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 of constructing a 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 anonymity set size and linear in signer threshold, its verification complexity is quasilinear. 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 lemma-Lin2-Selector lemmas that we formulate and prove herein. The Lin2-Xor lemma itself provides a novel* 2*-round public coin OR-proof protocol, whereas the Lin2-Selector lemma generalizes it to an n-round public coin proof of membership. Consequently, we construct an n-round special honest verifier zero-knowledge proof of membership and instantiate it 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, log-size, 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 possible signers is called an anonymity set or, interchangeably, a ring. It could be required that L signers out of the ring 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 efficient solution to the stated problem can play a role for cryptographic applications, e.g., 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 [25], Abe, Ohkubo, and Suzuki [1], Liu, Wei, and Wong [22], an example of a system that uses linkable ring signatures is, for instance, CryptoNote [28]. 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, 22] 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 [22], 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 [29], Sarang Noether and Brandon Goodell [23], Benjamin E. Diamond [4], Russell W. F. Lai, Viktoria Ronge, Tim Ruffing, Dominique Schröder, Sri Aravinda Krishnan Thyagarajan, and Jiafan Wang [20], William Black and Ryan Henry [5], 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 [16], Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox [18], 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 is out of the scope of our current work.

In this paper we construct a setup-free logarithmic-size linkable ring signature scheme in a prime-order cyclic group without bilinear pairings under the DDH assumption in the ROM. The novelty of this paper is that we present a fairly simple 2-round OR-proof protocol that may be of interest separately, and further derive our signature from it. A notion of OR-proof is given, for example, in the work of Ivan Damgård [9], however, our OR-proof differs from [9], and has some funny properties that we consider here.

Our presentation proceeds as follows. First, we construct our cryptographic protocols and prove their soundness in the language of the theory of computation, basing our proofs on the impossibility of circumventing the DL and DDH assumptions. This style of presentation can be found in early works on cryptography, e.g., by Goldwasser, Micali, Rackoff [14]. It is good because, being rigorous, it does not require knowledge of the methods of modern cryptography, and is therefore more accessible.

Then, to prove the unforgeability of the signature, we realize that it will be easier to do it in the canvas of modern cryptography, namely using the special soundness and, subsequently, witness-extended emulation by Lindell [21], as it is done for example in the works of Liu, Wei, Wong [22], Groth and Kohlweiss [17], Bünz, Bootle, Boneh, Poelstra, Wuille, Maxwell [7]. So we translate all our proofs into this canvas. Thus, our presentation establishes basic properties of protocols using minimal toolkit as in the early days of cryptography, while at the same time setting up the extended properties by involving modern techniques and theorems.

1.1 CONTRIBUTION

1.1.1 LIN2-XOR LEMMA

We formulate and prove Lin2-Xor lemma that allows for committing to exactly one pair of elements out of two pairs of elements, and subsequently proving this commitment is exactly what it is. The Lin2-Xor lemma defines a pure 2-round public coin protocol that, being successfully played between any prover and an honest verifier, convinces the latter 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), $s \in [1, 2]$, is taken from a publicly known set of four elements $\{P_1, Q_1, P_2, Q_2\}$ such that there is no known discrete logarithm relationship between any elements in this set.

By pure protocol we mean a regular cryptographic protocol in which, however, prover's strategy to compute the values it returns to verifier is not defined. Thus, if it is proved that some fact follows from the successful completion of a pure protocol, then it is thereby proved that this fact follows independently of the prover's strategy.

With the Lin2-Xor 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 2-round 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).

1.1.2 LIN2-SELECTOR LEMMA

Using the Lin2-Xor lemma protocol as a disjunction unit, we formulate and prove Lin2-Selector lemma that allows for convincing verifier that given element Z is a commitment to exactly one pair of elements out of many pairs of elements. Namely, the Lin2-Selector lemma provides a pure n-round public coin protocol for convincing verifier of prover's knowledge of the opening (k_0, k_1, s) to the commitment Z such that

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

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

Both of the Lin2-Xor and Lin2-Selector lemmas are proved for a prime-order group under the DL hardness assumption. The amount of data transmitted from prover to verifier during the Lin2-Selector protocol execution is logarithmic in size of the element pair set $\left\{\left(P_j, Q_j\right)\right\}_{j=1}^{N}$, which we name as a decoy set.

1.1.3 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 proof of membership protocol, called L2S. The L2S protocol inherits the properties of the Lin2-Selector lemma pure protocol and thus convinces verifier that the commitment $Z = k_0 P_s + k_1 Q_s$ is built over a member (P_s, Q_s) of the decoy set. 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 MRL2SPoM scheme, which is a non-interactive sHVZK many-out-of-many proof of membership, and consequently construct a logarithmic-size linkable threshold ring signature MRL2SLnkSig, which appears to be anonymous, unforgeable, and non-frameable under DDH in ROM.

Moreover, under the above assumptions the MRL2SLnkSig signature keeps its anonymity, unforgeability, and non-frameability even for the case when the 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.

1.1.4 NOVEL METHOD FOR CONSTRUCTING A LINKABLE RING SIGNATURE

In comparision to the setup-free log-size linkable ring signature schemes proposed in [29, 23, 4, 20], that originate from the ideas of Jens Groth and Markulf Kohlweiss [17] or from the ideas of Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell [7], our signature scheme is constructed on a basis different from [17, 7].

A parallel can be drawn with the work [17], which introduced a mechanism similar to the Kronecker's delta to select a member of anonymity set without revealing it. Our signature uses the Lin2-Xor and, consequently, Lin2-Selector lemmas in exactly the same role. However, there is a difference in the anonymity set constructions: the anonymity sets in [17] are scattered in a plane of two orthogonal generators, while the anonymity sets for the Lin2-Selector lemma protocol are orthogonal generator sets themselves.

Thus, the Lin2-Xor lemma provides a new cryptographic primitive that can be used to construct a ring signature and, probably, 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 P and Q in the equation

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

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

The lemma states that if no discrete logarithm relationship between P and Q is known, if 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 the following linear combination R of four fixed prime-order group elements P_1, Q_1, P_2, Q_2 with unknown discrete logarithm relationship between them

$$R = c_{20}\left(c_{10}P_1 + c_{11}Q_1\right) + c_{21}\left(c_{12}P_2 + c_{13}Q_2\right),$$

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

$$R = \left(P_1 + c_{11}Q_1\right) + c_{21}\left(P_2 + c_{13}Q_2\right).$$

It appears to be 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 equality

$$wR = Z + r_1H_1 + r_2H_2$$

holds for some secretly known nonzero scalar w, then Z is equal 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 implication 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, which becomes the lemma's protocol, ends successfully only if 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, 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 the four elements $Z = aP_1 + bQ_1 + dP_2 + eQ_2$ with known to the prover nonzero 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 the similar property

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

where f, g are known to the prover. Moreover, as another corollary, if the above 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

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

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, respectively.

In this game, Lin2-Selector lemma convinces verifier that Z is exactly one of $(aP_1 + bQ_1)$, $(aP_2 + bQ_2)$, $(aP_3 + bQ_3)$, $(aP_4 + bQ_4)$ for some known to prover a, b. Here is a core idea of how we can prove this. By applying one of the Lin2-Xor lemma corollaries, it becomes proved 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. By applying the Lin2-Xor lemma to the equality that holds, suppose, to the first one, it becomes proved that Z is exactly one of $(aP_1 + bQ_1)$, $(aP_2 + bQ_2)$ for some a, b known to the prover. The same applies for the case when the second equality holds.

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

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

where the verifier is convinced that $Z = k_0P_s + k_1Q_s$ for some known to prover scalars k_0, k_1, and index $s \in \left[1, 2^{n-1}\right]$. The actual s is made indistinguishable by keeping the scalars k_0 and k_1 in secret and letting k_0 be distributed uniformly. This reduction method resembles the n-round reduction by B. Bünz et al. [7], however, these reductions are different and neither one of them seems to be convertible to the other.

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

A pure protocol specifies in detail what verifier should do, however it does not specify the same for 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 certain pure protocols, namely, that the protocol successful completion implies that 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's strategy, including dishonest ones, verifier is convinced that the prover knows secret values after the successful completion of the corresponding pure protocol.

Completeness and zero-knowledge cannot be considered for pure protocols, since these properties depend on how prover prepares 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 prover's behavior is fully defined in a derived protocol, we begin to consider its completeness and zero-knowledge.

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 a 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 both 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 then verifier is assumed honest, whereas 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 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 initially knowing the secret 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

We prove that the protocols defined in the Lin2, Lin2-Xor, and Lin2-Selector lemmas are sound. In the soundness proofs we first and foremost use logical inference based on the impossibility to circumvent the DL assumption. This inference is non-trivial, therefore, to make it easier to write down the inference chains, we introduce a tiny symbolic logic system, which partially formalizes the usual way of reasoning and getting system properties from DL.

