \\ \tilde{r_1'} = 1/r_1',\ \tilde{a'} = a'/r_1',\ \tilde{b'} = b'/r_1' \end{array}\right.$$

After that, in the same way as we did in the proof of the Lin2-Xor lemma we arrive at the conclusion of this corollary.

Corollary 2 of Lin2-Xor lemma:
Under the conditions of the Lin2-Xor lemma, its protocol (Table 2) is an evidence of

$$(Z = \text{lin}(P_1, Q_1) \wedge H_1 = \text{lin}(P_1, Q_1)) \ \oplus \ (Z = \text{lin}(P_2, Q_2) \wedge H_1 = \text{lin}(P_2, Q_2))$$

.

Proof: On the Lin2-Xor lemma protocol successful completion, by the Lin2-Xor lemma and by its Corollary 1, Verifier is convinced that

$$(Z = \text{lin}(P_1, Q_1) \oplus Z = \text{lin}(P_2, Q_2)) \;\wedge\; (H_1 = \text{lin}(P_1, Q_1) \oplus H_1 = \text{lin}(P_2, Q_2)) \text{ for Prover.}$$

Suppose, $(Z = \text{lin}\,(P_1, Q_1) \wedge H_1 = \text{lin}\,(P_2, Q_2))$ for Prover. By definition of lin(), Prover knows z_1, z_2, h_1, h_2 such that $(Z = z_1 P_1 + z_2 Q_1$ and $H_1 = h_1 P_2 + h_2 Q_2)$. Hence, (10) rewrites as

$$z_1 P_1 + z_2 Q_1 + r_1\,(h_1 P_2 + h_2 Q_2) = a\,(P_1 + c_{11} Q_1) + b\,(P_2 + c_{13} Q_2)\,. \tag{25}$$

By the OrtUniqueRepresentation lemma, as ort(P_1, Q_1, P_2, Q_2) holds, from the equality (25) we have

$$z_1 = a \tag{26}$$

$$z_2 = a c_{11}\,, \tag{27}$$

If $a = 0$, then from the equalities (26) and (27) we obtain $z_1 = 0$ and $z_2 = 0$, which is a contradiction to $Z \neq 0$. Hence, as $a \neq 0$, it is possible to divide the equality (27) by the equality (26)

$$z_2/z_1 = c_{11}\,.$$

However, z_1, z_2 are constants, as Z, P_1, Q_1 are fixed. Hence, z_2/z_1 can't be equal to the random choice c_{11}, contradiction. Thus, the supposition about that $(Z = \text{lin}\,(P_1, Q_1) \wedge H_1 = \text{lin}\,(P_2, Q_2))$ holds for Prover is wrong.

Likewise, the case of $(Z = \text{lin}\,(P_2, Q_2) \wedge H_1 = \text{lin}\,(P_1, Q_1))$ is not possible. Thus, we have arrived at the conclusion of this corollary.

Corollary 3 of Lin2-Xor lemma:
Under the conditions of the Lin2-Xor lemma, its protocol (Table 2) is an evidence of

$$\begin{pmatrix} Z = \text{lin}(P_1, Q_1) & \wedge \\ H_1 = \text{lin}(P_1, Q_1) & \wedge \\ ((Z + r_1 H_1) \sim (P_1 + c_{11} Q_1)) & \wedge \\ ((Z + r_1 H_1)\,!\!\sim\!(P_2 + c_{13} Q_2)) & \end{pmatrix} \oplus \begin{pmatrix} Z = \text{lin}(P_2, Q_2) & \wedge \\ H_1 = \text{lin}(P_2, Q_2) & \wedge \\ ((Z + r_1 H_1) \sim (P_2 + c_{13} Q_2)) & \wedge \\ ((Z + r_1 H_1)\,!\!\sim\!(P_1 + c_{11} Q_1)) & \end{pmatrix}$$

Proof: According to Corollary 2 of Lin2-Xor lemma, there are only two possible cases

$$(Z = \text{lin}(P_1, Q_1) \wedge H_1 = \text{lin}(P_1, Q_1)) \;\oplus\; (Z = \text{lin}(P_2, Q_2) \wedge H_1 = \text{lin}(P_2, Q_2)) \;\text{ on the Prover's side.}$$

If $(Z = \text{lin}\,(P_1, Q_1) \wedge H_1 = \text{lin}\,(P_1, Q_1))$ for Prover, then using definition of lin() we obtain

$$(Z + r_1 H_1) = \text{lin}\,(P_1, Q_1) \;\text{ for Prover.} \tag{28}$$

At the same time, according to (10), Verifier is convinced that Prover knows a, b in

$$(Z + r_1 H_1) = a\,(P_1 + c_{11} Q_1) + b\,(P_2 + c_{13} Q_2)\,. \tag{29}$$

Combining the expressions (28) and (29), by the OrtUniqueRepresentation lemma, definitions of lin() and '$\sim$', we obtain

$$(Z + r_1 H_1) \sim (P_1 + c_{11} Q_1) \;\text{ for Prover.} \tag{30}$$

Suppose, $(Z + r_1 H_1) \sim (P_2 + c_{13} Q_2)$ holds for Prover simultaneously with the expression (30). This contradicts the OrtUniqueRepresentation lemma, as the element $(Z + r_1 H_1)$ gets two representations: $a\,(P_1 + c_{11} Q_1)$ and $b\,(P_2 + c_{13} Q_2)$, where a and b are known to Prover. Hence, $(Z + r_1 H_1)\,!\!\sim (P_2 + c_{13} Q_2)$ for Prover.

Thus, we have proven the first part of this corollary. Namely, we have proven that the expression

$$\begin{pmatrix} Z = \text{lin}(P_1, Q_1) & \wedge \\ H_1 = \text{lin}(P_1, Q_1) & \wedge \\ ((Z + r_1 H_1) \sim (P_1 + c_{11} Q_1)) & \wedge \\ ((Z + r_1 H_1)\,!\!\sim\!(P_2 + c_{13} Q_2)) & \end{pmatrix}$$

holds, when $(Z = \text{lin}\,(P_1, Q_1) \wedge H_1 = \text{lin}\,(P_1, Q_1))$ holds for Prover.

The same way we prove the second part, namely, that the expression

$$\begin{pmatrix} Z = \text{lin}(P_2, Q_2) & \wedge \\ H_1 = \text{lin}(P_2, Q_2) & \wedge \\ ((Z + r_1 H_1) \sim (P_2 + c_{13} Q_2)) & \wedge \\ ((Z + r_1 H_1)\,!\!\sim\!(P_1 + c_{11} Q_1)) & \end{pmatrix}$$

holds, when $(Z = \text{lin}\,(P_2, Q_2) \wedge H_1 = \text{lin}\,(P_2, Q_2))$ holds for Prover. Thus, this corollary is proven.

4.3 WITNESS EXTRACTION

Here we will reformulate the Lin2, Lin2-Xor lemmas and their corollaries so that, in addition to soundness, we also get witness-extended emulation for their protocols.

4.3.1 LIN2 LEMMA PROTOCOL WITNESS EXTRACTION

Let's look again at the Lin2 lemma protocol shown in Table 1 and consider the element H in the Lin2 lemma premise as the first message from Prover to Verifier. We will separate the evidence of $(P + cQ) \sim (Z + rH)$ from the protocol and assume that the evidence is provided by means of another protocol, which is played immediately after the Verifier's check of $(Z + rH) \neq 0, r \neq 0$.

Thus, for the Lin2 lemma, we have 1-round protocol followed by some n-round evidence protocol as shown in Table 4. We will assume that the n-round evidence protocol has witness-extended emulation. Now we will show that under this assumption the rewritten Lin2 lemma protocol (Table 4) also has witness-extended emulation.

Lin2-WEE lemma:
For any three non-zero fixed elements P, Q, Z such that $P \,!\!\sim Q$, for the relation $\mathcal{R} = \{(Z, (a, b)) \mid Z = aP + bQ\}$, the following protocol (Table 4) has witness-extended emulation.

Table 4: Lin2-WEE lemma protocol, rewritten protocol of Lin2 lemma.

Prover sends a non-zero element H	$\rightarrow$	Verifier checks $H \neq 0$
	$\leftarrow$	Verifier picks a non-zero random scalar c and sends it to Prover
Prover returns a non-zero scalar r	$\rightarrow$	Verifier checks $(Z + rH) \neq 0$, $r \neq 0$
Prover and Verifier play a WEE protocol of the evidence $(P + cQ) \sim (Z + rH)$.		

Proof: For this lemma protocol (Table 4), let's build a PPT emulator that will satisfy the definition of witness-extended emulation.

Suppose, the emulator is fed with a successful transcript of the protocol for some arbitrary Z such that $Z \neq 0$. The transcript has a random challenge c and a reply r. Also, it has, as a sub-transcript, a successful transcript of a WEE protocol of the evidence $(P + cQ) \sim (Z + rH)$.

By definition of WEE, properly unwinding the game for the evidence $(P + cQ) \sim (Z + rH)$ the emulator gets witness w such that

$$w(P + cQ) = (Z + rH). \tag{31}$$

Likewise, unwinding the protocol to the point where the challenge c was generated and generating it anew as c', the emulator gets witness w' for the challenge c' with reply r' such that

$$w'(P + c'Q) = (Z + r'H). \tag{32}$$

As has been shown in the Lin2 lemma proof, with overwhelming probability $c \neq c'$ and $r \neq r'$, so subtracting the equations (31) and (32) from each other

$$(w' - w)P + (w'c' - wc)Q = (r' - r)H,$$

the emulator obtains

$$H = ((w' - w)/(r' - r))P + ((w'c' - wc)/(r' - r))Q, \tag{33}$$
$$Z = (w - r(w' - w)/(r' - r))P + (wc - r(w'c' - wc)/(r' - r))Q, \tag{34}$$

that is, from the equality (34)

$$Z = aP + aQ,$$

where

$$a = w - r(w' - w)/(r' - r),$$
$$b = wc - r(w'c' - wc)/(r' - r).$$

Thus, we have shown that the emulator is able to obtain witness (a, b) for a statement Z such that $(Z, (a, b)) \in \mathcal{R}$. Hence, by definition of WEE, under this lemma conditions the lemma protocol (Table 4) has witness-extended emulation.

Corollary of Lin2-WEE lemma:
Under the conditions of the Lin2-WEE lemma, for element H sent in the first message of the protocol (Table 4), there is a witness-extended emulation algorithm for the protocol (Table 4), which is capable of extracting witness for the relation $\mathcal{R}_H = \{(H, (x, y)) \mid H = xP + yQ\}$.

Proof: In the course of proving the Lin2-WEE lemma, we have already shown that the element H is represented by the formula (33), where the coefficients $(w' - w)/(r' - r)$ and $(w'c' - wc)/(r' - r)$ in the linear combination of H with respect to P and Q are known. These coefficients are the witness sought.

4.3.2 LIN2-XOR LEMMA PROTOCOL WITNESS EXTRACTION

Now, looking at the Lin2-Xor lemma protocol (Table 2) let's figure out how we can rewrite it similar way as we did with the Lin2 lemma protocol (Tables 1, 4). The element H_1 from the Lin2-Xor lemma premise becomes the first message. Next, the first round with the challenge pair (c_1, c_3) and corresponding reply r_1 continues to completion. After that, as the second round, the Lin2-WEE lemma protocol (Table 4) with the symbolic substitutions as per Table 3 (with H_2 transmitted as its first message) could be played.

It should be noted that the Lin2 lemma protocol (Table 1) with substitutions (Table 3) is played in the Lin2-Xor lemma protocol (Table 2) second round only for the sake of obtaining the evidence (9), and for nothing else. Hence, we can drop the requirement to use exactly the Lin2 lemma protocol and instead to require any protocol, as long as it provides the evidence (9) and has WEE.

Overall, the rewritten protocol is shown in Table 5. Compared to the Lin2-Xor lemma protocol (Table 2), it has a stronger precondition in the sense that its sub-protocol for obtaining the evidence (9) requires WEE, and at the same time the precondition is relaxed in the sense that any WEE protocol providing the evidence (9) will suffice.

As a side note, to ease modeling of the protocol rewinding and resuming, transmission of the challenge pair (c_{11}, c_{13}) in the first round can be thought of as made of two subsequent sub-rounds, where c_{11} is transmitted in the first sub-round, and c_{13} in the second one.

Lin2-Xor-WEE lemma:
For any five non-zero fixed elements P_1, Q_1, P_2, Q_2, Z such that $\operatorname{ort}(P_1, Q_1, P_2, Q_2)$ holds, for the relation $\mathcal{R} = \{(Z, (x, y)) \mid (Z = xP_1 + yQ_1) \oplus (Z = xP_2 + yQ_2)\}$, the following protocol (Table 5) has witness-extended emulation.

Table 5: Lin2-Xor-WEE lemma protocol, rewritten protocol of Lin2-Xor lemma.

Prover sends a non-zero element H_1	$\rightarrow$	Verifier checks $H_1 \neq 0$
	$\leftarrow$	Verifier picks two non-zero random scalars c_{11}, c_{13} and sends them to Prover
Prover returns a non-zero scalar r_1. (Note H_2 is sent thereafter as the first message of the subsequent Lin2 lemma WEE protocol, in the case if it is used.)	$\rightarrow$	Verifier checks $(Z + r_1H_1) \neq 0$, $r_1 \neq 0$
Prover and Verifier play a WEE protocol of the evidence $(Z + r_1H_1) = \operatorname{lin}(P_1 + c_{11}Q_1, P_2 + c_{13}Q_2)$. (For instance, they play the Lin2 lemma WEE protocol from Table 4 with substitutions as per Table 3.)		

Proof: For this lemma protocol (Table 5), let's build a PPT emulator that will satisfy the definition of witness-extended emulation.

Suppose, the emulator is fed with a successful transcript of the protocol for some arbitrary Z such that $Z \neq 0$. The transcript has the random challenge pair (c_{11}, c_{13}) and reply r_1. Also it has, as a sub-transcript, a successful transcript of a WEE protocol of the evidence (9), namely, of $(Z + r_1H_1) = \operatorname{lin}(P_1 + c_{11}Q_1, P_2 + c_{13}Q_2)$ for Prover.

By definition of WEE, properly unwinding the game for the evidence $(Z + r_1H_1) = \operatorname{lin}(P_1 + c_{11}Q_1, P_2 + c_{13}Q_2)$, the emulator gets a witness pair (a, b) such that

$$(Z + r_1H_1) = a(P_1 + c_{11}Q_1) + b(P_2 + c_{13}Q_2). \tag{35}$$

Rewinding the protocol to the point where the first challenge pair was generated, and continuing to completion, properly unwinding the game for the aforementioned evidence, the emulator gets the other challenge pair (c'_{11}, c'_{13}), reply r'_1, and withess pair (a', b') such that

$$(Z + r'_1 H_1) = a'(P_1 + c'_{11} Q_1) + b'(P_2 + c'_{13} Q_2). \tag{36}$$

Subtracting the equalities (35) and (36) from each other, the emulator gets

$$(r'_1 - r_1)H_1 = (a' - a)P_1 + (a'c'_{11} - ac_{11})Q_1 + (b' - b)P_2 + (b'c'_{13} - bc_{13})Q_2. \tag{37}$$

Suppose, $(r'_1 - r_1) = 0$. In this case, the left-hand side of the equality (37) becomes equal to zero, and thus, if at least one of the weights for P_1, Q_1, P_2, Q_2 on the right-hand side of (37) is non-zero, then $\mathrm{ort}(P_1, Q_1, P_2, Q_2)$ is not satisfied, which contradicts the premise. At the same time, if all the weights for P_1, Q_1, P_2, Q_2 are zeros, then from (37) is seen that $a = a' = b = b' = 0$, and from (35) is seen that $(Z + r_1 H_1) = 0$, which is a contradiction to the protocol. Therefore,

$$(r'_1 - r_1) \neq 0\ . \tag{38}$$

Thus, the emulator gets the weights h_1, h_2, h_3, h_4 such that

$$\begin{cases} h_1 = (a' - a)/(r'_1 - r_1) \\ h_2 = (a'c'_{11} - ac_{11})/(r'_1 - r_1) \\ h_3 = (b' - b)/(r'_1 - r_1) \\ h_4 = (b'c'_{13} - bc_{13})/(r'_1 - r_1)\ , \end{cases} \tag{39}$$

$$(h_1 \neq 0) \vee (h_2 \neq 0) \vee (h_3 \neq 0) \vee (h_4 \neq 0)\ , \tag{40}$$

and

$$H_1 = h_1 P_1 + h_2 Q_1 + h_3 P_2 + h_4 Q_2\ . \tag{41}$$

For any other couple of transcripts of this protocol with the same P_1, Q_1, P_2, Q_2 and H_1, the weights h_1, h_2, h_3, h_4 calculated by the formulae (39) will be the same, since the opposite contradicts the premise of $\mathrm{ort}(P_1, Q_1, P_2, Q_2)$. Namely, having two distinct decompositions of H_1 the emulator is able to subtract them from each other and, thus, to demonstrate an example that violates the orthogonality of P_1, Q_1, P_2, Q_2.