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

The Lin2-WEE, Lin2-Xor-WEE, and Lin2-Selector-WEE lemmas are subset cases of their former counterparts, they translate the lemmas into the language of a polynomial-time relation satisfaction and impose some few additional requirements to the protocols. After this change, the long logical inference chains in the Lin2, Lin2-Xor, and Lin2-Selector lemma proofs are replaced by witness extraction algebra. Although, the latter often resembles the former logical inference. Having the WEE property of the Lin2-Selector lemma protocol proved, we prove unforgeability of our signature by adapting the methods from the works of Jens Groth and Markulf Kohlweiss [17] and Joseph K. Liu, Victor K. Wei, and Duncan S. Wong [22].

Another question is why did we leave the former 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 method that led us to the signature scheme. This method seems rather intuitive, allowing construction of a cryptographic scheme in a direct way. Namely, informally, it can be observed that the logical inference we use may often be interpreted geometrically, taking all orthogonal elements in a protocol for the basis of a multidimensional linear space where finding inner product of two elements is assumed hard.

1.2.6 L2S MEMBERSHIP PROOF, MRL2SLNKSIG SIGNATURE

We construct L2S proof of membership protocol on top of the Lin2-Selector 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 the latter from the Lin2-Selector-WEE lemma.

Then we analyze the L2S protocol transcript and show that all its records have distributions indistinguishable from the 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 on top of it.

The L2S protocol 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 $(2 \dots 4)S_{sz}/\log_2(S_{sz})$ 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 [12, 24], 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, which has 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 nonzero element A is a generator of $\mathbb{G}$, hence we assume G is an element chosen in advance.

2.1 A NOTE ABOUT CONTEXT

All definitions and lemmas herein 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 between themselves, so that Verifier becomes gradually convinced of the facts. A protocol can be translated into corresponding non-interactive scheme using the Fiat-Shamir heuristic in ROM [12, 24]. We start by proving our lemmas in the interactive setting, then turn them into non-interactive using this heuristic.

2.2 DEFINITIONS

2.2.1 SECURITY PARAMETER AND CRS

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

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

2.2.2 SETS AND VECTORS

Sets are assumed having cardinalities that are polynomial in $\boldsymbol{\lambda}$ 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 this is not ambiguous, e.g., if $S = \{B_1, B_2, \dots, B_n\}$, 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 nonzero 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 nonzero elements $S = \{B_1, B_2, \ldots, B_n\}$, we say a discrete logarithm relation of A to S is known iff the scalars $x_1, x_2, \ldots, x_n$ in the equation

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

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 proved 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 nonzero 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 nonzero 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 $\mathbb{G}$ everytime. When we prove zero-knowledge, we assume that DDH holds for $\mathbb{G}$.

2.2.5 SHORTHANDS FOR 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 the statement

$$A \sim B$$

denotes the fact of knowing the discrete logarithm relation between A and B. That is, if we write $A \sim B$, then x in the equality $A = xB$ is assumed or has been shown to be known.

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:

- $\neg(A \sim B \wedge A \mathrel{!}\sim B)$, meaning that it's not possible for a discrete logarithm relationship to be simultaneously known and unknown, that is, such a situation is a contradiction.
- $\neg(A \sim B) \Rightarrow A \mathrel{!}\sim B$, 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 discrete logarithms.

Thus, 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

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

A typical way of obtaining new statements using this symbolic logic system is to make a supposition $A \sim B$, to check if this leads to a contradiction, and, if a contradiction is found, to obtain the statements $\neg(A \sim B)$ and $A \mathrel{!}\sim B$.

Note that nothing can be obtained from the supposition $A \,!\sim B$. We have no need to go deeper into the properties of this symbolic system in this paper.

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

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

the fact of knowing the discrete logarithm relation of A to $\{B_1, B_2, \ldots, B_n\}$, namely, the fact of knowing $[x_i]_{i=1}^n$ such that $A = \sum_{i=1}^n x_i B_i$ holds.

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

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

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

- $\neg(\, A = \text{lin}\,(B_1, B_2, \ldots, B_n) \;\wedge\; A \,!= \text{lin}\,(B_1, B_2, \ldots, B_n)\,)$
- $\neg(A = \text{lin}\,(B_1, B_2, \ldots, B_n)) \;\Rightarrow\; A \,!= \text{lin}\,(B_1, B_2, \ldots, B_n)$

Also, according the above denotations, for any elements A and B such that $B \neq 0$

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

2.2.6 ORTHOGONAL SETS

For any set $S = \{B_1, B_2, \ldots, B_n\}$ of nonzero elements, if the following holds for each element $B_i \in S$: $B_i \,!= \text{lin}\,(S \backslash \{B_i\})$, then we denote this fact as

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

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

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

This definition of orthogonality corresponds to the definition of Discrete Log Relation, which is also taken as an assumption, in [7]. Namely, for a set S of randomly sampled elements of $\mathbb{G}$, the Discrete Log Relation assumption states $\text{ort}(S)$. We accept this assumption and use it in a scenario where S is a set of the hash to curve function $\mathbf{H}_{\mathbf{point}}$ images for different pre-images. By the Discrete Log Relation assumption $\text{ort}(S)$ holds 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 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 pair (s, c) representing Schnorr signature [26], 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

- $(A \sim B \text{ and } C \,!\sim D)$, 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,
- $(A \sim B \text{ and } C \,!\sim D)$ 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,
- 'something' is an evidence of $(A \sim B \text{ and } C \,!\sim D)$, when 'something' is a legally verifiable by Verifier proof of 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 [7] or in [21]. However, the "argument of knowledge" is a stricter term, as e.g. by definition in [7] 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 [21] in that the latter requires existence of a knowledge extractor. In any case, as shown, e.g. in [7], "knowledge soundness" implies "soundness".

When an evidence is sent from Prover to Verifier in a protocol, if it fails verification on the Verifier's side, then the protocol terminates with the 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 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 a statement is proven to hold only with negligible probability, we say the statement does not hold.

The same applies to contradictions. If we have an assumption and its implication such that the implication holds only with negligible probability, then we get a contradiction. For example, the following logical chain leads to a contradiction: (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 anonymity set as a decoy set. It is assumed that at least one entry of a decoy set belongs to an actual signer. We don't limit the actual signer to owning only one entry in the set, it may own all decoys.

An adversary may own any number of entries in the 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.

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

We use the terms "ring" and "set" as the synonyms to the "decoy set" in the following sense. "Decoy set" is usually a part of a low-level protocol; "set" is used when talking about the set membership proofs; "ring" is related to the 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

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

The scalars in the sum are sometimes called "weights". Nevertheless, they don't have 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

We call a value unique under certain conditions, when this value is known and also it is possible to efficiently calculate no more than one value that satisfies these conditions.

We call a vector unique under certain conditions, when it is known and also it is possible to efficiently calculate no more than one vector that satisfies these conditions. Namely, a known vector is unique when it is hard to calculate a different vector that satisfies the same conditions. Two vectors are called different if they have at least one position with different items.

For instance, the statement

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

means that 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 COMMITMENT

We use the term "commitment" in the same sense as it is used in [7], always considering commitment as binding. At the same time, as in [7], our commitments may or may not be hiding.

Formally, our commitments have the commitment space C_{pp} equal to $\mathbb{G}$. For an not-hiding commitment the randomness space R_{pp} is $\{0\}$. A simple hiding commitment is the Pedersen commitment with $\mathsf{M}_{\mathsf{pp}} = \mathbb{F}$, $\mathsf{R}_{\mathsf{pp}} = \mathbb{F}$.

The more elaborated Com2 commitment in Section 6.1 can be considered as having the message space $\mathsf{M}_{\mathsf{pp}} = \mathbb{I} \times \mathbb{F}$, where $\mathbb{I} \subset \mathbb{Z}$ is an index interval, $\mathsf{R}_{\mathsf{pp}} = \mathbb{F}$, and pp contains $|\mathbb{I}|$ pairs of orthogonal generators such that the commitment algorithm Com2 returns a Pedersen commitment over a generator pair at input index $s \in \mathbb{I}$.

2.2.16 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. CRS is assumed to be silently presented in $\mathcal{R}$ as the third argument.

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 $(u, w) \in \mathcal{R}$.

2.2.17 WITNESS-EXTENDED EMULATION

For the computational witness-extended emulation (also called simply witness-extended emulation, abbreviated as WEE) we use the definition from [7]. 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 to be equipped with an oracle that allows rewinding the conversation to a specific move and resuming it with a fresh randomness from that move onwards.

It should be noted that the definition in [7] is for protocols with specified Prover's behavior; we have adopted this definition to pure protocols without any loss. In fact, the definition in [7] 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 described and used in [7], a protocol has WEE if there exists a PPT extractor that finds 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 rewindable conversation or from a transcript tree indicates that 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 witness.

2.2.18 SPECIAL HONEST VERIFIER ZERO-KNOWLEDGE

We define special honest verifier zero-knowledge (sHVZK) the same way as in [7]. A protocol is sHVZK if there exists a PPT simulator capable of producing successful protocol transcripts, 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 [8] to n-round protocols.

3 PRELIMINARY LEMMAS

Lemma 1 (NotLin)**:**
For any three nonzero elements A, B, C such that $A \,!= \text{lin}\,(B, C)$*, the following three statements hold*

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

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

c) Both hold: $A \,!\sim B$ *and* $A \,!\sim C$

Proof.

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

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

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(), $A = \text{lin}\,(B, C) \Rightarrow$ Contradiction) $\Rightarrow A \,!\sim B$. Likewise, $A \,!\sim C$.

□

Lemma 2 (OrtUniqueRepresentation)**:**
For any element A and any vector $\vec{B} = [B_i]_{i=1}^n$ *of nonzero elements: if* $\text{ort}\left(\vec{B}\right)$ *and* $A = \text{lin}\left(\vec{B}\right)$*, 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 nonzero scalar.

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

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

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

Lemma 3 (OrtReduction)**:**
For any set of nonzero elements S, any two elements $B_j, B_k \in S$*, any two nonzero scalars a, b, the following holds*

$$\text{ort}\,(S) \Rightarrow \text{ort}\left(\left\{\left(aB_j + bB_k\right)\right\} \cup \left(S \backslash \left(\left\{B_j\right\} \cup \{B_k\}\right)\right)\right).$$

Proof. For $j = k$, the conclusion immediately follows from the definition of ort(). For $j \neq k$, suppose the opposite, that is, $\left(aB_j + bB_k\right) = \text{lin}\left(S \backslash \left(\left\{B_j\right\} \cup \{B_k\}\right)\right) \Rightarrow$ moving B_k to the right: $aB_j = \text{lin}\left(S \backslash \left\{B_j\right\}\right) \Rightarrow$ dividing by a: $B_j = \text{lin}\left(S \backslash \left\{B_j\right\}\right) \Rightarrow$ Contradiction to the definition of ort (S). □

Lemma 4 (ZeroRepresentation)**:**
For any $\vec{B} = [B_i]_{i=1}^n$ *and any* $\vec{x} = [x_i]_{i=1}^n$*, if* $\text{ort}\left(\vec{B}\right)$ *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 $\vec{x} = \vec{y} = \vec{0}$. □

Lemma 5 (OrtDisjunction)**:**
For any set of nonzero elements S, any vector of subsets $[S_i \mid S_i \subset S]_{i=1}^n$ *such that for any* $j, k \in [1, n]$*,* $j \neq k$ *there holds* $S_j \cap S_k = \emptyset$*, for any vector of nonzero elements* $[Y_i \mid Y_i = \text{lin}\,(S_i)]_{i=1}^n$*, the following holds*

$$\text{ort}\,(S) \Rightarrow \text{ort}\left([Y_i]_{i=1}^n\right).$$

Proof. Suppose the opposite, that is, by definitions of ort() and lin() there is a vector of known scalars $[x_i]_{i=1}^n$ such that at least one x_i is nonzero and the weighted sum of $[Y_i]_{i=1}^n$ with weights $[x_i]_{i=1}^n$ is zero

$$0 = \sum_{i=1\ldots n} x_i Y_i.$$

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 nonzero . This contradicts the ZeroRepresentation lemma. Thus, ort $\left([Y_i]_{i=1}^n\right)$.

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

Lemma 6 (OrtHalfShift)**:**
For any two vectors of nonzero 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 nonzero element* F *such that* $F \,!= \text{lin}(S)$, *the following holds*

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

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 nonzero 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 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 $F = \text{lin}(S)$ that contradicts $F \,!= \text{lin}(S)$. Thus, ort$([X_i]_{i=1}^m \cup [Y_i + F]_{i=1}^n)$. □

Lemma 7 (Lin2)**:**
For any four nonzero fixed elements P, Q, Z, H such that $P \,!\sim Q$, *the following protocol (Table 1) is an evidence of*

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

Table 1: Lin2 lemma protocol.

	← Verifier picks a nonzero random scalar c and sends it to Prover
Prover returns a nonzero scalar r and an evidence of $(P + cQ) \sim (Z + rH)$ →	Verifier checks $(Z + rH) \neq 0$, $r \neq 0$, and also 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 $(P + cQ) \sim (Z + rH)$ holds for Prover, and also that $(Z + rH) \neq 0$, $r \neq 0$. Hence, it is convinced that Prover knows t such that

$$P + cQ = tZ + trH \tag{1}$$

Suppose, $t = 0 \Rightarrow P + cQ = 0 \Rightarrow P \sim Q \Rightarrow$ Contradiction to $P \,!\sim Q$. Hence, $t \neq 0$.
Finding Z from the equality (1)

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

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 helper lemmas, which has been already proved above. We also prove two useful corollaries of the main lemma. Moreover, we show that under a bit stronger premise the lemma protocol admits witness extraction.

4.1 LIN2-XOR LEMMA

Lemma 8 (Lin2-Xor)**:**
For any four nonzero fixed elements P_1, Q_1, P_2, Q_2 such that ort (P_1, Q_1, P_2, Q_2)*, for any two nonzero fixed elements Z, H_1, the following protocol (Table 2) is an evidence of*

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

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

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

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

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

Table 2: Lin2-Xor lemma protocol.

	Verifier picks two nonzero random scalars c_{11}, c_{13} and sends them to Prover ←
Prover returns a nonzero scalar r_1 and a nonzero element H_2 →	Verifier checks $(Z + r_1H_1) \neq 0$, $r_1 \neq 0$, $H_2 \neq 0$
	Verifier picks a nonzero random scalar c_2 and sends it to Prover ←
Prover returns a nonzero scalar r_2 and an evidence of $(P_1 + c_{11}Q_1 + c_2P_2 + c_2c_{13}Q_2) \sim (Z + r_1H_1 + r_2H_2)$ →	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
$(Z + r_1H_1) = \text{lin}\,(P_1 + c_{11}Q_1, P_2 + c_{13}Q_2)$	$Z = \text{lin}\,(P, Q)$

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 nonzero .

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 nonzero .

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

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

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

$$\begin{cases} k_1 = \left(r_1a' - r'_1a\right)/\left(r_1 - r'_1\right) \\ k_2 = \left(r_1a'c'_{11} - r'_1ac_{11}\right)/\left(r_1 - r'_1\right) \\ k_3 = \left(r_1b' - r'_1b\right)/\left(r_1 - r'_1\right) \\ k_4 = \left(r_1b'c'_{13} - r'_1bc_{13}\right)/\left(r_1 - r'_1\right) \end{cases} \tag{13}$$

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 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 nonzero , as the opposite contradicts the premise of nonzero 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 nonzero , 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 $r'_1 \neq 0$

$$\begin{aligned} \left(r_1 - r'_1\right) k_1 = \left(r_1 a' - r'_1 a\right) \quad &\Rightarrow \\ r_1 \left(a' - k_1\right) = r'_1 \left(a - k_1\right) \quad &\Rightarrow \\ \left(a' - k_1\right) / r'_1 = \left(a - k_1\right) / r_1 \quad & \end{aligned} \tag{16}$$

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

$$\begin{aligned} \left(a' - k_1\right) / r'_1 = q \quad \text{and} \quad \left(a - k_1\right) / r_1 = q \quad &\Rightarrow \\ a' = q r'_1 + k_1 \quad \text{and} \quad a = q r_1 + k_1 \quad & \end{aligned} \tag{17}$$

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

$$\begin{aligned} t \left(r_1 a' - r'_1 a\right) = \left(r_1 a' c'_{11} - r'_1 a c_{11}\right) \quad &\Rightarrow \\ r'_1 a \left(c_{11} - t\right) = r_1 a' \left(c'_{11} - t\right) \quad &\Rightarrow \\ a \left(c_{11} - t\right) / r_1 = a' \left(c'_{11} - t\right) / r'_1 \quad & \end{aligned} \tag{18}$$

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

$$\begin{aligned} a \left(c_{11} - t\right) / r_1 = v \quad \text{and} \quad a' \left(c'_{11} - t\right) / r'_1 = v \quad &\Rightarrow \\ v r_1 = a \left(c_{11} - t\right) \quad \text{and} \quad v r'_1 = a' \left(c'_{11} - t\right) \quad & \end{aligned} \tag{19}$$

The constant v is nonzero , 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 $v r_1$

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

Note we have performed division by the expressions $(v - q(c_{11} - t))$ and $(c_{11} - t)$ above, as both they are nonzero 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}

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

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 nonzero . 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 proved. □

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

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

By the OrtUniqueRepresentation lemma, as $\mathrm{ort}(P_1, Q_1, P_2, Q_2)$ holds, from the equality (25) we have

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

$$z_2 = ac_{11}\,, \tag{27}$$

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 $(Z = \mathrm{lin}\,(P_1, Q_1) \wedge H_1 = \mathrm{lin}\,(P_2, Q_2))$ holds for Prover is wrong.