The emulator unwinds again to the point where the first challenge pair was generated and, resuming onward, obtains new $(c''_{11}, c''_{13}), r''_1, a'', b''$. As with the system (39), the emulator has the following system for these new values

$$\begin{cases} h_1 = (a'' - a)/(r''_1 - r_1) \\ h_2 = (a''c''_{11} - ac_{11})/(r''_1 - r_1) \\ h_3 = (b'' - b)/(r''_1 - r_1) \\ h_4 = (b''c''_{13} - bc_{13})/(r''_1 - r_1)\ . \end{cases} \tag{42}$$

The systems (39) and (42) can be unified and rewritten without any loss into a single system as follows

$$\begin{cases} h_1(r'_1 - r_1) = a' - a \\ h_2(r'_1 - r_1) = a'c'_{11} - ac_{11} \\ h_3(r'_1 - r_1) = b' - b \\ h_4(r'_1 - r_1) = b'c'_{13} - bc_{13} \\ h_1(r''_1 - r_1) = a'' - a \\ h_2(r''_1 - r_1) = a''c''_{11} - ac_{11} \\ h_3(r''_1 - r_1) = b'' - b \\ h_4(r''_1 - r_1) = b''c''_{13} - bc_{13}\ . \end{cases} \tag{43}$$

Or, in matrix form, as

$$\begin{bmatrix} -1 & 1 & 0 & 0 & 0 & 0 & -h_1 & 0 \\ -c_{11} & c'_{11} & 0 & 0 & 0 & 0 & -h_2 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 & 0 & -h_1 \\ -c_{11} & 0 & c''_{11} & 0 & 0 & 0 & 0 & -h_2 \\ 0 & 0 & 0 & -1 & 1 & 0 & -h_3 & 0 \\ 0 & 0 & 0 & -c_{13} & c'_{13} & 0 & -h_4 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 & 0 & -h_3 \\ 0 & 0 & 0 & -c_{13} & 0 & c''_{13} & 0 & -h_4 \end{bmatrix} \times \begin{bmatrix} a \\ a' \\ a'' \\ b \\ b' \\ b'' \\ (r'_1 - r_1) \\ (r''_1 - r_1) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{44}$$

The right-hand side of the equality (44) is the zero vector. At the same time, the matrix on the left-hand side of the equality (44), taking into account the inequality (38), is multiplied by a non-zero vector. Therefore, the determinant of the matrix must be equal to zero, that is

$$\det \begin{bmatrix} -1 & 1 & 0 & 0 & 0 & 0 & -h_1 & 0 \\ -c_{11} & c'_{11} & 0 & 0 & 0 & 0 & -h_2 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 & 0 & -h_1 \\ -c_{11} & 0 & c''_{11} & 0 & 0 & 0 & 0 & -h_2 \\ 0 & 0 & 0 & -1 & 1 & 0 & -h_3 & 0 \\ 0 & 0 & 0 & -c_{13} & c'_{13} & 0 & -h_4 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 & 0 & -h_3 \\ 0 & 0 & 0 & -c_{13} & 0 & c''_{13} & 0 & -h_4 \end{bmatrix} = 0 \,. \tag{45}$$

It can be seen that the equality (45) connecting the random challenges and weights h_1, h_2, h_3, h_4 together may contain some constraints on the choice of the weights. The emulator will now find what these constraints are. Using Laplace expansion with respect to the column, where c''_{13} is placed in the determinant, the equality (45) rewrites as an equality for the minors

$$c''_{13} \det \begin{bmatrix} -1 & 1 & 0 & 0 & 0 & -h_1 & 0 \\ -c_{11} & c'_{11} & 0 & 0 & 0 & -h_2 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 & -h_1 \\ -c_{11} & 0 & c''_{11} & 0 & 0 & 0 & -h_2 \\ 0 & 0 & 0 & -1 & 1 & -h_3 & 0 \\ 0 & 0 & 0 & -c_{13} & c'_{13} & -h_4 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & -h_3 \end{bmatrix} = \det \begin{bmatrix} -1 & 1 & 0 & 0 & 0 & -h_1 & 0 \\ -c_{11} & c'_{11} & 0 & 0 & 0 & -h_2 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 & -h_1 \\ -c_{11} & 0 & c''_{11} & 0 & 0 & 0 & -h_2 \\ 0 & 0 & 0 & -1 & 1 & -h_3 & 0 \\ 0 & 0 & 0 & -c_{13} & c'_{13} & -h_4 & 0 \\ 0 & 0 & 0 & -c_{13} & 0 & 0 & -h_4 \end{bmatrix} . \tag{46}$$

Unwinding to the point where c''_{13} was generated, and resuming, the emulator gets equality (46) for another value of the randomness c''_{13} and thus obtains that both determinants in (46) are necessarily equal to zero, it uses only the first one

$$\det \begin{bmatrix} -1 & 1 & 0 & 0 & 0 & -h_1 & 0 \\ -c_{11} & c'_{11} & 0 & 0 & 0 & -h_2 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 & -h_1 \\ -c_{11} & 0 & c''_{11} & 0 & 0 & 0 & -h_2 \\ 0 & 0 & 0 & -1 & 1 & -h_3 & 0 \\ 0 & 0 & 0 & -c_{13} & c'_{13} & -h_4 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & -h_3 \end{bmatrix} = 0 \,. \tag{47}$$

Doing the same for the determinant (47) with respect to the randomness c''_{11} column, the emulator obtains

$$\det \begin{bmatrix} -1 & 1 & 0 & 0 & -h_1 & 0 \\ -c_{11} & c'_{11} & 0 & 0 & -h_2 & 0 \\ -1 & 0 & 0 & 0 & 0 & -h_1 \\ 0 & 0 & -1 & 1 & -h_3 & 0 \\ 0 & 0 & -c_{13} & c'_{13} & -h_4 & 0 \\ 0 & 0 & -1 & 0 & 0 & -h_3 \end{bmatrix} = 0 \,. \tag{48}$$

Repeating the same for the determinant (48) with respect to the randomness c'_{13} column, the emulator obtains two more equalities

$$\det \begin{bmatrix} -1 & 1 & 0 & -h_1 & 0 \\ -c_{11} & c'_{11} & 0 & -h_2 & 0 \\ -1 & 0 & 0 & 0 & -h_1 \\ 0 & 0 & -1 & -h_3 & 0 \\ 0 & 0 & -1 & 0 & -h_3 \end{bmatrix} = 0 \,, \tag{49}$$

$$\det \begin{bmatrix} -1 & 1 & 0 & -h_1 & 0 \\ -c_{11} & c'_{11} & 0 & -h_2 & 0 \\ -1 & 0 & 0 & 0 & -h_1 \\ 0 & 0 & -c_{13} & -h_4 & 0 \\ 0 & 0 & -1 & 0 & -h_3 \end{bmatrix} = 0 \,. \tag{50}$$

Repeating the same for the determinant (49) with respect to the randomness c'_{11} column, the emulator obtains an equality for the minor of '1' in the column

$$\det\begin{bmatrix} -c_{11} & 0 & -h_2 & 0 \\ -1 & 0 & 0 & -h_1 \\ 0 & -1 & -h_3 & 0 \\ 0 & -1 & 0 & -h_3 \end{bmatrix} = 0 \,. \tag{51}$$

Repeating the same for the determinant (51) with respect to the randomness c_{11} column, the emulator obtains two equalities

$$\det\begin{bmatrix} 0 & 0 & -h_1 \\ -1 & -h_3 & 0 \\ -1 & 0 & -h_3 \end{bmatrix} = 0 \,, \tag{52}$$

$$\det\begin{bmatrix} 0 & -h_2 & 0 \\ -1 & -h_3 & 0 \\ -1 & 0 & -h_3 \end{bmatrix} = 0 \,. \tag{53}$$

From the equalities (52) and (53) the emulator finds

$$h_1 h_3 = 0 \,, \tag{54}$$
$$h_2 h_3 = 0 \,. \tag{55}$$

Likewise, from the equality (50) the emulator finds

$$h_1 h_4 = 0 \,, \tag{56}$$
$$h_2 h_4 = 0 \,. \tag{57}$$

Combining the equalities (40), (41), (54), (55), (56), (57) the emulator finds that the following holds

$$(h_1 = 0 \wedge h_2 = 0) \oplus (h_3 = 0 \wedge h_4 = 0) \,, \tag{58}$$

$$\left(\begin{array}{l} H_1 = uP_1 + vQ_1 \,, \\ u = h_1 \,, \\ v = h_2 \end{array} \right) \oplus \left(\begin{array}{l} H_1 = uP_2 + vQ_2 \,, \\ u = h_3 \,, \\ v = h_4 \end{array} \right) . \tag{59}$$

From (43) and (58) the emulator finds that the witnesses a and b in the equality (35) meet the following condition

$$(a = 0) \oplus (b = 0) \,. \tag{60}$$

From the equality (35), taking into account the known weights h_1, h_2, h_3, h_4, which were calculated by the formulae (42), the constraint (60), using the decomposition (59), from the known challenges and known values of a, b, r the emulator finds the sought witness pair (x, y)

$$\left(\begin{array}{l} Z = xP_1 + yQ_1 \,, \\ x = a - r_1 h_1 \,, \\ y = ac_{11} - r_1 h_2 \end{array} \right) \oplus \left(\begin{array}{l} Z = xP_2 + yQ_2 \,, \\ x = b - r_1 h_3 \,, \\ y = bc_{13} - r_1 h_4 \end{array} \right) \tag{61}$$

Thus, the lemma is proven.

Corollary 1 of Lin2-Xor-WEE lemma:
Under the conditions of the Lin2-Xor-WEE lemma, for the element H_1 sent in the first message of its protocol, for $\mathcal{R}_{H_1} = \{(H_1, (u, v)) \mid (H_1 = uP_1 + vQ_1) \oplus (H_1 = uP_2 + vQ_2)\}$, there is a witness-extended emulation algorithm for the lemma protocol (Table 5), which is capable of extracting witness for the relation $\mathcal{R}_{H_1}$.

Proof: In the course of proving the Lin2-Xor-WEE lemma, we have already shown that the element H_1 is represented by the formula (59), where the coefficients in the linear combinations of H_1 with respect to P_1, Q_1, P_2, Q_2 are efficiently calculated by the formulae (42). These coefficients, which satisfy the constraint (58), are the witness sought.

Corollary 2 of Lin2-Xor-WEE lemma:
Under the conditions of the Lin2-Xor-WEE lemma, for the element H_1 sent in the first message of its protocol, for $\mathcal{R}_{Z,H_1} = \{((Z, H_1), ((x, y), (u, v))) \mid (Z = xP_1 + yQ_1 \wedge H_1 = uP_1 + vQ_1) \oplus (Z = xP_2 + yQ_2 \wedge H_1 = uP_2 + vQ_2)\}$, there is a witness-extended emulation algorithm for the lemma protocol (Table 5), which is capable of extracting witness for the relation $\mathcal{R}_{Z,H_1}$.

Proof: In the course of proving the Lin2-Xor-WEE lemma, we have already found the sought witness $((x, y), (u, v))$, which is calculated by the formulae (61), (59), (42), and bounded by the equality (35) and the condition (60).

Corollary 3 of Lin2-Xor-WEE lemma:
Under the conditions of the Lin2-Xor-WEE lemma, for the element H_1, challenges (c_{11}, c_{13}) and reply r_1 sent in its protocol, for

$$\mathcal{R}_{Z,H_1,c_{11},c_{13},r_1} = \left\{ \begin{array}{c} ((Z, H_1, c_{11}, c_{13}, r_1), \\ ((x, y), (u, v), w)) \end{array} \middle| \left(\begin{array}{ll} Z = xP_1 + yQ_1 & \wedge \\ H_1 = uP_1 + vQ_1 & \wedge \\ Z + r_1H_1 = w(P_1 + c_{11}Q_1) & \end{array} \right) \oplus \left(\begin{array}{ll} Z = xP_2 + yQ_2 & \wedge \\ H_1 = uP_2 + vQ_2 & \wedge \\ Z + r_1H_1 = w(P_2 + c_{13}Q_2) & \end{array} \right) \right\},$$

there is a witness-extended emulation algorithm for the lemma protocol (Table 5), which is capable of extracting witness for the relation $\mathcal{R}_{Z,H_1,c_{11},c_{13},r_1}$.

Proof: The Corollary 2 gives the parts $(x, y), (u, v)$ of the sought witness. Having the elements Z and H_1 expressed as $Z = xP_1 + yQ_1$ and $H_1 = uP_1 + vQ_1$, from the equality (35), according to the condition (60), the w part of the sought witness is equal to the known scalar a. In the remaining case, if the elements Z and H_1 are expressed as $Z = xP_2 + yQ_2$ and $H_1 = uP_2 + vQ_2$, then the part w is equal to the known scalar b from the equality (35). Thus, the witness for the relation $\mathcal{R}_{Z,H_1,c_{11},c_{13},r_1}$ is found.

5 LIN2-SELECTOR LEMMA

5.1 PRELIMINARY DEFINITIONS AND PROPERTIES

5.1.1 RSUM

Let's rewrite the sum $R = P_1 + c_{11}Q_1 + c_2P_2 + c_2c_{13}Q_2$, which we considered in the Lin2-Xor lemma, in the form of the following tree structure (see Figure 1), where we renamed P_1, Q_1, P_2, Q_2 into X_0, X_1, X_2, X_3. Informally, this tree structure is evaluated to R recursively, each node performs summation and each arrow performs multiplication by its tag. If all arrow tags are known, then R represents a multi-exponent sum of four summands.

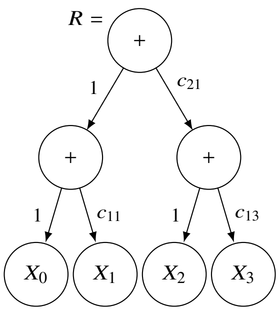

Figure 1: Rsum for four elements.

Generalizing this tree structure, we have, for example, for a set of sixteen elements $\left[X_j\right]_{j=0}^{15}$ a tree structure of the form as in Figure 2, that is, the sum $R = X_0 + c_{11}X_1 + c_{21}X_2 + c_{21}c_{13}X_3 + c_{31}X_4 + c_{31}c_{11}X_5 + c_{31}c_{23}X_6 + c_{31}c_{23}c_{13}X_7 + c_{41}X_8 + c_{41}c_{11}X_9 + c_{41}c_{21}X_{(10)} + c_{41}c_{21}c_{13}X_{(11)} + c_{41}c_{33}X_{(12)} + c_{41}c_{33}c_{11}X_{(13)} + c_{41}c_{33}c_{23}X_{(14)} + c_{41}c_{33}c_{23}c_{13}X_{(15)}$.

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

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

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

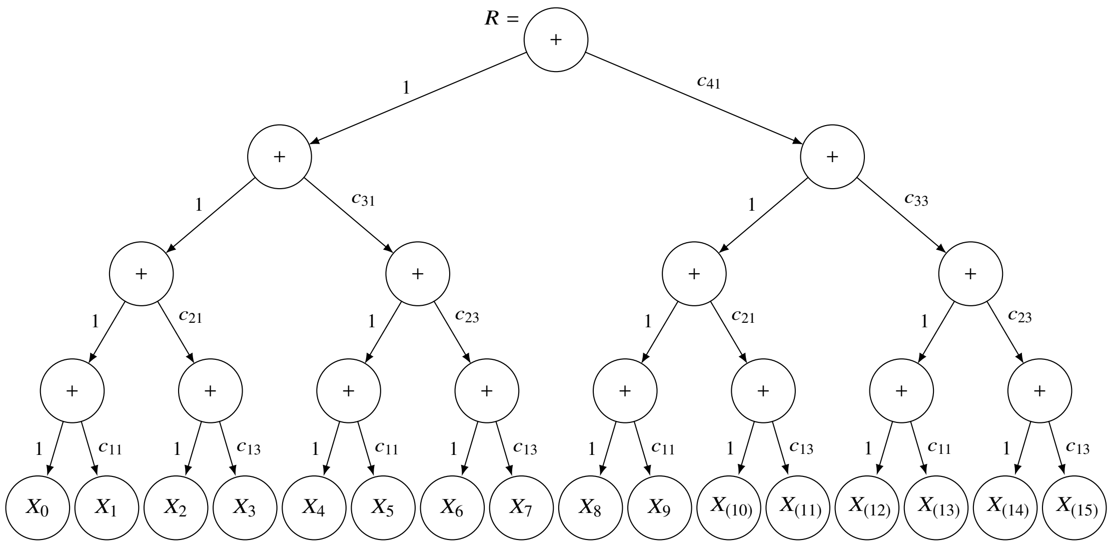

Figure 2: Rsum for sixteen elements.