Likewise, the case of $(Z = \mathrm{lin}\,(P_2, Q_2) \wedge H_1 = \mathrm{lin}\,(P_1, Q_1))$ 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

$$\begin{pmatrix} Z = \mathrm{lin}(P_1, Q_1) & \wedge \\ H_1 = \mathrm{lin}(P_1, Q_1) & \wedge \\ ((Z + r_1H_1) \sim (P_1 + c_{11}Q_1)) & \wedge \\ ((Z + r_1H_1)\,!\sim(P_2 + c_{13}Q_2)) & \end{pmatrix} \oplus \begin{pmatrix} Z = \mathrm{lin}(P_2, Q_2) & \wedge \\ H_1 = \mathrm{lin}(P_2, Q_2) & \wedge \\ ((Z + r_1H_1) \sim (P_2 + c_{13}Q_2)) & \wedge \\ ((Z + r_1H_1)\,!\sim(P_1 + c_{11}Q_1)) & \end{pmatrix}$$

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

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

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

$$(Z + r_1H_1) = \mathrm{lin}\,(P_1, Q_1) \;\text{ for Prover.} \tag{28}$$

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

$$(Z + r_1H_1) = a\,(P_1 + c_{11}Q_1) + b\,(P_2 + c_{13}Q_2)\,. \tag{29}$$

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

$$(Z + r_1H_1) \sim (P_1 + c_{11}Q_1) \;\text{ for Prover.} \tag{30}$$

Suppose, $(Z + r_1H_1) \sim (P_2 + c_{13}Q_2)$ holds for Prover simultaneously with the expression (30). This contradicts the OrtUniqueRepresentation lemma, as the element $(Z + r_1H_1)$ gets two representations: $a\,(P_1 + c_{11}Q_1)$ and $b\,(P_2 + c_{13}Q_2)$, where a and b are known to Prover. Hence, $(Z + r_1H_1)\,!\sim (P_2 + c_{13}Q_2)$ for Prover.

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

$$\begin{pmatrix} Z = \mathrm{lin}(P_1, Q_1) & \wedge \\ H_1 = \mathrm{lin}(P_1, Q_1) & \wedge \\ ((Z + r_1H_1) \sim (P_1 + c_{11}Q_1)) & \wedge \\ ((Z + r_1H_1)\,!\sim(P_2 + c_{13}Q_2)) & \end{pmatrix}$$

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

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

$$\begin{pmatrix} Z = \mathrm{lin}(P_2, Q_2) & \wedge \\ H_1 = \mathrm{lin}(P_2, Q_2) & \wedge \\ ((Z + r_1H_1) \sim (P_2 + c_{13}Q_2)) & \wedge \\ ((Z + r_1H_1)\,!\sim(P_1 + c_{11}Q_1)) & \end{pmatrix}$$

holds, when $(Z = \mathrm{lin}\,(P_2, Q_2) \wedge H_1 = \mathrm{lin}\,(P_2, Q_2))$ holds for Prover. □

Thus, this corollary is proved.

4.3 WITNESS EXTRACTION

Now we will reformulate the Lin2, Lin2-Xor lemmas and their corollaries, so that in addition to soundness we will obtain 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 will assume that the evidence is provided by means of another protocol, which is played immediately after Verifier checks that $(Z + rH) \neq 0, r \neq 0$ hold.

Thus, the Lin2 lemma protocol in Table 1 rewrites as the protocol in Table 4, which is now 1-round protocol followed by some n-round evidence protocol for $(P+cQ) \sim (Z+rH)$. Assuming that the n-round evidence protocol has witness-extended emulation, we will show that the protocol in Table 4 also has witness-extended emulation.

Lemma 9 (Lin2-WEE)**:**
For any three nonzero fixed elements P, Q, Z such that $P \not\sim Q$, 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 nonzero element H	→ Verifier checks $H \neq 0$
	← Verifier picks a nonzero random scalar c and sends it to Prover
Prover returns a nonzero scalar r	→ 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 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 in Table 2, let's figure out how we can rewrite it similarly to rewriting the Lin2 lemma protocol in 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 the Lin2-WEE lemma protocol first message) might be played.

The Lin2 lemma protocol (Table 1) with the substitutions (Table 3) is played in the Lin2-Xor lemma protocol (Table 2) second round only for the sake of obtaining an evidence of (9). Hence, we can relax the requirement of using exactly the Lin2 lemma protocol, instead requiring to use any protocol that provides an evidence of (9) and has WEE.

In sum, the rewritten protocol is shown in Table 5. Compared to the Lin2-Xor lemma protocol (Table 2), it has the stronger precondition in the sense that its sub-protocol providing an evidence of (9) is now required to have WEE, and, at the same time, the precondition is weakened in the sense that any WEE protocol providing an evidence of (9) will suffice.

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

Lemma 10 (Lin2-Xor-WEE)**:**
For any five nonzero fixed elements P_1, Q_1, P_2, Q_2, Z such that 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 nonzero element H_1 →	Verifier checks $H_1 \neq 0$
	← Verifier picks two nonzero random scalars c_{11}, c_{13} and sends them to Prover
Prover returns a nonzero 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.) →	Verifier checks $(Z + r_1H_1) \neq 0$, $r_1 \neq 0$
Prover and Verifier play a WEE protocol of the evidence $(Z + r_1H_1) = \text{lin}(P_1 + c_{11}Q_1, P_2 + c_{13}Q_2)$. (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 $(Z + r_1H_1) = \text{lin}(P_1 + c_{11}Q_1, P_2 + c_{13}Q_2)$ for Prover.

By definition of WEE, properly unwinding the game for the evidence $(Z + r_1H_1) = \text{lin}(P_1 + c_{11}Q_1, P_2 + c_{13}Q_2)$, 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 nonzero , then 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 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 nonzero 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

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

$$\begin{pmatrix} H_1 = uP_1 + vQ_1 \,, \\ u = h_1 \,, \\ v = h_2 \end{pmatrix} \oplus \begin{pmatrix} H_1 = uP_2 + vQ_2 \,, \\ u = h_3 \,, \\ v = h_4 \end{pmatrix} . \tag{59}$$

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)

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

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 represent 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 a tree structure shown in Figure 1, where we renamed P_1, Q_1, P_2, Q_2 as X_0, X_1, X_2, X_3. Let this tree structure be evaluated to R recursively, each node performing summation and each arrow performing multiplication by its tag. If all arrow tags are known, then R represents a multi-exponent sum of four summands.

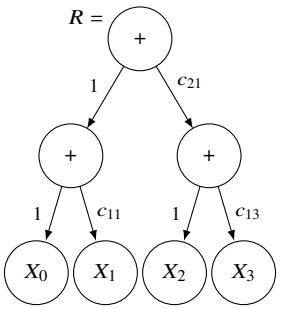

Figure 1: Rsum for four elements.

By generalizing this tree structure, we have, for example, for a set of sixteen elements $\left[X_j\right]_{j=0}^{15}$, the tree structure shown in Figure 2, which represents 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

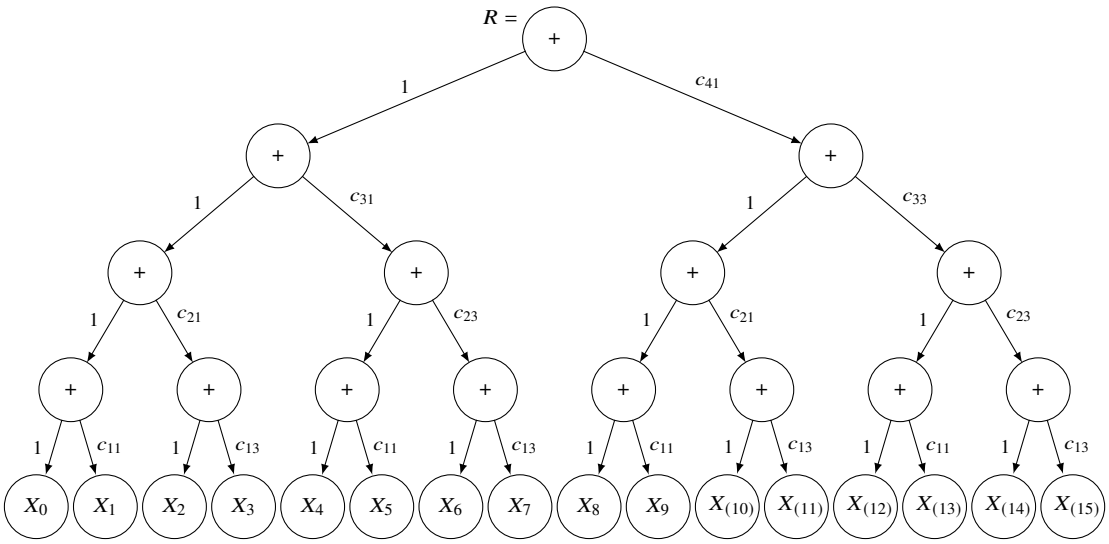

Figure 2: Rsum for sixteen elements.

scalar c_{n1}, let Rsum $\left(n, N, \left[X_j\right]_{j=0}^{N-1}, [(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_{n1}\right)$ be an element, such that:

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

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

Note that we have changed the beginning of indexing of the decoy set elements from 1 to 0, this is done for convenience of further presentation. All formulations of already proved lemmas can be easily translated to indexing from 0, we will use them in this form.

Lemma 11 (Lin2-Selector)**:**
For any $n > 1$ and $N = 2^n$, for any vector of nonzero 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 nonzero fixed element Z, for a vector of n nonzero elements $[H_i]_{i=1}^{n}$ where H_1 is fixed, and for a vector of nonzero 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 where the challenge pair (c_{i1}, c_{i3}), $i = 1$, is sampled.

Since

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

Table 6: Lin2-Selector lemma protocol.

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

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.

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

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

that is, Verifier has 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 $i = m$:

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

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

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

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

of four Rsum's Y_0, Y_1, Y_2, Y_3:

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

where:

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

By the Rsum property,

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

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

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

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

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

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

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

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

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

then it accepts that

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

and, then, the protocol of the Lin2-Xor lemma is successfully completed, and, by the Corollary 3 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

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

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

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

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

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

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

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

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

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

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

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

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

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

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.

Lemma 12 (Lin2-Selector-WEE)**:**
For any $n > 1$ and $N = 2^n$, for any vector of nonzero 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 nonzero fixed element Z, for a vector of n nonzero elements $[H_i]_{i=1}^{n}$ where H_1 is fixed, for a vector of nonzero 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 an evidence for the* Rsum *in the last step of the protocol has witness-extended emulation.*

Proof. The element H_1 is considered as the first message from Prover to Verifier in the protocol (Table 6). For this protocol we will prove existence of a PPT emulator that meets 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

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

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 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 when the challenge pair (c_{m1}, c_{m3}) is generated, the rest of the protocol meets 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 by 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 the witness (x, y, s) for Z in the statement (65), that is, finds 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) by simply translating the index s. Thus, we have proven this lemma for $n = (m + 1)$ and, by induction, the lemma is proven. □

6 L2S MEMBERSHIP PROOF

Now we will 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 will prove that the **L2S** protocol is complete and sound. Also we will prove that it has witness-extended emulation, is special honest verifier zero-knowledge, and, consequently, that 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 $s \in [0, N/2 - 1]$, we define Com2 $\left(k_0, k_1, s, \vec{X}\right)$ as follows

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

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

By the OrtUniqueRepresentation lemma, if (k_0, k_1, s) is an opening of Com2 commitment Z over $\vec{X}$, then the opening (k_0, k_1, s) is unique. Therefore, by the definition of binding from [17], the Com2 commitment is strongly binding. Also, Com2 commitment becomes hiding when at least one of k_0, k_1 is picked independently and uniformly at randon, as Com2 turns into Pedersen commitment in this case.

When the vector $\vec{X}$, Com2 commitment Z over $\vec{X}$, and scalars k_0, k_1 of Z's opening are known, it's possible to efficiently calculate the index s by iterating through $\vec{X}$ and checking if $Z = k_0 X_{2s} + k_1 X_{2s+1}$. Nevertheless, s is bound to Z and, thus, Z commits to a member-pair in $\vec{X}$ at index s. Hence, we call Com2 as a commitment to a member-pair. Also, as we talked about before, the set of member-pairs $\vec{X} = \left[X_j\right]_{j=0}^{N-1}$, $N = 2^n$, 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 $\mathrm{ort}(\vec{X})$ 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 $(\,(k_0, k_1, s), Z\,)$ such that k_0 is nonzero and chosen uniformly at random, k_1 is arbitrary, $s \in [0, N/2 - 1]$, and $Z = \mathrm{Com2}(\,k_0, k_1, s, \vec{X}\,)$. For any **ComGen** implementation choice, the independence and random uniformity of the k_0 distribution together with the conditions $Z = \mathrm{Com2}(\,k_0, k_1, s, \vec{X}\,)$ 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 the tuple (67) that Verifier has upon completion of **InteractionProcedure** along with the decoy set from **DecoySetGen**. It returns 1 or 0, which means the verification succeeded or failed.

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

- Decoy set $\vec{X}$ is generated at both Prover's and Verifier's sides, using same **L2S.DecoySetGen**.
- Prover gets 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 (67) obtained above. Iff it returns 1, then the **L2S** protocol is completed successfully. We will prove that the successful completion means $Z = \mathrm{Com2}(\,k_0, k_1, s, \vec{X}\,)$.

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

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

We use the *InvertLastBit* for the binary tree indices, 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

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

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

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

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

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

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

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

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

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

When $i = 3$, the expression

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

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

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

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

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

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

When $i = 1$, Prover sends to Verifier

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

so that $(Z + r_1 H_1) = w\,\text{Rsum}\left(1, 2, [X_t]_{t=2s}^{2s+1}, [], c_{1,((2s+1)\%4)}\right)$.

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

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

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

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

so that

$$\begin{aligned} (Z + r_1 H_1 + r_2 H_2) = w\,\text{Rsum}&\left(1, 2, [X_t]_{t=2s}^{2s+1}, [], c_{1,((2s+1)\%4)}\right) + \\ w(c_{2,(h\%4)}/c_{2,(s\%4)})\,\text{Rsum}&\left(1, 2, [X_t]_{t=2h}^{2h+1}, [], c_{1,((2h+1)\%4)}\right) = \\ w\,\text{Rsum}&\left(2, 4, [X_t]_{t=4((s//2)+1)-1}^{4((s//2)+1)-1}, \left[\left(c_{d,1}, c_{d,3}\right)\right]_{d=1}^{1}, c_{2,((2(s//2)+1)\%4)}\right)/c_{2,(s\%4)} \end{aligned}$$

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

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

When $i = 3$, Prover sends

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

so that

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

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 RETURNS 1

The (T, c, t) part of the **L2S.Verif** implementation (Table 9) input is the Schnorr identification scheme [26] 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 [26] 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 proved in 6.2.2, if Z at the Verifier's input is equal to $\mathrm{Com2}\left(k_0, k_1, s, \left[X_j\right]_{j=0}^{N-1}\right)$ such that 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) meet the Lin2-Selector lemma protocol (Table 6). Namely, **L2S.InteractionProcedure** and **L2S.Verif** populate the Lin2-Selector lemma pure protocol with the concrete implementation of Prover's behavior. Therefore, the Lin2-Selector lemma can be applied to the **LS2** protocol.

As shown in 6.2.2, if **L2S.Verif** returns 1, then $(tW + cR) == T$, and, since the Schnorr identification scheme [26] is sound, Verifier has an evidence of $W \sim R$, that is, an evidence of

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

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 $\left(X_{(2s)}, X_{(2s+1)}\right)$, $s \in [0, N/2 - 1]$. Using the definitions of lin() and Com2, this implies that Verifier is convinced that Prover knows 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 have shown that the Lin2-Selector lemma applies to the **LS2** protocol. In addition to the requirements imposed by the Lin2-Selector lemma, the Lin2-Selector-WEE lemma from 5.3 requires a witness-extended emulator to exist for a sub-protocol used for the evidence in the last step of the protocol in Table 6. That is, considering possible application of the Lin2-Selector-WEE lemma to the **LS2** protocol, the lemma requires a witness-extended emulator to exist for an evidence of $W \sim R$.

The evidence of $W \sim R$ is implemented with the Schnorr identification scheme, as we noted in 6.2.2. Since the Schnorr identification scheme is special sound, according to the definitions of witness-extended emulation and special soundness (the latter is a subset case of the former), a witness-extended emulator exists for the evidence of $W \sim R$. 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 tuple (67), let's copy it 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 elements W and R are calculated from the publicly visible elements and scalars

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

Thus, the element T is a combination of the variables seen for everyone. Hence, we exclude T from our consideration: for any transcript accepted by Verifier the item T carries no information, it 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, since each r_i is obfuscated by the private multiplier q, which is sampled anew for each r_i.

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

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

Using the terms introduced in [6], the $r_i H_i$'s and Z are linearly independent degree 2 polynomials of a private set of the independent and random uniform scalars

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

The coefficients of these polynomials are efficiently computable from the $[(c_{i1}, c_{i3})]_{i=1}^{n-1}$, c_n, and k_1. Thus, reducing the question of the $r_i H_i$'s distributions to the (P, Q)-DDH problem [6], we have

$$P = \left\{ \left[X_j\right]_{j=0}^{N-1} \right\} \text{ and } Q = \left\{ \{Z\} \cup \{r_i H_i\}_{i=1}^{n} \right\}$$

and, also, as P are all degree-1 polynomials, whereas Q are all degree-2,

$$\text{Span}\,(P) \cap \text{Span}\,(Q) = \emptyset.$$

Under the (P, Q)-DDH assumption [6], the distributions of all the $r_i H_i$'s and Z are indistinguishable from $\{e_i G\}_{i=1}^{n+1}$, where all the e_i's are independent and uniformly random.