Informally, for $n > 1$, Rsum $\left(n, N, \left[X_j\right]_{j=0}^{N-1}, [(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_{n1}\right)$ is a weighted sum of its left and right subtrees with the weights 1 and c_{n1}, respectively. The subtrees are the weighted sums of their left and right subtrees, and so on. For $n = 1$, the Rsum's are leaves and are calculated directly as weighted sums of two elements, with the weights $1, c_{11}$ or $1, c_{13}$.

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

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

5.2 LIN2-SELECTOR LEMMA

Lin2-Selector lemma:
For any $n > 1$ and $N = 2^n$, for any vector of non-zero fixed elements $\left[X_j\right]_{j=0}^{N-1}$ such that ort $\left(\left[X_j\right]_{j=0}^{N-1}\right)$ holds, for any non-zero fixed element Z, for a vector of n non-zero elements $[H_i]_{i=1}^{n}$ where H_1 is fixed, and for a vector of non-zero scalars $[r_i]_{i=1}^{n}$, the protocol in Table 6 is an evidence of $Z = \text{lin}\left(X_{(2s)}, X_{(2s+1)}\right)$ for some known to Prover $s \in [0, N/2 - 1]$.

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

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

Since

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

Verifier has an evidence of

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

in the last step of the protocol.

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

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

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

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

Table 6: Lin2-Selector lemma protocol.

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

For $n = (m+1) > 2$ and $N = 2^n = 2(2^m) = 2M$, for any vector of non-zero fixed elements $\left[X_j\right]_{j=0}^{2M-1}$, such that $\operatorname{ort}\left(\left[X_j\right]_{j=0}^{2M-1}\right)$ holds, any non-zero fixed element Z, a vector of $(m+1)$ non-zero elements $[H_i]_{i=1}^{m+1}$ where H_1 is fixed, and a vector of non-zero scalars $[r_i]_{i=1}^{m+1}$, the following protocol (Table 7) is an evidence of $Z = \operatorname{lin}\left(X_{(2s)}, X_{(2s+1)}\right)$, $s \in [0, M-1]$:

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

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

where:

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

Table 7: Lin2-Selector lemma protocol for $n = (m+1)$.

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

By the Rsum property,

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

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 $\text{ort}\left([X_j]_{j=0}^{2M-1}\right)$ by the premise, we have $\text{ort}(Y_0, Y_1, Y_2, Y_3)$ by the OrtDisjunction lemma. Thus, the evidence in the last step of the protocol rewrites as follows:

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

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

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

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

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

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

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

then it accepts that

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

5.3 WITNESS EXTRACTION

Now we will provide a WEE counterpart for the Lin2-Selector lemma, as we did in Section 4.3.2 for the Lin2-Xor lemma.

Lin2-Selector-WEE lemma:
For any $n > 1$ and $N = 2^n$, for any vector of non-zero fixed elements $\left[X_j\right]_{j=0}^{N-1}$ such that $\text{ort}\left(\left[X_j\right]_{j=0}^{N-1}\right)$ holds, for any non-zero fixed element Z, for a vector of n non-zero elements $[H_i]_{i=1}^{n}$ where H_1 is fixed, for a vector of non-zero scalars $[r_i]_{i=1}^{n}$, for the relation $\mathcal{R} = \{(Z, (x, y, s)) \mid Z = xX_{(2s)} + yX_{(2s+1)}, s \in [0, N/2 - 1]\}$, the protocol in Table 6 has witness-extended emulation, provided that the evidence for the Rsum in the last step of the protocol has witness-extended emulation.

Proof: The element H_1 is implied as the first message from Prover to Verifier in the protocol (Table 6). For this protocol, we will prove the existence of a PPT emulator that satisfies the definition of witness-extended emulation. The proof is by induction for each n starting from 2, where $n = \log_2(N)$.

For the induction base case, $n = 2$, this lemma premise and protocol meet the premise and protocol of the Lin2-Xor-WEE lemma. By the Lin2-Xor-WEE lemma conclusion, there exists a PPT emulator such that the sought scalars x and y can be efficiently calculated with it. The index s of the witness (x, y, s) is found by simply checking which pair of elements from $[X_j]_{j=0}^{N-1}$ matches the condition $Z = xX_{(2s)} + yX_{(2s+1)}$. Thus, the induction base case is proven.

The induction hypothesis is that the lemma holds for $m > 1$. Let's prove the lemma for $n = (m + 1)$. When the lemma protocol is successfully completed for $n = (m + 1)$, Verifier has an Rsum tree structure R built over the challenges and set $[X_j]_{j=0}^{2M-1}$ as

$$R = \text{Rsum}(m + 1, 2M, \left[X_j\right]_{j=0}^{2M-1}, [(c_{i1}, c_{i3})]_{i=1}^{m}, c_{m+1}), \quad \text{where } M = 2^m \text{ and } \text{ort}([X_j]_{j=0}^{2M-1}). \tag{62}$$

At depth 2 from the root, this tree structure has four sub-trees Y_0, Y_1, Y_2, Y_3 such that

$$R = Y_0 + c_{m1}Y_1 + c_{m+1}Y_2 + c_{m+1}c_{m3}Y_3 , \tag{63}$$

evaluated as

$$\begin{cases} Y_0 = \text{Rsum}\left(m - 1, M/2, \left[X_j\right]_{j=0}^{M/2-1}, [(c_{i1}, c_{i3})]_{i=1}^{m-2}, c_{(m-1),1}\right) \\ Y_1 = \text{Rsum}\left(m - 1, M/2, \left[X_j\right]_{j=M/2}^{M-1}, [(c_{i1}, c_{i3})]_{i=1}^{m-2}, c_{(m-1),3}\right) \\ Y_2 = \text{Rsum}\left(m - 1, M/2, \left[X_j\right]_{j=M}^{3M/2-1}, [(c_{i1}, c_{i3})]_{i=1}^{m-2}, c_{(m-1),1}\right) \\ Y_3 = \text{Rsum}\left(m - 1, M/2, \left[X_j\right]_{j=3M/2}^{2M-1}, [(c_{i1}, c_{i3})]_{i=1}^{m-2}, c_{(m-1),3}\right) \end{cases} \tag{64}$$

and, by the OrtDisjunction lemma, satisfying $\text{ort}(Y_0, Y_1, Y_2, Y_3)$. Thus, we see that during the execution of the protocol (Table 6), starting from the moment the challenge pair (c_{m1}, c_{m3}) is generated, the rest of the protocol satisfies the premise and the protocol (Table 5) of the Lin2-Xor-WEE lemma.

Therefore, according to the Corollary 3 of Lin2-Xor-WEE lemma, there is a PPT emulator that extracts scalar w such that

$$\left(Z + \sum_{i=1\ldots m} r_i H_i = w(Y_0 + c_{m1}Y_1)\right) \oplus \left(Z + \sum_{i=1\ldots m} r_i H_i = w(Y_2 + c_{m3}Y_3)\right) . \tag{65}$$

The elements $(Y_0 + c_{m1}Y_1)$ and $(Y_2 + c_{m3}Y_3)$, taking into account the formulae (63) and (64), are the roots of the left and right sub-trees of the Rsum R. Thus, depending on which of them the witness w belongs to according to the statement (65), this lemma protocol (Table 6) for $n = (m + 1)$ meets this lemma premise and protocol for $n = m$. Hence, by the induction hypothesis, since this lemma is assumed proven for $n = m$, there is a PPT emulator that finds a witness (x, y, s) for Z in the statement (65), that is, a witness for the following relation

$$\mathcal{R}' = \left\{ \ (\ Z, (x, y, s)\ ) \ \middle| \left(\begin{array}{l} Z = xX_{(2s)} + yX_{(2s+1)} , \\ s \in [0, M/2 - 1] \end{array} \right) \oplus \left(\begin{array}{l} Z = xX_{(M+2s)} + yX_{(M+2s+1)} , \\ s \in [0, M/2 - 1] \end{array} \right) \right\}. \tag{66}$$

The sought witness for the relation $\mathcal{R}$ is obtained from the witness for the relation $\mathcal{R}'$ (66) in one step by simple conversion of the index s. Thus, we have proven this lemma for $n = (m + 1)$ and, by induction, the lemma is proven.

6 L2S MEMBERSHIP PROOF

Here we construct a proof of membership (PoM) protocol called **L2S**. In this protocol, Verifier is fed with an element Z, and, upon successful completion of all steps of the protocol, Verifier is convinced that Z is a commitment to a pair of elements from a publicly known set of element pairs, such that Prover knows an opening for Z. In other words, Verifier is convinced that its input Z is, in fact, a member pair from the public set, enclosed into a commitment.

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

6.1 COM2 COMMITMENT

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

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

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

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

By the OrtUniqueRepresentation lemma, if Z has a (k_0, k_1, s) opening over $\vec{X}$, then the opening (k_0, k_1, s) is unique. Hence, according to definition from [14], the Com2 commitment is strongly binding.

We call a Com2 commitment as a commitment to a member-pair. A set of member-pairs $\left[X_j\right]_{j=0}^{N-1}$, $N = 2^n$, is called a decoy set.

6.2 L2S MEMBERSHIP PROOF PROTOCOL

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

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

where:

- **DecoySetGen** (n), where $n > 1$, is an arbitrary function that returns an element vector $\vec{X} = \left[X_j\right]_{j=0}^{N-1}$ of $N = 2^n$ elements such that ort$(\vec{X})$ holds. Each element in the generated $\vec{X}$ has a distribution that is independent of the distributions of other elements in the same $\vec{X}$ and is indistinguishable from the uniform randomness. Two vectors generated by **DecoySetGen** may have a non-empty intersection. For any **DecoySetGen** implementation choice, the returned vector $\vec{X}$ orthogonality, independence of the element distributions and their uniform randomness are to be guaranteed.
- **ComGen**$(\vec{X})$ is an arbitrary function that returns a pair $((k_0, k_1, s), Z)$, where k_0 is non-zero and chosen uniformly at random, k_1 is arbitrary, $s \in [0, N/2 - 1]$, and $Z = \text{Com2}(k_0, k_1, s, \vec{X})$. For any **ComGen** implementation choice, the independence and random uniformity of the k_0 distribution together with the conditions $Z = \text{Com2}(k_0, k_1, s, \vec{X})$ and $k_0 \neq 0$ are to be guaranteed.
- **InteractionProcedure** is shown in Table 8. It starts with Prover having an opening (k_0, k_1, s), and Verifier having a commitment Z. On completion of **InteractionProcedure**, Verifier has a tuple

$$\left([(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n, Z, [(r_i, H_i)]_{i=1}^{n}, c, T, t\right) \tag{67}$$

  that contains Z together with all the challenges and replies occurred during the Prover and Verifier interaction.
- **Verif** function is shown in Table 9. It accepts a tuple that Verifier has upon completion of **InteractionProcedure** along with a decoy set from **DecoySetGen**. It returns 1 or 0, which means the verification succeeded or failed.

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

- A decoy set $\vec{X}$ is generated at both Prover's and Verifier's sides, using same **L2S.DecoySetGen**.
- Prover gets an opening (k_0, k_1, s) from **L2S.ComGen**. At the same time, Verifier gets some element Z.
- All steps of **L2S.InteractionProcedure** are performed between the Prover and Verifier. On completion of **L2S.InteractionProcedure** Verifier has the tuple (67).
- Verifier calls **L2S.Verif** for the decoy set and tuple obtained above. Iff it returns 1, then the **L2S** protocol is completed successfully. We will prove below that the successful completion means $Z = \text{Com2}(k_0, k_1, s, \vec{X})$.

Table 8: **L2S.InteractionProcedure**.

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

Table 9: **L2S.Verif** function.

```
Input: $n$, $\left[X_j\right]_{j=0}^{N-1}$, $\left([(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n, Z, [(r_i, H_i)]_{i=1}^{n}, c, T, t\right)$, where $N = 2^n, n > 1$
$S = Z$
For $i = 1 \dots n$:
    If ($r_i == 0$ or $H_i == 0$) then return 0
    $S = S + r_i H_i$
    If $S == 0$ then return 0
$W = S$
$R = \text{Rsum}\left(n, N, \left[X_j\right]_{j=0}^{N-1}, [(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n\right)$
If $(tW + cR) == T$ then return 1
Else return 0.
```

Note the *InvertLastBit* function, which is used in the **L2S.InteractionProcedure** implementation (Table 8), takes an unsigned integer and returns this integer with inverted least significant bit in its binary representation. That is, it is defined as

$$\textit{InvertLastBit}\,(i) = (2\,(i//2) + (i+1)\,\%2)\,, \text{ where the } // \text{ and } \% \text{ are the quotient and remainder operators.}$$

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

6.2.1 PROOF OF THE RELATION BETWEEN R AND W

Now, looking at the **L2S.InteractionProcedure** implementation in Table 8, we show that

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

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

On the Prover's side of **L2S.InteractionProcedure**, at the beginning, the expression

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

lets all Y_j's be X_j's.
Next, down the protocol execution flow, when $i = 1$, the expression

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

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

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

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

When $i = 2$, the expression

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

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

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

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

When $i = 3$, the expression

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

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

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

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

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

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

When $i = 1$, Prover sends to Verifier

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

so that $(Z + r_1H_1) = w\,\mathrm{Rsum}\left(1, 2, [X_t]_{t=2s}^{2s+1}, [], c_{1,((2s+1)\%4)}\right)$.
Next, Prover reshuffles q, sets $h = \mathit{InvertLastBit}(s)$ and sends

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

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

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

so that

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

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

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

When $i = 3$, Prover sends

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

so that

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

And so on, until $i = n$ and

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

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

6.2.2 PROOF THAT CORRECT OPENING IMPLIES L2S.VERIF RETURN 1

The (T, c, t) part of the **L2S.Verif** implementation (Table 9) input is the Schnorr identification scheme [23] initial message, challenge and reply for the relation $R = xW$.

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

As proven in 6.2.1, $\mathrm{Rsum}\left(n, N, \left[X_j\right]_{j=0}^{N-1}, [(1, c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n\right) = xW$. Hence, on the Prover's side xW is equal to R used in **L2S.Verif**. As the Schnorr identification scheme [23] is complete, this implies $(tW + cR) == T$.

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

6.3 LS2 PROTOCOL PROPERTIES

6.3.1 COMPLETENESS

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

6.3.2 SOUNDNESS

L2S.InteractionProcedure (Table 8) together with the subsequent call of the **L2S.Verif** function (Table 9) matches the Lin2-Selector lemma protocol (Table 6). Namely, the **L2S.InteractionProcedure** and **L2S.Verif** populate the Lin2-Selector lemma pure protocol with a concrete implementation of Prover's behavior. Therefore, as such, the **LS2** protocol satisfies the criterion of the Lin2-Selector lemma protocol for applying the Lin2-Selector lemma.

As shown in 6.2.2, if **L2S.Verif** returns 1, then $(tW + cR) == T$, and, since the Schnorr identification scheme [23] is sound, Verifier has an evidence of $W \sim R$, that is, an evidence of

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

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

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

6.3.3 WITNESS-EXTENDED EMULATION

In 6.3.2 we showed that the Lin2-Selector lemma applies to the **LS2** protocol. In addition to the requirements imposed by the Lin2-Selector lemma on a protocol, the Lin2-Selector-WEE lemma from 5.3 to be applicable requires a witness-extended emulator to exist for the evidence at the last step of the protocol (Table 6). That is, in relation to the **LS2** protocol, it requires a witness-extended emulator to exist for the evidence of $W \sim R$.