As the DDH assumption implies (P, Q)-DDH [6], for our polynomials in the above sets P and Q, under DDH, we have all the $r_i H_i$'s and Z distributed indistinguishable from independent and having random uniform distribution elements. Thus, we have proved independency and random uniformity of all $r_i H_i$'s and Z's in honest conversation transcripts between Prover and Verifier over any fixed decoy set $\left[X_j\right]_{j=0}^{N-1}$ generated by **L2S.DecoySetGen**.

For readability, we will be omitting the word 'indistinguishable' further for this and similar cases. It will be always implied.

As the decoy sets can vary and can be reused for different conversations, we have to consider the case of all honest conversation transcripts over all really used and possibly intersecting decoy sets. In this case, we reduce the

question to the same (P, Q)-DDH problem with

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

with the private set of the independent and random uniform scalars

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

By requiring w to be chosen independently and uniformly at random for each transcript, which also implies Z is never used in any two different conversations, we have P containing only degree-1 polynomials, and Q only degree-2's. Thus,

$$\text{Span}\,(P) \cap \text{Span}\,(Q) = \emptyset,$$

and, therefore, all the $r_i H_i$'s and Z's publicly seen across all the accepted transcripts are distributed independently and uniformly at random under DDH. Their distributions are independent of each other and of the distributions of the elements X_j's of decoy sets.

In sum, we have got to the conclusion that all items, except for the items T, of all ever recorded honest **L2S** conversation transcripts have uniformly random and independent distributions under the DDH, provided that the input commitments Z are never reused, that is, generated anew with **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; T's serve only to distinguish the acceptable transcripts from the ones 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 [7]. Having the random independence property proved 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 samples random uniform scalars and puts them in the places of scalars in the placeholder.
- It independently samples 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 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 proved 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 proved 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 where a linear relationship between different input commitments is known to an adversary, we extend **L2S** protocol with 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 of **L2S** 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 nonzero , 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.
- **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.

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 steps of 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 public **RL2S** protocol transcript, and as Z in the transcript becomes $Z = fZ_0$, it's necessary to determine the distributions of them:

Table 10: **RL2S.InteractionProcedure**.

Prover and Verifier common parameters:

- $n, N = 2^n$, where $n > 1$
- a set of elements $\left[X_j\right]_{j=0}^{N-1} = \textbf{DecoySetGen}(n)$

Prover:		Verifier:
(k_0, k_1, s), where $k_0 \neq 0$		Z_0
s – secret index, $s \in [0, N/2 - 1]$		
$Z_0 = \text{Com2}\left(k_0, k_1, s, \left[X_j\right]_{j=0}^{N-1}\right)$		
$f \leftarrow$ random, nonzero		
$Z = fZ_0$	$Z \rightarrow$	
$(k_0, k_1, s) = (fk_0, fk_1, s)$		
$q \leftarrow$ random, nonzero		
$T_0 = qZ_0$	$T_0 \rightarrow$	
	$\leftarrow c_0 \leftarrow$ random, nonzero	
$t_0 = q - fc_0$	$t_0 \rightarrow$	

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

Verifier has two tuples:

(Z_0, c_0, T_0, t_0),

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

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

- 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 $T_0 = (t_0Z_0 + c_0Z)$ 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 input commitment 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 $(t_0Z_0 + c_0Z)$ and $(tW + cR)$ at 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.

Table 12: **MRL2S.MapInteractionProcedure**.

Prover and Verifier common parameters: • L • $n, N = 2^n$, where $n > 1$	
Prover: $\left[\left(k_0^p, k_1^p, s^p\right) \| k_0^p \neq 0\right]_{p=1}^{L}$	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: $\left[\left(\left(Z_0^p, c_0, T_0^p, t_0^p\right), \left([(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n, Z^p, \left[\left(r_i^p, H_i^p\right)\right]_{i=1}^{n}, c, T^p, t^p\right)\right)\right]_{p=1}^{L}$

- **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

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

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:

MapInteractionProcedure → **InteractionProcedure**,
JoinVerif → **Verif**

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

Input: $L, n, \left[X_j\right]_{j=0}^{N-1}$, where $N = 2^n, n > 1$, $\left(\left(Z_0^p, c_0, T_0^p, t_0^p\right), \left([(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n, Z^p, \left[(r_i^p, H_i^p)\right]_{i=1}^{n}, c, T^p, t^p\right)\right)_{p=1}^{L}$
$R = \text{Rsum}\left(n, N, \left[X_j\right]_{j=0}^{N-1}, [(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n\right)$ For each $p \in [1, L]$: run **RL2S.Verif** using n, $\left[X_j\right]_{j=0}^{N-1}$ and $\left(Z_0^p, c_0, T_0^p, t_0^p\right), \left([(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n, Z^p, \left[(r_i^p, H_i^p)\right]_{i=1}^{n}, c, T^p, t^p\right)$ as arguments. Within each **RL2S.Verif** call, inside 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.

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 L (polynomial number) nested **RL2S** protocol emulators, which synchronously rewind to symmetrical points in the L transcript trees. In sum, here is the polynomial time relation that the **MRL2S** protocol emulator finds witness for

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

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 one addition, as follows.

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 the equal challenges 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 as revealing no information (this was discussed in 7.1.2, 6.3.4), are distributed independently and uniformly at random. Independence here is meant as the ultimate 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 $\left[X_j\right]_{j=0}^{N-1}$ are 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 is 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 [12, 24, 3]. We will create a non-interactive zero-knowledge PoM scheme on the base of **MRL2S**. Also, in this non-interactive zero-knowledge PoM we will not allow the odd elements of the **MRL2S** decoy set to participate in the commitment Z_0, and thus Z_0 will become an element picked at some even position in the decoy set and multiplied by some secret, not necessarily random, scalar.

In the role of random oracle, to generate challenges, we will be using 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 will be using '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 will be using 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. Namely, the arguments are strings, scalars in $\mathbb{F}$, and points in $\mathbb{G}$. The returned value of $\mathbf{H_{scalar}}(\dots)$ is a scalar in $\mathbb{F}$. 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 [10]. For the $\mathbf{H_{point}}$ it can be, for instance, one of the functions described in [19, 11, 13].

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. As our PoM will be selecting only from even elements of decoy set, it will be actually selecting from $N/2$ elements.
- $n = \log_2(N)$.
- For signatures, L is a threshold such that $1 \leqslant L \leqslant N/2$. For membership proof, L is any number, $1 \leqslant L$.
- 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 $[(P_j, Q_j)]_{j=0}^{N/2-1}$, 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 images $[G_i]_{i=0}^{D-1} = [\mathbf{H_{point}}(0,i)]_{i=0}^{D-1}$, and use it for the odd elements of the decoy sets. All the other $\mathbf{H_{point}}$ images are orthogonal to $[G_i]_{i=0}^{D-1}$ as well, the following implementation is implied for this

$$\mathbf{H_{point}}(S) = \mathbf{H_{point}}(1, [G_i]_{i=0}^{D-1}, S).$$

8.2 MRL2SPOM NIZK POM SCHEME

The abbreviation **MRL2SPoM** stands for the **MRL2S**-based proof of membership scheme, it proves 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 to 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} = [H_point("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 != 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} = GetPredefinedGenerators()
    [P_j]_{j=0}^{N/2-1} = SetGen()
    [Q_j]_{j=0}^{N/2-1} = [G_j + F]_{j=0}^{N/2-1}
    [X_j]_{j=0}^{N-1} = Flatten([(P_j, Q_j)]_{j=0}^{N/2-1})
```

Return $[X_j]_{j=0}^{N-1}$

- **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 $\mathbb{0}$ --proof, vector of σ's on success, --$\mathbb{0}$ on failure
Procedure:

- Let $[P_j]_{j=0}^{N/2-1} = \mathbf{SetGen}()$
- Let $[Z_0^p]_{p=1}^{L} = \mathbf{MembersGen}()$
- Let $F = \mathbf{H_{point}}([P_j]_{j=0}^{N/2-1}, [Z_0^p]_{p=1}^{L})$. --Thus, there holds --$F != \mathrm{lin}([Z_0^p]_{p=1}^{L} \cup [P_j]_{j=0}^{N/2-1} \cup [G_i]_{i=0}^{D-1})$.
- Let $[X_j]_{j=0}^{N-1} = \mathbf{GetDecoySet}(F)$
- For $p = 1 \ldots L$: --Ensure the private keys corresponds to the member set elements.
  If $Z_0^p \neq v^p X_{2s^p}$ then Return $\mathbb{0}$.
- 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 $e = \mathbf{H_{scalar}}(e, [X_j]_{j=0}^{N-1}, [(Z_0^p, T_0^p, Z^p)]_{p=1}^{L})$
- 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 $e = \mathbf{H_{scalar}}(e, [t_0^p]_{p=1}^{L}, [H_1^p]_{p=1}^{L})$
- Let $(c_{11}, c_{13}) = (e, \mathbf{H_{scalar}}(e))$
- 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 $e = \mathbf{H_{scalar}}(e, [r_1^p]_{p=1}^{L}, [H_2^p]_{p=1}^{L})$
- Let $(c_{21}, c_{23}) = (e, \mathbf{H_{scalar}}(e))$
- 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. Its 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 σ's
Output: 0 or 1 --verification is failed or completed ok
Procedure:

- Let $[P_j]_{j=0}^{N/2-1} = \mathbf{SetGen}()$
- Let $[Z_0^p]_{p=1}^{L} = \mathbf{MembersGen}()$
- Let $F = \mathbf{H_{point}}([P_j]_{j=0}^{N/2-1}, [Z_0^p]_{p=1}^{L})$. 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 $[X_j]_{j=0}^{N-1} = \mathbf{GetDecoySet}(F)$
- Extract the values of $[(T_0^p, Z^p)]_{p=1}^{L}$ from $[\sigma^p]_{p=1}^{L}$
- Calculate $e = \mathbf{H_{scalar}}(e, [X_j]_{j=0}^{N-1}, [(Z_0^p, T_0^p, Z^p)]_{p=1}^{L})$
- 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 $e = \mathbf{H_{scalar}}(e, [t_0^p]_{p=1}^{L}, [H_1^p]_{p=1}^{L})$
- Let $(c_{11}, c_{13}) = (e, \mathbf{H_{scalar}}(e))$
- Extract the values of $[r_1^p]_{p=1}^{L}$ and $[H_2^p]_{p=1}^{L}$ from $[\sigma^p]_{p=1}^{L}$
- Calculate $e = \mathbf{H_{scalar}}(e, [r_1^p]_{p=1}^{L}, [H_2^p]_{p=1}^{L})$
- Let $(c_{21}, c_{23}) = (e, \mathbf{H_{scalar}}(e))$
- 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 $p = 1 \ldots L$:
  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 $p = 1 \ldots L$:
  Let $S = Z^p$
  For $i = 1 \ldots n$:
  $S = S + r_i^p H_i^p$
  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 to be 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 seed e usage is the same as in [17].
- 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 meet the **MRL2S** procedures translated to the non-interactive setting with the Fiat-Shamir heuristic. Orthogonality of the decoy set returned from **MRL2SPoM.GetDecoySet** follows from the OrtHalfShift lemma in 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 a successful **MRL2SPoM** transcript, let's show that Verifier is convinced that $k_1 = 0$ for all input Z_0's. For each $p \in [1, L]$, from the **MRL2SPoM.GetDecoySet** implementation (Listing 2), Verifier has $Q_{sp} = G_{sp} + F$, and hence the equation (72) rewrites as

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

According to the **MRL2SPoM.Prove** specification (Listing 3) and by the definition of $\mathbf{H_{point}}$, F is a hash point of the points G_{sp}, P_{sp}, and Z_0^p. Therefore, if $k_1 \neq 0$, then $F = \mathrm{lin}(Z_0^p, P_{sp}, G_{sp})$, 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 SHVZK AND WITNESS-EXTENDED EMULATION

As proved in 7.2.2, each **MRL2S** transcript contains the independently and 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** built in 7.2.2. Thus, the **MRL2SPoM** scheme is sHVZK.

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

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

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 not accounted as 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 threshold ring signature scheme on the base of the **MRL2SPoM** membership proof. For any threshold L, we assume there are L signers each having a key pair (b, B) of secret and public keys such that $B = bG$. The signers sign common message m with their private keys; in doing so, they include their public keys into the ring, and dilute the ring with other elements selected ad hoc or, in the worst case, adversarilly.

In brief, the realization idea of **MRL2SLnkSig** is that we encode each element B in the ring as a point $(z\mathbf{H_{point}}(B) + B)$ of the **MRL2SPoM** set. We let each signer publish an additional point I, called key image. After that, an **MRL2SPoM** membership proof about that each point $(zI + G)$ belongs to the set of $(z\mathbf{H_{point}}(B) + B)$'s is constructed. The coefficient z is a randomness picked when all I's are published. This way the signers prove knowledge of factors w's such that for each I there holds $(z\mathbf{H_{point}}(B) + B) = w(zI + G)$ for some B in the ring. It

turns out that $w = b$ always holds in this case. Thus, in sum, the **MRL2SLnkSig** signature is the **MRL2SPoM** membership proof for the points of the form $(zI + G)$, where I is the key image and z is the randomness.

To show the **MRL2SLnkSig** signature is practical, we will prove it meets the common linkable ring signature security requirements. In particular, we will prove no PPT adversary is able to forge it, i.e. there is no way to produce an acceptable signature without knowing the corresponding private keys, even by observing any number of signatures made by others.

9.1 PRELIMINARY LEMMA

Random weighting is a common cryptographic technique for combining two or more proofs into one. It is used in various forms for different scenarios, e.g., as in [23, 29]. Let's formulate the following lemma for a variant of random weighting that we will use.

Lemma 13 (RandomWeighting-WEE)**:**
For any four nonzero elements X, Y, C, D such that $C \,!\sim D$ 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.

	⟵ Verifier sends a nonzero 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 $C \,!\sim D$. The protocol soundness follows from the witness-extended emulation that we will prove now. Note, we do not mention sHVZK for this protocol, since Prover does not send any data in it, and, hence, this protocol inherits the sHVZK property from the evidence protocol played in the last step, if the latter has it.

Let's build a WEE emulator for the protocol in 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 nonzero $(z' - z)$, 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 built the 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 threshold ring signature scheme is the following four procedures

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

where:

- **MRL2SLnkSig.RingGen** returns a vector $[B_j]_{j=0}^{N/2-1}$ of arbitrary points. These points are only required to be nonzero and different from each other. The actual signer public keys are to be placed among them.
- **MRL2SLnkSig.Sign** takes a scalar message m, a vector of actual signer's private keys $[(b^p, s^p)]_{p=1}^{L}$ such that $[(b^p, s^p) \mid b^pG = B_{s^p},\ s^p \in [0, N/2-1],\ \forall i, j : s^i \neq s^j]_{p=1}^{L}$, and returns

  $$\text{the signature } \mathfrak{S} = [(I^p, \sigma^p)]_{p=1}^{L} \text{ on success or 0 on error.}$$

  The points $[I^p]_{p=1}^{L}$ contained in the signature are called key images. Implementation is shown in Listing 5. Note, we use lambda notation for procedure substitutions in the listings.

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

```
Input:  m                       --message
        [(b^p, s^p)]_{p=1}^L    --private keys
Output: [(I^p, σ^p)]_{p=1}^L or 0  --signature on success,
                                --0 on failure
Procedure:
    [B_j]_{j=0}^{N/2-1} = RingGen()
    [I^p]_{p=1}^L = [H_point(b^p G)/b^p]_{p=1}^L
    For j = 1...L:
        If I^j ∈ ([I^p]_{p=1}^L \ {I^j}) then Return 0
    z = H_scalar(m, [B_j]_{j=0}^{N/2-1}, [I^p]_{p=1}^L)
    MRL2SPoM.SetGen = λ.([B_j + zH_point(B_j)]_{j=0}^{N/2-1})
    MRL2SPoM.MembersGen = λ.([G + zI^p]_{p=1}^L)
    e = H_scalar(z)
    proof = MRL2SPoM.Prove(e, [(1/b^p, s^p)]_{p=1}^L)
    If proof == 0 then Return 0
    [σ^p]_{p=1}^L = proof
    Return [(I^p, σ^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, σ^p)]_{p=1}^L    --signature
Output: 1 or 0                  --success or failure
Procedure:
    [B_j]_{j=0}^{N/2-1} = RingGen()
    z = H_scalar(m, [B_j]_{j=0}^{N/2-1}, [I^p]_{p=1}^L)
    MRL2SPoM.SetGen = λ.([B_j + zH_point(B_j)]_{j=0}^{N/2-1})
    MRL2SPoM.MembersGen = λ.([G + zI^p]_{p=1}^L)
    e = H_scalar(z)
```