The evidence of $W \sim R$ is implemented with the Schnorr identification scheme, more on this in 6.2.2. Since the Schnorr identification scheme is special sound, a witness-extended emulator exists for the evidence of $W \sim R$ according to the definitions of the witness-extended emulation and special soundness (the latter is a case of the former). Hence, the Lin2-Selector-WEE lemma applies to the **LS2** protocol.

Thus, by the Lin2-Selector-WEE lemma, the **LS2** protocol has witness-extended emulation.

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

The **LS2** protocol Prover-Verifier public transcript is the following tuple, copying (67) here

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

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

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

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

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

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

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

All the elements $Z, r_1H_1, \dots, r_iH_i$ are obfuscated by the multiplier w. The multiplier w is private and uniformly random, as $w = k_0$, where k_0 is uniformly random by the definition of **L2S.ComGen**. By the definition of Rsum, each r_iH_i is a linear combination of the elements from the $\left[X_j\right]_{j=0}^{N-1}$ with efficiently computable scalar coefficients. Moreover, all r_iH_i's depend on the different non-intersecting subsets of the $\left[X_j\right]_{j=0}^{N-1}$.

Using the terms introduced in [5], the r_iH_i's and Z are linearly independent degree 2 polynomials of a private set of the independent and random uniform scalars

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

The coefficients of these polynomials are efficiently computable from the $[(c_{i1}, c_{i3})]_{i=1}^{n-1}$, c_n, and k_1. Thus, reducing the question of the r_iH_i's distributions to the (P, Q)-DDH problem [5], we have

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

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

As the DDH assumption implies (P, Q)-DDH [5] for our polynomials in the above sets P and Q, we have all the r_iH_i's and Z distributed independently and uniformly at random under DDH. We have proven this for any conversation transcript between honest Prover and Verifier over any fixed decoy set $\left[X_j\right]_{j=0}^{N-1}$ generated by **L2S.DecoySetGen**. For readability, we omit the word 'indistinguishable', reserving it for the distributions.

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

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

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

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

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

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

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

6.3.5 SPECIAL HONEST VERIFIER ZERO-KNOWLEDGE

We will show the **L2S** protocol is sHVZK following definition from [6]. Having the random independence property proven for the transcript items in 6.3.4, it's easy to build a simulator, that for any given challenges and for any given input Z generates a simulated transcript that Verifier accepts, and no PPT algorithm is able to distinguish it from the space of honest transcripts with the same challenges.

The simulator acts as follows:

- It takes an empty L2S transcript placeholder and puts the given input Z and challenges $[(c_{i1}, c_{i3})]_{i=1}^{n-1}$, c_n in their places.
- It independently generates random uniform scalars and puts them in the places of scalars in the placeholder.
- It independently generates random uniform scalars and puts their exponents in the places of elements in the placeholder, except for the place of element T.

- It takes the values $[(c_{i1}, c_{i3})]_{i=1}^{n-1}$, c_n, Z, $[(r_i, H_i)]_{i=1}^{n}$, c, t from the already filled in places of the placeholder, obtains $\left[X_j\right]_{j=0}^{N-1}$ by calling **L2S.DecoySetGen**, calculates

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

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

Thus, the simulated transcript is ready. Verifier accepts it, as it passes the $(tW + cR) == T$ check in **L2S.Verif**. (The other checks in **L2S.Verif** are also passed with overwhelming probability as the checked values are uniformly random.)

Suppose, there exists a PPT algorithm that distinguishes with non-negligible probability the simulated transcript from the space of honest transcripts with the same challenges. As proven in 6.3.4, the space contains the transcripts with all items having distributions indistinguishable from the distributions of the items of the simulated transcript, except for the item T. However, T is calculated the same way from the same sources for honest and for simulated transcripts, hence the algorithm is not able to distinguish the transcripts by T's. Hence, we have that the PPT algorithm is able to distinguish indistinguishable distributions, contradiction.

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

6.3.6 INDISTINGUISHABILITY OF THE MEMBER-PAIR INDEX

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

Suppose, there exists a PPT algorithm that distinguishes s with non-negligible probability from a honest Prover-Verifier conversation transcript. Applying the algorithm to all transcripts in the honest transcript space, we obtain a partitioning of the space where each partition with non-negligible probability distinguishes some information about the actual values of s in it. However, according to 6.3.4 the space entries contain only the items indistinguishable from the independent and uniform randomness, with the exclusion of the dependent items T that carry no additional information. Thus, we have the algorithm that distinguishes with non-negligible probability some information about the actual values of s from the independent and uniform randomness, that is a contradiction.

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

7 L2S PROTOCOL EXTENSIONS

7.1 RL2S PROTOCOL, SHVZK FOR NON-RANDOM INPUT

As shown in 6.3.5, **L2S** is sHVZK under DDH, provided that the scalar k_0 in Prover's input (k_0, k_1, s) has independent and uniformly random distribution. To remove this restriction and to allow the protocol to keep the sHVZK property for any input commitment distribution, including the cases when a linear relationship between different input commitments is known to an adversary, we extend **L2S** protocol with an input randomization. Of course, as the input commitments are publicly seen in the transcripts, an adversary is still able to track the known relationships between them, however, with sHVZK no adversary is able to obtain any information beyond that from the transcripts.

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

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

We define **RL2S** protocol as four procedures

$$\textbf{RL2S} = \{\textbf{DecoySetGen=L2S.DecoySetGen}, \textbf{ComGen}, \textbf{InteractionProcedure}, \textbf{Verif}\},$$

where

- **RL2S.ComGen**$(\vec{X})$ is an arbitrary function that returns a pair $((k_0, k_1, s), Z_0)$, where k_0 is arbitrary non-zero, k_1 is arbitrary, $s \in [0, N/2 - 1]$, and $Z_0 = \text{Com2}\left(k_0, k_1, s, \vec{X}\right)$. For any **ComGen** implementation choice, the conditions $k_0 \neq 0$ and $Z_0 = \text{Com2}\left(k_0, k_1, s, \vec{X}\right)$ are to be guaranteed.
- **RL2S.InteractionProcedure** is shown in Table 10. It starts with Prover having (k_0, k_1, s), $k_0 \neq 0$, and Verifier having Z_0. On completion of **RL2S.InteractionProcedure**, Verifier has two tuples: (Z_0, c_0, T_0, t_0) and $\left([(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n, Z, [(r_i, H_i)]_{i=1}^{n}, c, T, t\right)$, that contain the initial input as Z_0 and the randomized input as Z together with all the challenges and replies occurred during the Prover and Verifier interaction.

Table 10: **RL2S.InteractionProcedure**.

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

- **RL2S.Verif** function is shown in Table 11. It takes the two tuples from the **RL2S.InteractionProcedure** output together with the decoy set from **DecoySetGen** and returns 1 or 0.

Table 11: **RL2S.Verif** function.

Input: n, $\left[X_j\right]_{j=0}^{N-1}$, where $N = 2^n$, $n > 1$, (Z_0, c_0, T_0, t_0), $\left([(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n, Z, [(r_i, H_i)]_{i=1}^{n}, c, T, t\right)$
If $(t_0Z_0 + c_0Z) == T_0$ then continue Else return 0 Run **L2S.Verif** for the n, $\left[X_j\right]_{j=0}^{N-1}$, $\left([(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n, Z, [(r_i, H_i)]_{i=1}^{n}, c, T, t\right)$

The steps for the **RL2S** protocol are the same as for the **L2S** protocol.

7.1.1 RL2S PROTOCOL COMPLETENESS AND SOUNDNESS

As the Schnorr identification and the **L2S** protocols are complete and sound, the **RL2S** protocol is complete and sound.

7.1.2 RL2S PROTOCOL SHVZK

The **RL2S** protocol is sHVZK. To prove this, we repeat the same steps as those for the **L2S** sHVZK proof in 6.3.5 with the only two additions

- As the (Z_0, c_0, T_0, t_0) tuple is put at the beginning of a public **RL2S** protocol transcript, and as Z in the transcript becomes $Z = fZ_0$, it's necessary to determine the distributions of them:
  - c_0 is an independent and uniformly random honest Verifier's challenge.
  - Z has independent and random uniform distribution, as f in the equation $Z = fZ_0$ is private, independent, and uniformly random.
  - t_0 is independent and uniformly random, as it is obfuscated by the private independent and uniformly random scalar q in the formula $t_0 = q - fc_0$.
  - Z_0 is independent of the other items in the transcript, however, it is not uniformly random.
  - T_0 is not independent, it is evaluated as $T_0 = (t_0 Z_0 + c_0 Z)$ from the items (Z_0, Z, c_0, t_0).

  Thus, all T_0's can be excluded from consideration, as they carry no information. We get to conclusion, that an **RL2S** transcript contains two dependent items: T_0 and T, that are evaluated from the other items. It contains the input commitments as Z_0, and there is no item, except for T_0, distinguishably dependent on Z_0 in the transcript. All the other items are independent and uniformly random.

- **RL2S** simulator puts the input commitment in the place of Z_0 and fills in all the other places, except for the ones of T_0 and T, with the independent and uniformly random values. It puts the evaluated values $(t_0 Z_0 + c_0 Z)$ and $(tW + cR)$ in the places of T_0 and T, respectively.

7.1.3 RL2S PROTOCOL WITNESS-EXTENDED EMULATION

The **RL2S** protocol is the Schnorr identification protocol followed by the **L2S** protocol. Since both Schnorr identification and **L2S** protocols have witness-extended emulators, a witness-extended emulator for the **RL2S** protocol can be obtained by simply invoking the emulator for **L2S** and then the emulator for Schnorr identification protocols. Thus, the **RL2S** protocol has witness-extended emulation.

7.2 MRL2S PROTOCOL

A parallel version of the **RL2S** protocol is a protocol that runs multiple instances of **RL2S.InteractionProcedure** in parallel and thus proves membership for multiple commitments at once. We call it **MRL2S** protocol and define as follows

MRL2S = {**DecoySetGen=L2S.DecoySetGen**, **ComGen=RL2S.ComGen**, **MapInteractionProcedure**, **JoinVerif**},

where

- **MRL2S.MapInteractionProcedure** is shown in Table 12. It starts with Prover having L openings $\left[\left(k_0^p, k_1^p, s^p\right) | k_0^p \neq 0\right]_{p=1}^{L}$ and Verifier having L commitments $\left[Z_0^p\right]_{p=1}^{L}$. On completion of **MRL2S.InteractionProcedure**, Verifier has L tuples

$$\left(\left(Z_0^p, c_0, T_0^p, t_0^p\right), \left([(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n, Z^p, \left[\left(r_i^p, H_i^p\right)\right]_{i=1}^{n}, c, T^p, t^p\right)\right)_{p=1}^{L}, \tag{68}$$

  which contain the outputs of L concurrent runs of **MRL2S.InteractionProcedure** with the same decoy set and common challenges.

- **MRL2S.JoinVerif** function is shown in Table 13. It takes the L tuples from **MRL2S.MapInteractionProcedure** together with the decoy set from **DecoySetGen** and returns 1 or 0.

  MRL2S.JoinVerif performs L verifications in parallel. As all the Rsum's R inside the nested **RL2S.Verif.L2S.Verif** calls are the same, **MRL2S.JoinVerif** performs their calculation only once, at the beginning, and uses the calculated value

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

Table 12: **MRL2S.MapInteractionProcedure**.

Prover and Verifier common parameters: • L • $n, N = 2^n$, where $n > 1$	
Prover: $\left[\left(k_0^p, k_1^p, s^p\right) \mid k_0^p \neq 0\right]_{p=1}^{L}$	Verifier: $\left[Z_0^p\right]_{p=1}^{L}$
For each $p \in [1, L]$: run **RL2S.InteractionProcedure** using n, $\left(k_0^p, k_1^p, s^p\right)$ as arguments for Prover, and n, Z_0^p as arguments for Verifier. All the parallel **RL2S.InteractionProcedure** instances share the same decoy set $\left[X_j\right]_{j=0}^{N-1}$ = **DecoySetGen** (n) and same Verifier's challenges $c_0, [(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n, c$	
	Verifier has L tuples: $\left[\left(\left(Z_0^p, c_0, T_0^p, t_0^p\right), \left([(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n, Z^p, \left[\left(r_i^p, H_i^p\right)\right]_{i=1}^{n}, c, T^p, t^p\right)\right)\right]_{p=1}^{L}$

Table 13: **MRL2S.JoinVerif** function.

Input: $L, n, \left[X_j\right]_{j=0}^{N-1}$, where $N = 2^n, n > 1$, $\left(\left(Z_0^p, c_0, T_0^p, t_0^p\right), \left([(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n, Z^p, \left[\left(r_i^p, H_i^p\right)\right]_{i=1}^{n}, c, T^p, t^p\right)\right)_{p=1}^{L}$
$R = \text{Rsum}\left(n, N, \left[X_j\right]_{j=0}^{N-1}, [(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n\right)$ For each $p \in [1, L]$: run **RL2S.Verif** using n, $\left[X_j\right]_{j=0}^{N-1}$ and $\left(Z_0^p, c_0, T_0^p, t_0^p\right), \left([(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n, Z^p, \left[\left(r_i^p, H_i^p\right)\right]_{i=1}^{n}, c, T^p, t^p\right)$ as arguments.   Inside each **RL2S.Verif** call, within nested **L2S.Verif** call, use the calculated above R for the **RL2S.Verif.L2S.Verif**.R Return 0 if one of the **RL2S.Verif** calls returns 0. Otherwise, return 1.

The steps for the **MRL2S** protocol are identical to the steps of the **RL2S** protocol, with the only difference in that the parallel procedure versions are used instead of the sequential ones:

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

7.2.1 MRL2S PROTOCOL COMPLETENESS, SOUNDNESS, AND WITNESS-EXTENDED EMULATION

The **MRL2S** protocol completeness and soundness immediately follow from the completeness and soundness of the **RL2S** protocol.

The **MRL2S** protocol also has witness-extended emulation, its emulator calls the L (polynomial number) nested **RL2S** protocol emulators, which synchronously rewind to the same points in the L transcript trees. To summarize, here is the polynomial time relation that the **MRL2S** protocol emulator finds witness for

$$\mathcal{R} = \bigcup_{p=1}^{L} \{ \; (Z_0^p, (k_0^p, k_1^p, s^p)) \mid Z_0^p = k_0^p X_{(2s^p)} + k_1^p X_{(2s^p+1)} \,, \; s \in [0, N/2 - 1] \; \} \tag{69}$$

7.2.2 MRL2S PROTOCOL SHVZK

The **MRL2S** protocol is sHVZK. To prove this, we repeat the same steps as for the proof of **RL2S** sHVZK in 7.1.2 and, therefore, as for the proof of **L2S** sHVZK in 6.3.5, with the only addition below.

The space of honest **MRL2S** transcripts is the space of honest **RL2S** transcripts partitioned by the **MRL2S** proofs. Each partition contains L **RL2S** transcripts with equal challenges, that were generated during the corresponding proof with L inputs. For each partition, all items of its L transcripts, omitting the challenges and items Z_0,

T_0, T discussed above (in 7.1.2 and 6.3.4) as revealing no information, are distributed independently and uniformly at random. Independence here is meant as the total independence from the other items in own transcript, own partition, and other partitions as well. Hence, the honest **MRL2S** transcript space doesn't reveal any information other than the information accessible from the input commitments and partitioning per se.

A simulator for the **MRL2S** protocol runs L **RL2S** protocol simulators in parallel and, upon completion of all of them, the simulated transcript contains L **RL2S** simulated transcripts that are indistinguishable from honest ones. Thus, a simulated **MRL2S** transcript is indistinguishable from an honest **MRL2S** transcript.

7.2.3 MRL2S PROTOCOL COMPLEXITIES

Recalling the **MRL2S** transcript (68),

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

where all data, except for the initial elements $\{Z_0^p\}_{p=1}^L$ and challenges, are regarded as transmitted from Prover to Verifier, the amount of transmitted data is shown in Table 14.

Table 14: **MRL2S** transmitted data amount.

	$\mathbb{G}$	$\mathbb{F}$
MRL2S	$L\,(n+3)$	$L\,(n+2)$

The $R = \text{Rsum}(n, N, \left[X_j\right]_{j=0}^{N-1}, [(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n)$ 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 expanded so that all the scalar coefficients for the elements from $\left[X_j\right]_{j=0}^{N-1}$ would be calculated as scalar-scalar multiplications and, after that, a single multi-exponentiation of the elements from $\left[X_j\right]_{j=0}^{N-1}$ to their respective coefficients would be performed. **MRL2S** verification complexity is shown in Table 15, where $N = 2^n$:

Table 15: **MRL2S** verification complexity.

	multi-exp(N)	single-exp
MRL2S	1	$nL + 3L + 1$

8 MRL2S-BASED NON-INTERACTIVE PROOF OF MEMBERSHIP

Having an interactive public coin protocol, it's possible to turn it into a non-interactive scheme using the Fiat-Shamir heuristic in ROM [10, 21, 2]. We create a non-interactive zero-knowledge PoM scheme on the base of **MRL2S**. Overall, the idea behind our PoM is that we do not allow the odd elements of the **MRL2S** decoy set to participate in the commitment Z_0, and thus Z_0 becomes an element at even position in the decoy set multiplied by some secret, not necessarily random, scalar.

As a random oracle, to generate challenges, we use hash function $\mathbf{H_{scalar}}(\dots)$ defined below. The **MRL2S** protocol requires an orthogonal decoy set with the element distributions indistinguishable from independent uniform randomness, so we also use 'hash-to-curve' function $\mathbf{H_{point}}(\dots)$.

8.1 PRELIMINARIES

8.1.1 ELLIPTIC CURVE POINTS AND ELEMENTS

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

8.1.2 ANY TO SCALAR HASH FUNCTION $\mathbf{H_{SCALAR}}(\dots)$

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

8.1.3 ANY TO POINT HASH FUNCTION $\mathbf{H_{POINT}}(\dots)$

We call $\mathbf{H_{point}}(\dots)$ an ideal hash function that accepts any number of ordered arguments of any type, i.e. the arguments are strings, scalars in $\mathbb{F}$, points in $\mathbb{G}$. It returns a point in $\mathbb{G}$.

8.1.4 IDEAL HASH FUNCTIONS AND RANDOM ORACLES

We use the term 'ideal hash function' as a shorthand for the term 'cryptographic hash function that is indifferentiable from a random oracle'. For the $\mathbf{H_{scalar}}$ it can be, for instance, SHA-3 [8]. For the $\mathbf{H_{point}}$ it can be, for instance, one of the functions described in [16, 9, 11].

8.1.5 RESERVED INTEGER NAMES AND CONSTANTS

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

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

8.1.6 DECOY VECTOR AS A VECTOR OF PAIRS

The procedure **MRL2S.DecoySetGen** in 7.2 returns the decoy vector $[X_j]_{j=0}^{N-1}$. We reshape this vector to be a vector of pairs $[(P_j, Q_j)]_{j=0}^{N/2-1}$. Thus, the vector $[X_j]_{j=0}^{N-1}$ becomes a flattened view of the $[(P_j, Q_j)]_{j=0}^{N/2-1}$, where for any $s \in [0, N/2-1]$: $P_s = X_{2s}, Q_s = X_{2s+1}$. We write $[X_j]_{j=0}^{N-1} = \mathit{Flatten}([(P_j, Q_j)]_{j=0}^{N/2-1})$ for this.

8.1.7 PREDEFINED SET OF ORTHOGONAL GENARATORS

Let $[G_i]_{i=0}^{D-1}$ be a predefined set of orthogonal generators. We construct it with $\mathbf{H_{point}}$ as a separate family of hash points $[G_i]_{i=0}^{D-1} = [\mathbf{H_{point}}(0, i)]_{i=0}^{D-1}$, and use it for the odd elements of the decoy sets.

8.2 MRL2SPOM NIZK POM SCHEME

The abbreviation **MRL2SPoM** stands for the **MRL2S**-based proof of membership scheme, that is, the above non-interactive proof that we create for the following relation

$$\mathcal{R} = \bigcup_{p=1}^{L} \left\{ \, (Z_0^p, (v^p, s^p)) \mid Z_0^p = v^p P_{s^p} \ , \ s^p \in [0, N/2-1] \, \right\} \tag{70}$$

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

$$\sigma = \left(T_0, Z, t_0, [(r_i, H_i)]_{i=1}^{n}, T, t\right) \tag{71}$$

In fact, this data structure is a part of **MRL2S** transcript that is interactively transmitted from Prover to Verifier. For any L, the proof data transmitted from Prover to Verifier is L instances of σ, that is, $[\sigma^p]_{p=1}^{L}$. Note the input commitment Z_0 is not transmitted since it is known in advance by both parties.

The **MRL2SPoM** scheme is six procedures:

$$\mathbf{MRL2SPoM} = \{\mathbf{GetPredefinedGenerators}, \mathbf{SetGen}, \mathbf{MembersGen}, \mathbf{GetDecoySet}, \mathbf{Prove}, \mathbf{Verif}\},$$

where:

- **MRL2SPoM.GetPredefinedGenerators** returns the vector of orthogonal generators $[G_i]_{i=0}^{D-1}$ such that they have uniformly random and independent distributions. Implementation is shown in Listing 1.

Listing 1: **MRL2SPoM.GetPredefinedGenerators** initial implementation.

```
Input:    none
Output:   $[G_i]_{i=0}^{D-1}$      ---orthogonal generators
Procedure:
    $[G_i]_{i=0}^{D-1} = [\mathbf{H_{point}}(\text{“Predefined generator family”}, G, i)]_{i=0}^{D-1}$
    Return $[G_i]_{i=0}^{D-1}$
```

- **MRL2SPoM.SetGen** returns an orthogonal set of points $[P_j]_{j=0}^{N/2-1}$ such that all points in it have uniformly random and independent distributions. Also, it's guaranteed that ort($[P_j]_{j=0}^{N/2-1} \cup [G_i]_{i=0}^{D-1}$) holds for the returned points.

- **MRL2SPoM.MembersGen** returns a vector of points $[Z_0^p]_{p=1}^L$ such that each point in it will be proven a member of the set $[P_j]_{j=0}^{N/2-1}$ multiplied by some known to Prover scalar.
- **MRL2SPoM.GetDecoySet** takes a hint point F such that $F \mathrel{!}= \text{lin}(\ [Z_0^p]_{p=1}^L \cup [P_j]_{j=0}^{N/2-1} \cup [G_i]_{i=0}^{D-1}\ )$, and returns the decoy set $[X_j]_{j=0}^{N-1}$ for use in the proof. Implementation is shown in Listing 2.

Listing 2: **MRL2SPoM.GetDecoySet** implementation.

```
Input:   $F$               ---hint point
Output:  $[X_j]_{j=0}^{N-1}$       ---decoy set
Procedure:
    $[G_i]_{i=0}^{D-1} = \texttt{GetPredefinedGenerators}()$
    $[P_j]_{j=0}^{N/2-1} = \texttt{SetGen}()$
    $[Q_j]_{j=0}^{N/2-1} = [G_j + F]_{j=0}^{N/2-1}$
    $[X_j]_{j=0}^{N-1} = \texttt{Flatten}([(P_j, Q_j)]_{j=0}^{N/2-1})$
    Return $[X_j]_{j=0}^{N-1}$
```

- **MRL2SPoM.Prove** takes a vector of private keys $[(v^p, s^p)]_{p=1}^L$ together with a public scalar seed e, and returns the vector $[\sigma^p]_{p=1}^L$ or 0 on error. **Prove** is **MRL2S.MapInteractionProcedure** translated to the non-interactive setting. Specification is in Listing 3.

Listing 3: **MRL2SPoM.Prove** specification.

```
Input:  $e$                    --scalar seed
        $[(v^p, s^p)]_{p=1}^L$         --private keys
Output: $[\sigma^p]_{p=1}^L$ or $\emptyset$      --proof, vector of $\sigma$'s on success,
                             --$\emptyset$ on failure
Procedure:
  • Let $[P_j]_{j=0}^{N/2-1} = \texttt{SetGen}()$
  • Let $[Z_0^p]_{p=1}^L = \texttt{MembersGen}()$
  • Let $F = \mathtt{H_{point}}([P_j]_{j=0}^{N/2-1}, [Z_0^p]_{p=1}^L)$. Thus, $F \mathrel{!}= \text{lin}(\ [Z_0^p]_{p=1}^L \cup [P_j]_{j=0}^{N/2-1} \cup [G_i]_{i=0}^{D-1}\ )$
                                     holds.
  • Let $[X_j]_{j=0}^{N-1} = \texttt{GetDecoySet}(F)$
  • For $p = 1 \ldots L$:                  Ensure the private keys correspond
      If $Z_0^p \neq v^p X_{2s^p}$ then Return $\emptyset$.  to the member set elements.
  • Let $[(k_0^p, k_1^p, s^p)]_{p=1}^L = [(v^p, 0, s^p)]_{p=1}^L$
  • Run all $L$ RL2S.InteractionProcedure's in parallel with $[(k_0^p, k_1^p, s^p)]_{p=1}^L$
    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
    of $[(Z_0^p, T_0^p, Z^p)]_{p=1}^L$ are already calculated.
  • Calculate $e = \mathtt{H_{scalar}}(e, [X_j]_{j=0}^{N-1}, [(Z_0^p, T_0^p, Z^p)]_{p=1}^L)$
  • Let $c_0 = e$
  • Continue all the $L$ parallel procedures to the point, where the
    challenge pair $(c_{11}, c_{13})$ is to be obtained. At that moment the
    values of $[t_0^p]_{p=1}^L$ and $[H_1^p]_{p=1}^L$ are already calculated.
  • Calculate $e = \mathtt{H_{scalar}}(e, [t_0^p]_{p=1}^L, [H_1^p]_{p=1}^L)$
  • Let $(c_{11}, c_{13}) = (e, \mathtt{H_{scalar}}(e))$
  • Continue all the $L$ parallel procedures to the point, where the
    challenge pair $(c_{21}, c_{23})$ is to be obtained. At that moment the
    values of $[r_1^p]_{p=1}^L$ and $[H_2^p]_{p=1}^L$ are already calculated.
  • Calculate $e = \mathtt{H_{scalar}}(e, [r_1^p]_{p=1}^L, [H_2^p]_{p=1}^L)$
  • Let $(c_{21}, c_{23}) = (e, \mathtt{H_{scalar}}(e))$
  • And so on..., until all the tuples $[(T_0^p, Z^p, t_0^p, [(r_i^p, H_i^p)]_{i=1}^n, T^p, t^p)]_{p=1}^L$ and
    $(c_0, [(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n, c)$ are calculated.
  • Let $[\sigma^p]_{p=1}^L = [(T_0^p, Z^p, t_0^p, [(r_i^p, H_i^p)]_{i=1}^n, T^p, t^p)]_{p=1}^L$
  • Return $[\sigma^p]_{p=1}^L$
```

- **MRL2SPoM.Verif** takes a proof generated by **Prove** and returns 0 or 1. **Verif** is **MRL2S.JoinVerif** translated to the non-interactive setting. Specification is in Listing 4.

Listing 4: **MRL2SPoM.Verif** specification.

```
Input:  $e$                  --scalar seed, same as used for GetProof call
        $[\sigma^p]_{p=1}^L$  --proof, a vector of $\sigma$'s
Output: 0 or 1              --verification is failed or completed ok
Procedure:
  • Let $[P_j]_{j=0}^{N/2-1}$ = SetGen()
  • Let $[Z_0^p]_{p=1}^L$ = MembersGen()
  • Let $F = \mathtt{H_{point}}([P_j]_{j=0}^{N/2-1}, [Z_0^p]_{p=1}^L)$. Thus, $F \,!= \mathrm{lin}(\ [Z_0^p]_{p=1}^L \cup [P_j]_{j=0}^{N/2-1} \cup [G_i]_{i=0}^{D-1}\ )$
                                  holds.
  • Let $[X_j]_{j=0}^{N-1}$ = GetDecoySet($F$)
  • Extract the values of $[(T_0^p, Z^p)]_{p=1}^L$ from $[\sigma^p]_{p=1}^L$
  • Calculate $e = \mathtt{H_{scalar}}(e, [X_j]_{j=0}^{N-1}, [(Z_0^p, T_0^p, Z^p)]_{p=1}^L)$
  • Let $c_0 = e$
  • Extract the values of $[t_0^p]_{p=1}^L$ and $[H_1^p]_{p=1}^L$ from $[\sigma^p]_{p=1}^L$
  • Calculate $e = \mathtt{H_{scalar}}(e, [t_0^p]_{p=1}^L, [H_1^p]_{p=1}^L)$
  • Let $(c_{11}, c_{13}) = (e, \mathtt{H_{scalar}}(e))$
  • Extract the values of $[r_1^p]_{p=1}^L$ and $[H_2^p]_{p=1}^L$ from $[\sigma^p]_{p=1}^L$
  • Calculate $e = \mathtt{H_{scalar}}(e, [r_1^p]_{p=1}^L, [H_2^p]_{p=1}^L)$
  • Let $(c_{21}, c_{23}) = (e, \mathtt{H_{scalar}}(e))$
  • And so on..., until the tuple $(c_0, [(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n, c)$ is restored.
    At this moment all the 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 $[\sigma^p]_{p=1}^L$.
  • For $p = 1 \dots L$:
      If $(t_0^p Z_0^p + c_0 Z^p) \neq T_0^p$ then Return 0
  • Calculate $R = \mathrm{Rsum}(n, N, [X_j]_{j=0}^{N-1}, [(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n)$
  • For $p = 1 \dots L$:
      Let $S = Z^p$
      For $i = 1 \dots n$:
        $S = S + r_i^p H_i^p$
        If $(S == 0)$ or $(r_i^p == 0)$ or $(H_i^p == 0)$ then Return 0
      $W = S$
      If $(t^p W + cR) \neq T^p$ then Return 0
  • Return 1
```

Overall, the **MRL2SPoM** scheme works as follows:

- Prover and Verifier agree on the sets used, namely, on the **SetGen** and **MembersGen** functions.
- Knowing private keys $[(v^p, s^p)]_{p=1}^L$ that connect the member set elements $[Z_0^p]_{p=1}^L$ returned by **MembersGen** to the elements of the set $[P_j]_{j=0}^{N/2-1}$ returned by **SetGen**, Prover calls **Prove** and obtains the proof $[\sigma^p]_{p=1}^L$.

It should be added that Prover also passes to **Prove** an arbitrary seed e, which sets up the random oracle. The meaning and use of the seed e is the same as the use of seed in [14].

- Prover sends the proof $[\sigma^p]_{p=1}^L$ and the seed e to Verifier.
- Verifier calls **Verif** for $[\sigma^p]_{p=1}^L$ and e. If 1 is returned, then Verifier is convinced that Prover knows the private keys that connect each element of the member set $[Z_0^p]_{p=1}^L$ to an element of the set $[P_j]_{j=0}^{N/2-1}$.

8.2.1 MRL2SPOM COMPLETENESS AND SOUNDNESS

The **MRL2SPoM** procedures match the **MRL2S** procedures translated to the non-interactive setting with the Fiat-Shamir heuristic. To ascertain orthogonality of the decoy set returned from **MRL2SPoM.GetDecoySet** we refer to the OrtHalfShift lemma from Section 3. Thus, from the **MRL2S** protocol completeness and soundness, the **MRL2SPoM** scheme is complete, and successful **MRL2SPoM.Verif** implies Prover's knowledge of tuples $[(k_0^p, k_1^p, s^p)]_{p=1}^L$ such that for each of them holds

$$Z_0^p = k_0^p P_{s^p} + k_1^p Q_{s^p} \ , \tag{72}$$

where $[P_j]_{j=0}^{N/2-1}$ and $[Q_j]_{j=0}^{N/2-1}$ are defined as in the **MRL2SPoM.GetDecoySet** implementation (Listing 2).

For each $p \in [1, L]$, from the **MRL2SPoM.GetDecoySet** implementation (Listing 2), we have $Q_{s^p} = G_{s^p} + F$, and hence the equation (72) rewrites as

$$Z_0^p = k_0^p P_{s^p} + k_1^p G_{s^p} + k_1^p F \ . \tag{73}$$

According to the **MRL2SPoM.Prove** specification (Listing 3), F is a hash point of the points P_{s^p} and Z_0^p, with the point G_{s^p} fixed before the hashing. Therefore, if $k_1 \neq 0$, then $F = \text{lin}(Z_0^p, P_{s^p}, G_{s^p})$, that breaks the $\mathbf{H_{point}}$ function property of being indifferentiable from a random oracle.

Thus, with overwhelming probability the scalar k_1 in the equations (73) and (72) is equal to zero. This turns the L equations (72) into the relation (70) and hence the **MRL2SPoM** scheme is sound.

8.2.2 MRL2SPOM ZERO-KNOWLEDGE AND WITNESS-EXTENDED EMULATION

As shown in 7.2.2, each **MRL2S** transcript contains the independently uniformly distributed random items together with the inputs $[Z_0^p]_{p=1}^L$ and with the completely dependent $[T_0^p, T^p]_{p=1}^L$ items. The **MRL2SPoM** scheme honest transcript space form a subspace of the **MRL2S** protocol honest transcript space.

Namely, the **MRL2SPoM** honest transcripts are those **MRL2S** honest transcripts that have $k_1^p = 0$ in the $[Z_0^p]_{p=1}^L$ input openings. Therefore, any **MRL2SPoM** honest transcript reveals no more than the same **MRL2S** honest transcript may reveal, that is, as **MRL2S** is zero-knowledge, it reveals no more than the inputs $[Z_0^p]_{p=1}^L$ reveals.

A simulator for the **MRL2SPoM** scheme is identical to the simulator for **MRL2S**. Thus, the **MRL2SPoM** scheme is zero-knowledge.

A WEE emulator for the **MRL2SPoM** scheme is identical to the emulator for **MRL2S**. It finds witness for the relation (70) instead of the relation (69), as all the L coefficients k_1^p in the L equations (72) are equal to zero.

8.2.3 MRL2SPOM COMPLEXITIES

The **MRL2SPoM** proof size, recalling the proof is $[\sigma^p]_{p=1}^L$, is shown in Table 16. It is the same as the **MRL2S** amount of transmitted data. The scalar seed e is not accounted, as it can have any value agreed between Prover and Verifier, e.g. be fixed as $e = 0$.

Table 16: **MRL2SPoM** proof size.

	$\mathbb{G}$	$\mathbb{F}$
MRL2SPoM	$L(n+3)$	$L(n+2)$

The **MRL2SPoM** verification complexity is shown in Table 17, where $N = 2^n$. We use the same optimization for the Rsum calculation, as in **MRL2S**. The scalar-scalar multiplications and $\mathbf{H_{scalar}}$ calls are assumed taking a negligible amount of the computational time.

Table 17: **MRL2SPoM** verification complexity.

	multi-exp(N)	single-exp	$\mathbf{H_{point}}$
MRL2SPoM	1	$nL + 3L + 1$	1

9 LINKABLE RING SIGNATURE BASED ON MRL2SPOM

We construct **MRL2SLnkSig** linkable ring signature scheme on the base of the **MRL2SPoM** membership proof. We will assume that there are senders of messages that have key pairs (b, B) of secret and public keys such that $B = bG$, and that senders sign their messages with our ring signature, selecting ring members ad hoc.

The idea of **MRL2SLnkSig** is that we encode each public key B in the ring as a point $(z\mathbf{H_{point}}(B) + B)$ in the **MRL2SPoM** set. We let sender publish an auxiliary point I together with a **MRL2SPoM** membership proof for that the point $(zI + G)$ multiplied by some secret factor belongs to a set composed of $(z\mathbf{H_{point}}(B) + B)$'s. The coefficient z is a randomness picked after I is published. This way the sender proves knowledge of factor w such that the equality $(z\mathbf{H_{point}}(B) + B) = w(zI + G)$ holds for some public key B in the ring. It turns out that $(w = b)$ always holds in this case. Thus, the **MRL2SLnkSig** ring signature will be the **MRL2SPoM** membership proof followed by the auxiliary point I.

To make sure the constructed **MRL2SLnkSig** signature scheme is applicable in the real world, we will describe and prove its security using the common linkable ring signature security model definitions and methods. Particularly, we will prove no PPT adversary is able to forge it, i.e. to produce an acceptable signature without knowing an appropriate private key, even observing any number of signatures made by others.

9.1 PRELIMINARY LEMMA

Random weighting is a common cryptography technique for combining two or more proofs into a single one. It is used in different forms for various scenarios, e.g. as in [20, 26]. Let us formulate the following lemma for a variant of the random weighting that we need in.

RandomWeighting-WEE lemma:
For any four non-zero elements X, Y, C, D such that $C \mathrel{!\sim} D$ holds, for the relation

$$\mathcal{R} = \{ \ (\ (X, Y, C, D), w \ ) \mid (X = wC) \wedge (Y = wD) \ \}, \tag{74}$$

the following protocol (Table 18) is complete, sound, and has witness-extended emulation.

Table 18: RandomWeighting-WEE lemma protocol.

$\longleftarrow$ Verifier sends a non-zero random scalars z to Prover
Prover and Verifier play a WEE protocol of the evidence $(zX + Y) \sim (zC + D)$

Proof: The protocol completeness follows from that it is defined for any elements X, Y, C, D such that $C \mathrel{!\sim} D$. The protocol soundness follows from the witness-extended emulation that we will prove now.

We build a WEE emulator for this lemma protocol (Table 18). For the first, the emulator unwinds the evidence $(zX + Y) \sim (zC + D)$ and obtains witness w such that

$$(zX + Y) = w(zC + D) \ . \tag{75}$$

Next, it unwinds to the point of the challenge z and, resuming with a new value z' for it, obtains w' such that

$$(z'X + Y) = w'(z'C + D) \ . \tag{76}$$

Subtracting the equations (75) and (76) from each other and dividing by non-zero $(z' - z)$, it gets

$$X = ((w'z' - wz)/(z' - z))C + ((w' - w)/(z' - z))D \ . \tag{77}$$

Thus, it has a representation of X as a linear combination of C and D, which are orthogonal by the premise. Hence, by the OrtUniqueRepresentation from Section 3, it has two equations

$$\begin{aligned} u &= (w'z' - wz)/(z' - z) \ , \\ v &= (w' - w)/(z' - z) \ , \end{aligned} \tag{78}$$

where u and v are constants. From the equations (78) it obtains the equation

$$u = vz' + w \,, \tag{79}$$

which connects z' and w from two independent of each other transcripts. For the equation (79) to hold for the independent transcripts, v should be equal to zero. Therefore, w appears to be a constant, that is, $w = w'$. Next, from the decomposition (77) the emulator obtains

$$X = wC \,,$$

and from the equation (75) it obtains

$$Y = wD \,.$$

We have shown an emulator that extracts witness for the relation (74) by traversing the transcript tree in a polynomial number of steps. Hence, the protocol has witness-extended emulation. The lemma is proven.

9.2 MRL2SLNKSIG LINKABLE RING SIGNATURE

9.2.1 MRL2SLNKSIG SCHEME

The **MRL2SLnkSig** linkable ring signature scheme is the following four procedures

$$\mathbf{MRL2SLnkSig} = \{\mathbf{RingGen}, \mathbf{Sign}, \mathbf{Verif}, \mathbf{Link}\} \,,$$

where:

- **MRL2SLnkSig.RingGen** returns a vector $[B_j]_{j=0}^{N/2-1}$ of arbitrary points. These points are only required to be non-zero and unequal to each other. Actual signer's public keys are to be placed among them.
- **MRL2SLnkSig.Sign** takes a scalar message m, an actual signer's vector of private keys $[(b^p, s^p)]_{p=1}^{L}$ such that $[(b^p, s^p) \ | \ b^p G = B_{s^p}, \ s^p \in [0, N/2-1], \ \forall i, j : s^i \neq s^j]_{p=1}^{L}$, and returns a signature $\mathfrak{S} = [(I^p, \sigma^p)]_{p=1}^{L}$ on success or 0 on error. The $[I^p]_{p=1}^{L}$ values contained in the signature are called key images. Implementation is shown in Listing 5. Note we use lambda notation to denote procedure assignments in the listings.

Listing 5: **MRL2SLnkSig.Sign** implementation.

```
Input:  $m$                              --message
        $[(b^p, s^p)]_{p=1}^{L}$               --private keys
Output: $[(I^p, \sigma^p)]_{p=1}^{L}$ or 0        --signature on success,
                                         --0 on failure
Procedure:
    $[B_j]_{j=0}^{N/2-1} = \mathtt{RingGen}()$
    $[I^p]_{p=1}^{L} = [\mathtt{H_{point}}(b^p G)/b^p]_{p=1}^{L}$
    For $j = 1 \ldots L$:
        If $I^j \in ([I^p]_{p=1}^{L} \setminus \{I^j\})$ then Return 0
    $z = \mathtt{H_{scalar}}(m, [B_j]_{j=0}^{N/2-1}, [I^p]_{p=1}^{L})$
    $\mathtt{MRL2SPoM.SetGen} = \lambda.([B_j + z\mathtt{H_{point}}(B_j)]_{j=0}^{N/2-1})$
    $\mathtt{MRL2SPoM.MembersGen} = \lambda.([G + zI^p]_{p=1}^{L})$
    $e = \mathtt{H_{scalar}}(z)$
    $proof = \mathtt{MRL2SPoM.Prove}(e, [(1/b^p, s^p)]_{p=1}^{L})$
    If $proof$ == 0 then Return 0
    $[\sigma^p]_{p=1}^{L} = proof$
    Return $[(I^p, \sigma^p)]_{p=1}^{L}$
```

- **MRL2SLnkSig.Verif** takes a scalar message m and a signature $\mathfrak{S}$ generated by **Sign**. It returns 1 or 0, meaning successful or failed verification completion. Implementation is in Listing 6.

Listing 6: **MRL2SLnkSig.Verif** implementation.

```
Input:  $m$                           --message
        $[(I^p, \sigma^p)]_{p=1}^L$   --signature
Output: 1 or 0                        --success or failure
Procedure:
    $[B_j]_{j=0}^{N/2-1}$ = RingGen()
    $z = \mathbf{H_{scalar}}(m, [B_j]_{j=0}^{N/2-1}, [I^p]_{p=1}^L)$
    MRL2SPoM.SetGen = $\lambda.([B_j + z\mathbf{H_{point}}(B_j)]_{j=0}^{N/2-1})$
    MRL2SPoM.MembersGen = $\lambda.([G + zI^p]_{p=1}^L)$
    $e = \mathbf{H_{scalar}}(z)$
    If MRL2SPoM.Verif($e, [\sigma^p]_{p=1}^L$) == 0 then Return 0
    Return 1
```

- **MRL2SLnkSig.Link** takes a pair $([I_0^p]_{p=1}^L, [I_1^p]_{p=1}^L)$ of key image sets from two signatures successfully verified by **Verif**. It returns 1 or 0, meaning the signatures are linked or not linked. Implementation is in Listing 7.

Listing 7: **MRL2SLnkSig.Link** implementation.

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

The main **MRL2SLnkSig** usage scenario is as follows:

- Prover and Verifier agree on a **MRL2SLnkSig.RingGen** implementation to return the same ring $[B_j]_{j=0}^{N/2-1}$ for both parties.
- Prover signs a message m with L private keys $[(b^p, s^p)]_{p=1}^L$ by calling **MRL2SLnkSig.Sign** and gets a signature $\mathfrak{S} = [(I^p, \sigma^p)]_{p=1}^L$.
- Verifier receives the message m and signature $\mathfrak{S}$, and calls **MRL2SLnkSig.Verif** for them. Iff the call returns 1, then Verifier is convinced that Prover has signed the message m with L private keys, which correspond to some L public keys in the ring. It is also convinced that the vector $[I^p]_{p=1}^L$ contains key images of the signing private keys. Note if it is allowed for $[B_j]_{j=0}^{N/2-1}$ to contain equal public keys, then Prover may sign with equal private keys, if this is a case then the vector of key images $[I^p]_{p=1}^L$ contain duplicates.
- When the above steps are performed multiple times, Verifier is convinced that a number of messages were actually signed. For any two successfully verified signatures, Verifier has two vectors $[I_0^p]_{p=1}^L$ and $[I_1^p]_{p=1}^L$ of key images contained in them. Verifier calls **MRL2SLnkSig.Link** for these key image vectors and, iff it returns 1, Verifier gets convinced that at least one common private key was used to sign both signatures.

9.2.2 SCHEME COMPLETENESS, SOUNDNESS, AND WITNESS-EXTENDED EMULATION

The **MRL2SLnkSig** scheme is a composition of the random weighting protocol in Table 18 and the **MRL2SPoM** scheme. The **MRL2SPoM** scheme plays a role of the evidence $(zX + Y) \sim (zC + D)$ in the protocol. By the **MRL2SPoM** scheme properties and according to the RandomWeighting-WEE lemma, we can assert the following.

As both the protocol in Table 18 and the **MRL2SPoM** scheme are complete, the resulting **MRL2SLnkSig** scheme is complete. As both the protocol and the **MRL2SPoM** scheme are sound, the **MRL2SLnkSig** scheme is sound.

As both the protocol in Table 18 and the **MRL2SPoM** scheme have witness-extended emulations, the **MRL2SLnkSig** scheme has witness-extended emulation. The relation that the **MRL2SLnkSig** emulator finds witness for is the intersection of the relations (70) and (74)

$$\mathcal{R} = \bigcup_{p=1}^{L} \left\{ \, (I^p, (b^p, s^p)) \;\middle|\; (B_{s^p} = b^p G \;\wedge\; \mathbf{H_{point}}(B_{s^p}) = b^p I) \,,\; s^p \in [0, N/2-1] \, \right\} \tag{80}$$

9.2.3 STRUCTURE AND VIEW OF THE MRL2SLNKSIG SIGNATURE

The **MRL2SLnkSig** signature is a vector of L pairs, where the first item in each pair is a key image denoted as I, and the second item is a **MRL2SPoM** proof (71) denoted as σ. Structure and view of the $[\sigma^p]_{p=1}^{L}$ part of the signature is described in 8.2.2, there is shown that σ's don't reveal any information and can be viewed as completely random, excluding simply dependent items. The question is how much information can be obtained from $[I^p]_{p=1}^{L}$.

Evidently, I is revealing some information. For instance, according to the relation (80), every time B is the actual signer, the same I appears in the key image vector. For any public key B, an adversary that has access to the signing oracle is able to find the corresponding key image I. In other words, due to the key images, the space of all **MRL2SLnkSig** signatures is partitioned by public keys. That is, each public key together with its corresponding key image establishes a distinguishable partition in the space.

On the other hand, if the rule of signing only once with the same public key is somehow imposed to the system, then the space looks completely random, provided that the public keys B are distributed independently and uniformly at random. This is due to that the triples $(B, \mathbf{H_{point}}(B), I)$ look completely random by DDH assumption in this case. Moreover, even if the public keys B are distributed non-uniformly, the space remains completely random except for the keys B.

9.2.4 SIGNATURE SIMULATION

If the **MRL2SLnkSig** signature was zero knowledge, then, by definition of zero knowledge, there would be an interactive simulator that for any ring $[P_j]_{j=0}^{N/2-1}$, any input I, and any set of random challenges known in advance yields an acceptable signature indistinguishable from an honest one. However, it is not.

An example of why such simulator doesn't exist is following. Suppose a simulator is fed with the ring of two honest public keys $\{P_0, P_1\}$ and with a random I. The honest key images I_0, I_1 to the keys in the ring $\{P_0, P_1\}$ can always be found from the space of the honest **MRL2SLnkSig** signatures. Thus, any signature produced by the simulator will be distinguishable from an honest one, as there is only negligible probability that the random I is equal to one of the known I_0, I_1.

Nevertheless, if the ring contains a dishonest public key, i.e. a public key with an absolutely unknown private key, then the simulation is possible for random input I. Suppose, the key P_1 in the ring $\{P_0, P_1\}$ is dishonest. Then the simulator yields (I, σ), where σ is a **MRL2SPoM** simulated transcript indistinguishable from an honest **MRL2SPoM** transcript. The random point I is indistinguishable from an honest key image, as there is no key image for P_1 in the space of the honest signatures to disprove that. Thus, the simulated signature (I, σ) is indistinguishable from an honest one in this case.

9.2.5 COMPLEXITIES

The **MRL2SLnkSig** signature size is the size of its internal **MRL2SPoM** proof plus the size of L key images. It is shown in Table 19.

Table 19: **MRL2SLnkSig** signature size.

	$\mathbb{G}$	$\mathbb{F}$
MRL2SLnkSig	$L(n+4)$	$L(n+2)$

The **MRL2SLnkSig** verification complexity is shown in Table 20. In addition to the optimization used in **MRL2SPoM**, in **MRL2SLnkSig** we optimize the calculations related to the decoy set even elements $[P_j]_{j=0}^{N/2-1}$. Instead of calculating them directly within **MRL2SPoM.SetGen**, we defer the exponentiations by the coefficient z until the single multi-exponentiation is collected and do all exponentiations with it. In the meantime, before it is collected, the necessary hashes of P_j's are calculated as the hashes of the $(B_j, \mathbf{H_{point}}(B_j), z)$ tuples.

Table 20: **MRL2SLnkSig** verification complexity.

	multi-exp$(3N/2)$	single-exp	$\mathbf{H_{point}}$
MRL2SLnkSig	1	$nL + 4L + 2$	$N/2 + 1$

Recalling N commonly denotes a ring size, whereas we use N to denote the internal decoy set size which is two times larger than the ring size, in Table 21 we provide the same data as in Tables 19, 20 in the common terms. Also, in Table 21 we assume the size of a point from $\mathbb{G}$ is equal to the size of a scalar from $\mathbb{F}$.

Table 21: **MRL2SLnkSig** signature size and verification complexity, where:
- N is the ring size
- L is the threshold
- $\boldsymbol{mexp}(3N)$ is the multi-exponentiation of $3N$ summands
- $\mathbf{H_{pt}}$ is one call to $\mathbf{H_{point}}$

	Size	Verification complexity
MRL2SLnkSig	$2L \cdot \log_2 N + 8L$	$\boldsymbol{mexp}(3N) + L \cdot \log_2 N + 5L + 2 + (N + 1)\mathbf{H_{pt}}$

9.3 MRL2SLNKSIG SIGNATURE SECURITY

The works presenting new signature schemes, such as [1, 4, 3, 14, 17, 19, 20, 23, 26], contain proofs that they cannot be tampered with certain types of attacks. These proofs have a lot in common, e.g. in modeling the attacks, however they may differ from each other in details. The works [14, 19, 20] contain a set of security requirements usually imposed on linkable ring signatures, so below we mainly rehash the definitions from these works and use the methods presented there to prove resistance to attacks.

First, we consider a generic linkable ring signature concept, that we call GLRS, along with its security requirements. GLRS is similar to the LRS concept from [20], with the only difference that KeyGen does not necessarily generate independently and evenly distributed secret keys. We prove that the **MRL2SLnkSig** signature fits in the GLRS security requirements. Moreover, we show that **MRL2SLnkSig** is unforgeable w.r.t. insider corruption and also existentially unforgeable against attacks using adaptive chosen messages and adaptive chosen public keys (EU_CMA/CPA). Next, we discuss the case of signing with multiple keys and briefly show that **MRL2SLnkSig** remains secure in this case.

9.3.1 DEFINITIONS FOR $L = 1$

Generic linkable ring signature (GLRS) definition:
GLRS is four procedures:
- KeyGen() $\rightarrow (x, X)$: Generates a secret key x and corresponding public key X such that $X = xG$. It is not required here for the secret keys to be chosen uniformly at random. The only requirement is that seeing only the X's generated by KeyGen and having no additional information it is hard to find x for any X.
- Sign$(x, m, R) \rightarrow \boldsymbol{\sigma}$: Generates a signature $\boldsymbol{\sigma}$ on a message m with respect to the ring $R = \{X_0, \ldots, X_{n-1}\}$, provided that x is a secret key corresponding to some $X_i \in R$ generated by KeyGen. The ring R itself is not required to be composed only of keys generated by KeyGen, the only two requirements to R are that the actual signer's public key $X_i \in R$ have to be generated by KeyGen and R has to contain no duplicates.
- Verify$(\boldsymbol{\sigma}, m, R) \rightarrow \{0, 1\}$: Verifies a signature $\boldsymbol{\sigma}$ on a message m with respect to the ring R. Outputs 0 if the signature is rejected, and 1 if accepted.
- Link$(\boldsymbol{\sigma}, \boldsymbol{\sigma}') \rightarrow \{0, 1\}$: Determines if signatures $\boldsymbol{\sigma}$ and $\boldsymbol{\sigma}'$ were signed using the same private key. Outputs 0 if the signatures were signed using different private keys, and 1 if they were signed using the same private key.

Correctness definition:
Consider this game between a challenger and a PPT adversary $\mathcal{A}$:
- The challenger runs KeyGen() $\rightarrow (x, X)$ and supplies the keys to $\mathcal{A}$.
- The adversary $\mathcal{A}$ chooses a ring such that $X \in R$ and a message m, and sends them to the challenger.
- The challenger signs the message with Sign$(x, m, R) \rightarrow \boldsymbol{\sigma}$.

If Pr[Verify$(\boldsymbol{\sigma}, m, R) = 1] = 1$, we say that the GLRS is perfectly correct. If Pr[Verify$(\boldsymbol{\sigma}, m, R) = 1] \approx 1$, we say that the GLRS is simply correct. Note the sign '$\approx$' means overwhelming probability, whereas '$=$' means equality.

Unforgeability w.r.t. insider corruption definition:
Consider this game between a challenger and a PPT adversary $\mathcal{A}$:
- The adversary $\mathcal{A}$ is granted access to a public key oracle GenOracle that (on the i-th invocation) runs KeyGen() $\rightarrow (x_i, X_i)$ and returns X_i to $\mathcal{A}$. In this game, KeyGen genarates key pairs (x, X)'s where x's are chosen independently and uniformly at random.

- The adversary $\mathcal{A}$ is granted access to a corruption oracle CorruptOracle(i) that returns x_i if it corresponds to a query to GenOracle.
- The adversary $\mathcal{A}$ is granted access to a signing oracle SignOracle(X, m, R) that runs Sign(x, m, R) $\rightarrow \boldsymbol{\sigma}$ and returns $\boldsymbol{\sigma}$ to $\mathcal{A}$, provided that X corresponds to a query to GenOracle and $X \in R$.
- Then, $\mathcal{A}$ outputs ($\boldsymbol{\sigma}, m, R$) such that SignOracle was not queried with (_, m, R), all keys in R were generated by queries to GenOracle, and no key in R was corrupted by CorruptOracle.

If Pr[Verify($\boldsymbol{\sigma}, m, R$) = 1] ≈ 0, we say that the GLRS is unforgeable w.r.t. insider corruption.

Existential unforgeability against adaptive chosen message / public key attackers (EU_CMA/CPA) definition:
Consider this game between a challenger and a PPT adversary $\mathcal{A}$:
- The adversary $\mathcal{A}$ is granted access to a public key oracle GenOracle that (on the i-th invocation) runs KeyGen() $\rightarrow (x_i, X_i)$ and returns X_i to $\mathcal{A}$.
- The adversary $\mathcal{A}$ is granted access to a signing oracle SignOracle(X, m, R) that runs Sign(x, m, R) $\rightarrow \boldsymbol{\sigma}$ and returns $\boldsymbol{\sigma}$ to $\mathcal{A}$, provided that X corresponds to a query to GenOracle and $X \in R$.
- Then, $\mathcal{A}$ outputs ($\boldsymbol{\sigma}, m, R$) such that SignOracle was not queried with (_, m, R), all keys in R were generated by queries to GenOracle.

If Pr[Verify($\boldsymbol{\sigma}, m, R$) = 1] ≈ 0, we say that the GLRS is unforgeable against adaptive chosen message and adaptive chosen public key attackers (EU_CMA/CPA).

As can be seen from the definitions of unforgeability w.r.t. insider corruption and EU_CMA/CPA, their games differ only in that the former deals with independent uniformly random distribution of private keys and involves key corruption using CorruptOracle, whereas the latter deals with possibly dependent private keys without any possibility of corruption. Naturally, the EU_CMA/CPA definition cannot allow for key corruption, because a single call to CorruptOracle may corrupt all the keys at once, when they are dependent of each other.

Anonymity definition:
Consider this game between a challenger and a PPT adversary $\mathcal{A}$:
- The adversary $\mathcal{A}$ is granted access to the public key oracle GenOracle and the corruption oracle CorruptOracle. In this game, KeyGen genarates key pairs (x, X)'s where x's are chosen independently and uniformly at random.
- The adversary $\mathcal{A}$ chooses a message m, a ring R, and indices i_0 and i_1, and sends them to the challenger. We require that $X_{i_0}, X_{i_1} \in R$ such that both keys were generated by queries to GenOracle, and neither key was corrupted by CorruptOracle.
- The challenger selects a uniformly random bit $b \in \{0, 1\}$, generates the signature Sign(x_{i_b}, m, R) $\rightarrow \boldsymbol{\sigma}$, and sends it to $\mathcal{A}$.
- The adversary $\mathcal{A}$ chooses a bit $b' \in \{0, 1\}$.

If $\Pr[b' = b] \approx 1/2$ and $\mathcal{A}$ did not make any corruption queries after receiving $\boldsymbol{\sigma}$, we say that the GLRS is anonymous.

Anonymity w.r.t. chosen public key attackers (anonymity w.r.t. CPA) definition:
Consider this game between a challenger and a PPT adversary $\mathcal{A}$:
- The adversary $\mathcal{A}$ is granted access to the public key oracle GenOracle.
- The adversary $\mathcal{A}$ chooses a message m, a ring R, and indices i_0 and i_1, and sends them to the challenger. We require that $X_{i_0}, X_{i_1} \in R$ such that both keys were generated by queries to GenOracle.
- The challenger selects a uniformly random bit $b \in \{0, 1\}$, generates the signature Sign(x_{i_b}, m, R) $\rightarrow \boldsymbol{\sigma}$, and sends it to $\mathcal{A}$.
- The adversary $\mathcal{A}$ chooses a bit $b' \in \{0, 1\}$.

If $\Pr[b' = b] \approx 1/2$ and $\mathcal{A}$ did not make any corruption queries after receiving $\boldsymbol{\sigma}$, we say that the GLRS is anonymous w.r.t. CPA.

As can be seen from the definitions of anonymity and anonymity w.r.t. CPA, their games are as different as the games of unforgeability w.r.t. insider corruption and EU_CMA/CPA.

Linkability definition:
Consider the following game between a challenger and a PPT adversary $\mathcal{A}$:
- For $i \in [0, k-1]$, the adversary $\mathcal{A}$ produces a public key X_i, message m_i, ring R_i, and signature $\boldsymbol{\sigma}_i$.
- The adversary $\mathcal{A}$ produces another message m, ring R, and signature $\boldsymbol{\sigma}$.

- All tuples $(X_i, m_i, R_i, \boldsymbol{\sigma}_i)$ and $(m, R, \boldsymbol{\sigma})$ are sent to the challenger.
- The challenger checks the following:
  - $|V| = k$, where $V = \bigcup_{i=0}^{k-1} R_i$.
  - Each $X_i \in V$.
  - Each $R_i \subset V$.
  - Verify$(\boldsymbol{\sigma}_i, m_i, R_i) = 1$ for all i.
  - Verify$(\boldsymbol{\sigma}, m, R) = 1$.
  - For all $i \neq j$, we have Link$(\boldsymbol{\sigma}_i, \boldsymbol{\sigma}_j) =$ Link$(\boldsymbol{\sigma}_i, \boldsymbol{\sigma}) = 0$.
- If all checks pass, $\mathcal{A}$ wins.

If $\mathcal{A}$ wins with only negligible probability for all k, we say the GLRS is linkable.

Non-frameability definition:
Consider also the following game between a challenger and a PPT adversary $\mathcal{A}$:
- The adversary $\mathcal{A}$ is granted access to the public-key oracle GenOracle.
- The adversary $\mathcal{A}$ is granted access to the corruption oracle CorruptOracle.
- The adversary $\mathcal{A}$ is granted access to the signing oracle SignOracle.
- In this game, KeyGen genarates key pairs (x, X)'s, where x's are chosen independently and uniformly at random.
- The adversary $\mathcal{A}$ chooses a public key X that was generated by a query to GenOracle, but was not corrupted by CorruptOracle. It selects a message m and ring R such that $X \in R$. It queries SignOracle$(X, m, R) \to \boldsymbol{\sigma}$.
- The adversary $\mathcal{A}$ then produces a tuple $(m', R', \boldsymbol{\sigma}')$ and sends $(m', R', \boldsymbol{\sigma}')$ to the challenger, along with $(X, m, R, \boldsymbol{\sigma})$.
- If Verify$(\boldsymbol{\sigma}', m', R') = 0$ or if $\boldsymbol{\sigma}'$ was produced using a query to SignOracle, the challenger aborts.

If Pr[Link$(\boldsymbol{\sigma}, \boldsymbol{\sigma}') = 1] \approx 0$, we say that the GLRS is non-frameable.

Non-frameability w.r.t. chosen public key attackers (non-frameability w.r.t. CPA) definition:
Consider also the following game between a challenger and a PPT adversary $\mathcal{A}$:
- The adversary $\mathcal{A}$ is granted access to the public-key oracle GenOracle.
- The adversary $\mathcal{A}$ is granted access to the signing oracle SignOracle.
- The adversary $\mathcal{A}$ chooses a public key X that was generated by a query to GenOracle. It selects a message m and ring R such that $X \in R$. It queries SignOracle$(X, m, R) \to \boldsymbol{\sigma}$.
- The adversary $\mathcal{A}$ then produces a tuple $(m', R', \boldsymbol{\sigma}')$ and sends $(m', R', \boldsymbol{\sigma}')$ to the challenger, along with $(X, m, R, \boldsymbol{\sigma})$.
- If Verify$(\boldsymbol{\sigma}', m', R') = 0$ or if $\boldsymbol{\sigma}'$ was produced using a query to SignOracle, the challenger aborts.

If Pr[Link$(\boldsymbol{\sigma}, \boldsymbol{\sigma}') = 1] \approx 0$, we say that the GLRS is non-frameable w.r.t. CPA.

The games of non-frameability and non-frameability w.r.t. CPA are as different as the games of unforgeability w.r.t. insider corruption and EU_CMA/CPA.

9.3.2 SECURITY PROOF FOR $L = 1$

The **MRL2SLnkSig** signature scheme defined in 9.2.1, for $L = 1$, can be considered GLRS by appropriately renaming procedures and adding KeyGen as specified for the GLRS adversarial games. We will now prove the security properties of the **MRL2SLnkSig** scheme for this case.

In 9.2.4 we noticed indistinguishability of simulated signature from an honest one when a dishonest key exists in the ring. The following lemma formalizes that observation.

MRL2SLnkSig-Simulation lemma:
For the **MRL2SLnkSig** signature scheme considered as GLRS, for $L = 1$, for the games specified in the unforgeability w.r.t. insider corruption and EU_CMA/CPA definitions,
- if for some X_j obtained from GenOracle $\mathcal{A}$ calls SignOracle$(X_j, _, _)$ one or many times with any messages, any rings R such that $X_j \in R$,
- if for these calls SignOracle returns signatures $\boldsymbol{\sigma} = (I, \sigma)$'s such that the point I is common for all these returned $\boldsymbol{\sigma}$'s with actual signer X_j, and I was picked by SignOracle independently and uniformly at random one time when producing the first $\boldsymbol{\sigma}$ for X_j,

- ∘ and if σ's in $\boldsymbol{\sigma}$'s are the corresponding **MRL2SPoM** simulated proofs,
- ∘ if RandomOracle always returns to $\mathcal{A}$ the same randomness for the same queries notwithstanding of the **MRL2SPoM** proof simulations,

then $\mathcal{A}$ is unable to distinguish these $\boldsymbol{\sigma}$'s from the honest **MRL2SLnkSig** signatures, unless X_j is corrupted. That is, in this game $\mathcal{A}$ is unable to determine if SignOracle simulates the **MRL2SLnkSig** signatures for the actual signer X_j or not, until x_j becomes known to $\mathcal{A}$.

Proof: Each honest **MRL2SLnkSig** signature, where actual signer is X_j, contains honest key image $I = \mathbf{H_{point}}(X_j)/x_j$. A distribution of the honest key image I, as follows from definition of the ideal hash function $\mathbf{H_{point}}$, is independent and uniformly random. Thus, the simulated signatures cannot be distinguished using key images.

Suppose, there is anyway an adversary $\mathcal{A}$ that distinguishes a simulated signature from the space of honest ones. Thus, $\mathcal{A}$ distinguishes a simulated **MRL2SPoM** proof σ from the space of honest **MRL2SPoM** proofs, that contradicts the **MRL2SPoM** scheme zero-knowledge property proven in 8.2.2. Therefore, there is no $\mathcal{A}$ capable of distinguishing simulated signatures in this game. The lemma is proven.

We can now move on to the next theorems.

Theorem 1:
The **MRL2SLnkSig** signature scheme for $L = 1$, considered as GLRS, has the following properties: perfect correctness, unforgeability w.r.t. insider corruption and EU_CMA/CPA.

Proof: The **MRL2SLnkSig** scheme completeness, which is proven in 9.2.2, implies correctness. Moreover, it is seen from the implementation of the scheme core protocol (Table 8) that **MRL2SLnkSig** is perfectly correct.

The **MRL2SLnkSig** scheme is unforgeable w.r.t. insider corruption, we provide a proof sketch using the method from [14], which we modified to account for differences between our signature and the signature in [14]. Instead of special soundness as in [14], we rely on the WEE property of our signature scheme. We model the $\mathbf{H_{scalar}}$ function as a random oracle RandomOracle. Considering a PPT adversary $\mathcal{A}$ which is supposed to produce a forged signature with non-negligible probability, we construct a polynomial time attack that breaks DL assumption. Instead of zero-knowledge property as in [14], we use the simulated signature indistinguishability asserted by the MRL2SLnkSig-Simulation lemma. The proof sketch is the following.

- ✓ To set up our attack, we take a PPT adversary $\mathcal{A}$ that produces an acceptable signature for a ring R of uncorrupted keys generated by KeyGen, and insert our PPT master algorithm $\mathcal{M}$ in the middle between $\mathcal{A}$ and the oracles GenOracle, CorruptOracle, SignOracle, and RandomOracle. To distinguish between the original oracles at $\mathcal{M}$ input from their counterparts at $\mathcal{M}$ output we append suffix 'In' to the input ones. Note $\mathcal{M}$ doesn't have any access to the private keys behind GenOracleIn, it can know them only with calling to CorruptOracleIn.
- ✓ Initially, the master $\mathcal{M}$ runs $\mathcal{A}$ without changing the behavior of the oracles. In polynomial time and with non-negligible probability $\mathcal{A}$ produces forgeries $\boldsymbol{\sigma}$'s for uncorrupted rings R's. Suppose, $\mathcal{A}$ makes at most q_V GenOracle calls to have $1/p$ probability of producing a forgery (explained in detail in [14]).
- ✓ Next, $\mathcal{M}$ changes the behavior of SignOracle and RandomOracle and restarts the game. Modified SignOracle, for some random $j \in [1, \dots, q_V]$, no longer redirects to SignOracleIn when requesting a signature for the actual signer X_j. Instead, it picks a point I_j independently and uniformly at random on the first call of SignOracle for the signer X_j, and starts returning signatures $\boldsymbol{\sigma} = (I_j, \sigma)$ each time it is called for X_j. Each time σ is generated, $\mathcal{M}$ provides the **MRL2SPoM** simulator with a series of random challenges. Then $\mathcal{M}$ patches RandomOracle so that instead of redirecting to RandomOracleIn, it starts returning those challenges as responses to the corresponding queries. As shown in [14], this patch does not result in the problem of getting multiple different responses for the same query to RandomOracle.
- ✓ $\mathcal{M}$ also modifies CorruptOracle so that when called to corrupt X_j it just stops $\mathcal{A}$. This modification does not reduce the likelihood of obtaining a successful forgery for the actual signer X_j, neither in the case of the unmodified SignOracle, nor in the case of the modified SignOracle. This is because the corruption of X_j during a run of $\mathcal{A}$ means that the creation of a successful forgery for the signer X_j has failed in this run.
- ✓ According to the MRL2SLnkSig-Simulation lemma, when $\mathcal{A}$ is executed with the modified version of CorruptOracle, then $\mathcal{A}$ cannot distinguish whether it is run with the modified oracles SignOracle and RandomOracle, or not. Since in the case of running $\mathcal{A}$ with unmodified SignOracle and RandomOracle the probability of successful forgery for X_j is $1/(pq_V)$, the same probability remains for the case of running $\mathcal{A}$ with the modified versions of SignOracle and RandomOracle. Thus, running $\mathcal{A}$ with simulated signatures for X_j yields in a polynomial time a successful forgery with the actual signer X_j and with an uncorrupted ring $R \ni X_j$.

✓ Now, by running $\mathcal{A}$ with simulated signatures for X_j, the master $\mathcal{M}$ rewinds in the usual way and thus gets in a polynomial time a transcript tree rooted in the uncorrupted (X_j, R) with leaves in successful forgeries. As the **MRL2SLnkSig** scheme has witness-extended emulation for the relation (80), $\mathcal{M}$ extracts witness x_j such that $X_j = x_j G$ from this transcript tree.

✓ However, the witness x_j is not known to $\mathcal{M}$, because X_j is uncorrupted and $\mathcal{M}$ doesn't have access to KeyGen, except through the use of GenOracle, which is also not revealing x_j. Thus, having supposed that **MRL2SLnkSig** can be forged in polynomial time we have built the PPT algorithm $\mathcal{M}$ that breaks DL assumption for X_j. Therefore, the supposition was incorrect and **MRL2SLnkSig** is unforgeable w.r.t. insider corruption.

The proof of the EU_CMA/CPA property for **MRL2SLnkSig** is similar to the above proof of unforgeability w.r.t. insider corruption. The difference in games is that $\mathcal{A}$ can no longer corrupt keys, however it can now use uneven key distribution. Using the Rewind-on-Success lemma from [19], this difference can be ignored. Here's a sketch of the proof.

✓ The attack setup is the same as for the above unforgeability w.r.t. insider corruption attack, CorruptOracle is no longer used.

✓ Initially, the master $\mathcal{M}$ runs $\mathcal{A}$ and obtains a forgery for some key in a polynomial time.

✓ Next, $\mathcal{M}$ builds a successful transcript tree by rewinding the forgery. By the Rewind-on-Success lemma [19], $\mathcal{M}$ is able to build it in a polynomial time.

✓ From the successful transcript tree, due to the witness-extended emulation of **MRL2SLnkSig**, $\mathcal{M}$ restores witness x_j, and thus breaks DL assumption.

The theorem is proven.

Theorem 2:
The **MRL2SLnkSig** signature scheme for $L = 1$, considered as GLRS, has the following properties: anonymity and anonymity w.r.t. CPA.

Proof: The **MRL2PoM** part σ of the **MRL2SLnkSig** signature $\boldsymbol{\sigma} = (I, \sigma)$ is sHVZK, it doesn't reveal any information about the actual signer key. Therefore, any adversarial advantage in breaking anonymity arises from the signature key image together with public key ring (I, R). For any public key X_i, by definition of the hash function $\mathbf{H_{point}}$, the corresponding key image $I = \mathbf{H_{point}}(X_i)/x_i$ has an independent uniformly random distribution. Hence, the key image I also brings no adversarial advantage.

The public key ring R with two uncorrupted keys $X_{i_0}, X_{i_1} \in R$ remains the only source of information for the adversary $\mathcal{A}$ to win the anonymity and anonymity w.r.t. CPA adversarial games. However, R, X_{i_0}, X_{i_1} are known to $\mathcal{A}$ beforehand, and thus bring no additional information. Therefore, $\mathcal{A}$ wins only with the probability of a random coin flip, i.e. according to the respective definitions, **MRL2SLnkSig** is anonymous and anonymous w.r.t. CPA.

Theorem 3:
The **MRL2SLnkSig** signature scheme for $L = 1$, considered as GLRS, has the following properties: linkability, non-frameability, and non-frameability w.r.t. CPA.

Proof: The linkability proof presented in [20] for linkable ring signature also applies to **MRL2SLnkSig**. For both the signature provided in [20] and ours, the underlying protocols are sHVZK. The difference in the linking tags doesn't affect the proof, so we refer to [20] and don't repeat the proof here.

It's the same with non-frameability. The non-frameability proof in [20] applies to **MRL2SLnkSig** as well, so we refer to the proof in [20] and don't repeat it here. Moreover, since the proof relies only on the scheme witness extraction property, namely, on special soundness in [20], and on WEE for the relation (80) in our case, and since the proof doesn't depend on the key distribution generated by KeyGen, it simultaneously attests the non-frameability and non-frameability w.r.t. CPA for our scheme.

9.3.3 SECURITY FOR MULTIPLE INPUTS

For $L \geq 1$, we define a natural extension of GLRS as follows.

Generic linkable threshold ring signature (GLTRS) definition:
GLTRS is a threshold version of GLRS, it is four procedures:

- KeyGen() $\to (x, X)$: Generates key pairs, the same as for GLRS.
- L is a threshold.
- Sign$(\vec{x}, m, R) \to \mathfrak{S}$: Generates a signature $\mathfrak{S}$ on a message m with respect to the ring $R = \{X_0, \ldots, X_{n-1}\}$, provided that $\vec{x}$, $|\vec{x}| = L$, is a set of different secret keys corresponding to some subset $\vec{X} \subseteq R$ generated

by KeyGen. The ring R itself is not required to be composed only of keys generated by KeyGen, the only two requirements to R are that the actual signer public keys $\vec{X}$ have to be generated by KeyGen and R has to contain no duplicates.

- Verify($\mathfrak{S}, m, R$) $\rightarrow \{0, 1\}$: Verifies a signature $\mathfrak{S}$ on a message m with respect to the ring R. Outputs 0 if the signature is rejected, and 1 if accepted. This procedure result has to be equal to the conjunction of L GLRS.Verify results if GLRS.Sign was respectively called for all $x \in \vec{x}$.
- Link($\mathfrak{S}, \mathfrak{S}'$) $\rightarrow \{0, 1\}$: Determines if signatures $\mathfrak{S}$ and $\mathfrak{S}'$ have a common signing key. Outputs 0 if the signatures were signed using different private keys, and 1 if they have at least one common signing key. This procedure result has to be equal to the disjunction of $L \cdot L'$ GLRS.Link results if GLRS.Sign was respectively called for all $x \in \vec{x}$ and for all $x' \in \vec{x}'$.

The **MRL2SLnkSig** signature is considered GLTRS by appropriately renaming procedures and adding KeyGen. The natural $L-$dimensional extensions for the security properties defined in 9.3.1 seem straightforward, hence we don't provide formal definitions for them here. Although, we recognize that a deeper formal investigation of linkable threshold ($L-$dimensional) ring signatures may reveal some new properties, such as in [12, 24].

MRL2SLnkSig is perfectly correct for $L \geq 1$, this is seen from the implementation (Table 8). Here we also recall that for each $(I^p, \sigma^p) \in \mathfrak{S},\ p \in [1 \dots L],\ (I^p, \sigma^p)$ is totally independent of $\mathfrak{S} \setminus (I^p, \sigma^p)$, and each σ^p reveals no more information than random noise.

MRL2SLnkSig is unforgeable w.r.t. insider corruption and also is EU_CMA/CPA for $L \geq 1$, since everything the PPT master algorithm $\mathcal{M}$ does and gets in Theorem 1 (9.3.2), it can do in polynomial time in $L-$dimensions.

Linkability, non-frameability, and non-frameability w.r.t. CPA also alive for **MRL2SLnkSig** in $L-$dimensions, because if one of them was broken in $L-$dimensions, then it would be easy to break its $1-$dimensional counterpart that would contradict to Theorem 2 or to Theorem 3 (9.3.2).

9.4 MRL2SLNKSIG SIGNATURE AND RECENTLY PROPOSED LOG-SIZE SCHEMES

We refer to the work [20], where proof sizes and verification complexities for two of the recently proposed top-performative schemes are shown in Table 1 [20]. Although we don't provide a direct performance comparison of our **MRL2SLnkSig** signature to the schemes analyzed in [20] due to the following reasons:

- The linkable signature schemes analyzed in [20] include homomorphic commitment sum proofs as well, whereas our scheme is just a linkable signature,
- Our linkable signature operates with the linking tags of the form $x^{-1}\mathbf{H_{point}}\,(xG)$, whereas, for instance, the Triptych scheme from [20] operates with the linking tags of the form $x^{-1}U$, where U is an independent generator,

in any case, we can look at the asymptotes. Assuming an $\mathbf{H_{point}}$ call is about ten times faster than an exponentiation, and thus takes about the same time as an exponentiation calculated within a multi-exponent for a roughly 2^{10} element ring, we can see that, for instance, for big N's, our signature verification time asymptote $\boldsymbol{mexp}(3N) + N\mathbf{H_{pt}}$ is not far from the RingCT 3.0 asymptote $\boldsymbol{mexp}(4N)$. Triptych asymptote $\boldsymbol{mexp}(2N)$ is about 2 times faster than both ours and RingCT 3.0.

The size asymptotes for big N's and threshold $L = 1$ are $2\log_2 N$ for our signature, the same for RingCT 3.0, and $3\log_2 N$ for Triptych. Thus, for the big, say, of 2^{15} element, rings, all the scheme sizes seem roughly equal. As for threshold $L > 1$, RingCT 3.0 provides the best asymptotic time $O(\log_2 N + L)$, whereas our signature is only $O(L\log_2 N)$.

It is worth mentioning that we use one of the most time-tested form of linking tag, $x^{\pm 1}\mathbf{H_{point}}(xG)$, which we can see, e.g. in [19, 25]. This form of linking tag is absolutely insensitive to any distribution of public keys and allows the signature to be anonymous w.r.t. chosen public key attackers.

Below we provide a couple of notes regarding possible modifications to our signature that include the homomorphic commitment sum proof and better verification time.

10 POSSIBLE EXTENSIONS

The **MRL2SPoM** membership proof and **MRL2SLnkSig** signature are things that are achievable with a few steps starting from the Lin2-Xor lemma. Thus, possible extentions could include further optimization of **MRL2SLnkSig** as well as creating other arguments of knowledge using the methods from the Lin2 (-WEE), Lin2-Xor(-WEE), Lin2-Selector(-WEE) lemmas. For instance,

- It is possible to move more exponentiations under the single multi-exponent in **MRL2SLnkSig**.
- It is possible to verify a batch of signatures at once using the random weighting technique.

- A homomorhic commitments sum proof could be attached to the signature by appending the homomorhic commitments A_j to the decoys $P_j = B_j + z\mathbf{H_{point}}(B_j)$ and separating them with another random weight ξ as $P_j = B_j + z\mathbf{H_{point}}(B_j) + \xi A_j$.
- It seems possible to create a range proof argument as in [6], developing the method of the Lin2-Xor lemma.

ACKNOWLEDGEMENTS

Author thanks everyone who occasionally had talks with him about privacy systems while writing this article, and thanks Olga Kolesnikova for reading the early drafts and making amicable comments on the narrative. Also, author would like to thank the people from Zano project for their support during the first reviews, for sharing knowledge on the current state of the art with hashing to curve, special thanks to Valeriy Pisarkov for proofreading the Lin2-Xor, Lin2- Selector lemmas and protocols.

REFERENCES

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

[2] Mihir Bellare and Phillip Rogaway. "Random oracles are practical: a paradigm for designing efficient protocols". In: *CCS '93: Proceedings of the 1st ACM conference on Computer and communications security*. Association for Computing Machinery. 1993, pp. 62–73.

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

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

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

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

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

[8] Morris Dworkin. *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. 2015-08-04 2015. DOI: `https://doi.org/10.6028/NIST.FIPS.202`.

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

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

[11] Pierre-Alain Fouque and Mehdi Tibouchi. "Indifferentiable Hashing to Barreto–Naehrig Curves". In: *Progress in Cryptology – LATINCRYPT 2012*. Springer Berlin Heidelberg, 2012, pp. 1–17.

[12] Brandon Goodell, Sarang Noether, and RandomRun. *Concise Linkable Ring Signatures and Forgery Against Adversarial Keys*. Cryptology ePrint Archive, Report 2019/654. `https://ia.cr/2019/654`. 2019.

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

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

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

[16] Thomas Icart. "How to Hash into Elliptic Curves". In: *Advances in Cryptology - CRYPTO 2009*. Springer Berlin Heidelberg, 2009, pp. 303–316.

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

[18] Yehuda Lindell. *Parallel Coin-Tossing and Constant-Round Secure Two-Party Computation*. Tech. rep. Cryptology ePrint Archive, Report 2001/107, 2003. `https://eprint.iacr.org/2003/107`, 2003.

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

[20] Sarang Noether and Brandon Goodell. *Triptych: logarithmic-sized linkable ring signatures with applications*. Cryptology ePrint Archive, Report 2020/018. `https://ia.cr/2020/018`. 2020.

[21] David Pointcheval and Jacques Stern. "Security Proofs for Signature Schemes". In: *Advances in Cryptology, EUROCRYPT '96*. Springer Berlin Heidelberg, 1996, pp. 387–398.

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

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

[24] Patrick P. Tsang et al. *Separable Linkable Threshold Ring Signatures*. Cryptology ePrint Archive, Report 2004/267. `https://ia.cr/2004/267`. 2004.

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

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