```
    If MRL2SPoM.Verif(e, [σ^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...L:
        If I_0^j ∈ [I_1^p]_{p=1}^L then Return 1
    Return 0
```

The main **MRL2SLnkSig** usage scenario is:

- 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 the signature $\mathfrak{S} = [(I^p, \sigma^p)]_{p=1}^L$.
- Verifier receives the message m with the signature $\mathfrak{S}$, and calls **MRL2SLnkSig.Verif** for them. If 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. Verifier is also convinced that the vector $[I^p]_{p=1}^L$ contains key images of the signing private keys in this case. Note, if Prover signs with some equal private keys, then the vector of key images $[I^p]_{p=1}^L$ contains duplicates, which is seen to Verifier.
- 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. Verifier calls **MRL2SLnkSig.Link** for these key image vectors and, iff it returns 1, then Verifier gets convinced that at least one private key was used to sign both signatures.

9.2.2 SCHEME COMPLETENESS, SOUNDNESS, AND WITNESS-EXTENDED EMULATION

The **MRL2SLnkSig** scheme is a combination of the **MRL2SPoM** scheme defined in 8.2 with the random weighting protocol in Table 18. The **MRL2SPoM** scheme plays a role of the evidence $(zX + Y) \sim (zC + D)$ for the protocol in Table 18. According to the **MRL2SPoM** scheme properties which were established in Sections 8.2.1, 8.2.2 and by the RandomWeighting-WEE lemma in Section 9.1, the following can be stated.

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 for which the **MRL2SLnkSig** emulator finds witness is the intersection of the relations (70) and (74)

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

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 the key image denoted as I, and the second item is the **MRL2SPoM** proof (71) denoted as σ. Structure and view of the $[\sigma^p]_{p=1}^L$ part of the signature was described in 8.2.2, where it was shown that σ's reveals no information and can be treated as absolutely random, excluding few completely dependent items which do not change the case. The question is how much information can be obtained from $[I^p]_{p=1}^L$.

Apparently, I reveals 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 acceptable **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 on the system, then the space looks completely random, provided that the public keys B's are distributed independently and uniformly at random. This is due to the DDH assumption, by which the triples $(B,\ \mathbf{H_{point}}(B),\ I)$ look completely random, indistinguishable from the triplets (aG, bG, cG) with the independent and uniformly random a, b, c. To be precise, we refer to the DDDH assumption which follows [2] from DDH.

For uneven distributions of the public keys B's, which we also consider, the situation is a bit more complicated, as we cannot use DDH for the triples $(B,\ \mathbf{H_{point}}(B),\ I)$. Nevertheless, in 9.3.2 we will prove that for any distribution of public keys B, including uneven, our linking tag $I = x^{-1}\mathbf{H_{point}}(xG)$ is indistinguishable from the linking tag $x\mathbf{H_{point}}(xG)$ used, e.g., in the well-known LSAG signature [22], and both are indistinguishable from the independent and uniformly random distribution, provided that the corresponding private keys b's cannot be recovered.

The similarity of **MRL2SLnkSig** to LSAG is not limited by the similarity of their key images, both have all their view items either indistinguishable from white noise or completely dependent. So we can transfer many security proofs from LSAG to our signature. The same also applies to the CLSAG signature [15], which uses the LSAG's linking tag formula.

9.2.4 SIGNATURE SIMULATION

If the **MRL2SLnkSig** signature were zero knowledge, then by definition of sHVZK there would be an interactive simulator that for any distributed independently and uniformly ring $[P_j]_{j=0}^{N/2-1}$, input I, and any set of random challenges known in advance, produces an acceptable signature indistinguishable from the 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 (with known private keys) public keys $\{P_0, P_1\}$ and with the simulated (indepandently and uniformly sampled) key image 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 the honest one, as there is only negligible probability that the random I is equal to one of the known I_0, I_1.

Interestingly, according to the signature view discussed in 9.2.3, if the ring contains a dishonest public key such that its private key can never be efficiently calculated, then the simulation seems possible with the independent and uniformly sampled input I. Suppose, key P_1 is dishonest in the ring $\{P_0, P_1\}$. Then the simulator yields (I, σ), where σ is a **MRL2SPoM** simulated transcript indistinguishable from the honest one. The random point I, having distribution of an honest key image, is indistinguishable from it since 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 the honest one in this case.

9.2.5 COMPLEXITIES

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	$L(n+4)$	$L(n+2)$

MRL2SLnkSig verification complexity is shown in Table 20. In addition to the optimization used for the **MRL2SPoM** scheme, 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 a single multi-exponential expression is collected, and perform all exponents with it. In the meantime, before the expression is finally collected, the necessary hashes of P_j's are calculated as hashes of the tuples $(B_j, \mathbf{H_{point}}(B_j), z)$.

Recalling N commonly denotes 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 size of a point in $\mathbb{G}$ is equal to size of a scalar in $\mathbb{F}$.

Table 20: **MRL2SLnkSig** verification complexity.

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

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

Works presenting new signature schemes, such as [1, 4, 5, 15, 17, 20, 22, 23, 26, 29], provide proofs that they cannot be tampered with certain types of attacks. These proofs have many things in common, e.g. in attack modeling, however they may differ in details. The works [17, 22, 23] contain a set of security requirements typically imposed on linkable ring signatures, so below we mainly retell the definitions from these works and refer to the approaches presented there to prove attack resistance.

We first consider a generic concept of linkable ring signature, which we call GLRS, along with its security model. GLRS is similar to the LRS concept from [23], the only difference is that KeyGen in GLRS does not necessarily return independently and uniformly distributed keys, which is similar to LSAG [22]. We prove that the signature **MRL2SLnkSig** meets the GLRS security requirements, including that **MRL2SLnkSig** is unforgeable w.r.t. insider corruption and also is 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 conclude that **MRL2SLnkSig** remains secure in this case as well.

Furthermore, wondering about the applicability of our signature in the real world, we note that one of the classical definitions of unforgeability do not account for the scenario where a forger falsifies the signature by populating the ring with malformed public keys. In Section **??** we refer to the CLSAG paper [15], where this issue is discussed, and find that such forging is impossible for **MRL2SLnkSig**, just as it is impossible for CLSAG and LSAG.

9.3.1 DEFINITIONS FOR $L = 1$, GLRS AND ITS SECURITY MODEL

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 observing only the X's generated by KeyGen and having no additional information, for any X, it is hard to find the corresponding 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 the signature $\boldsymbol{\sigma}$ on the 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 the 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 $\Pr[\text{Verify}(\boldsymbol{\sigma}, m, R) = 1] = 1$, we say that the GLRS is perfectly correct. If $\Pr[\text{Verify}(\boldsymbol{\sigma}, m, R) = 1] \approx 1$, 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 $\text{KeyGen}() \rightarrow (x_i, X_i)$ 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 $\text{CorruptOracle}(i)$ that returns x_i if it corresponds to a query to GenOracle.
- The adversary $\mathcal{A}$ is granted access to a signing oracle $\text{SignOracle}(X, m, R)$ that runs $\text{Sign}(x, m, R) \rightarrow \boldsymbol{\sigma}$ 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 $\Pr[\text{Verify}(\boldsymbol{\sigma}, m, R) = 1] \approx 0$, 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 $\text{KeyGen}() \rightarrow (x_i, X_i)$ and returns X_i to $\mathcal{A}$.
- The adversary $\mathcal{A}$ is granted access to a signing oracle $\text{SignOracle}(X, m, R)$ that runs $\text{Sign}(x, m, R) \rightarrow \boldsymbol{\sigma}$ 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 $\Pr[\text{Verify}(\boldsymbol{\sigma}, m, R) = 1] \approx 0$, we say that the GLRS is unforgeable against adaptive chosen message and adaptive chosen public key attackers (EU_CMA/CPA).

Note, this definition of EU_CMA/CPA is equivalent to the definition of Existential unforgeability against adaptive chosen plaintext, adaptive chosen-public-key attackers in [22].

As can be seen from the above 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 does not allow key corruption, because a single call to CorruptOracle may corrupt all keys at once when they have such an uneven distribution that they all depend on 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 $\text{Sign}(x_{i_b}, m, R) \rightarrow \boldsymbol{\sigma}$, 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 $\text{Sign}(x_{i_b}, m, R) \rightarrow \boldsymbol{\sigma}$, 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.

Note, this definition of anonymity w.r.t. CPA looks like a subset of the definition of Signer Ambiguity in [22], although they are possibly equal, we do not investigate this now.

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 $R_i \subset V$.
  - $\text{Verify}(\boldsymbol{\sigma}_i, m_i, R_i) = 1$ for all i.
  - $\text{Verify}(\boldsymbol{\sigma}, m, R) = 1$.
  - For all $i \neq j$, we have $\text{Link}(\boldsymbol{\sigma}_i, \boldsymbol{\sigma}_j) = \text{Link}(\boldsymbol{\sigma}_i, \boldsymbol{\sigma}) = 0$.
- 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, which redirects to KeyGen and returns the public key generated by KeyGen. 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 the corruption oracle CorruptOracle.
- 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, but was not corrupted by CorruptOracle. It selects a message m and ring R such that $X \in R$. It queries $\text{SignOracle}(X, m, R) \rightarrow \boldsymbol{\sigma}$.
- 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 $\text{Verify}(\boldsymbol{\sigma}', m', R') = 0$ or if $\boldsymbol{\sigma}'$ was produced using a query to SignOracle, the challenger aborts.

If $\Pr[\text{Link}(\boldsymbol{\sigma}, \boldsymbol{\sigma}') = 1] \approx 0$, 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 $\text{SignOracle}(X, m, R) \rightarrow \boldsymbol{\sigma}$.
- 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 $\text{Verify}(\boldsymbol{\sigma}', m', R') = 0$ or if $\boldsymbol{\sigma}'$ was produced using a query to SignOracle, the challenger aborts.

If $\Pr[\text{Link}(\boldsymbol{\sigma}, \boldsymbol{\sigma}') = 1] \approx 0$, 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 LINKING TAG INDISTINGUISHABILITY FROM INDEPENDENT UNIFORM RANDOMNESS

The following lemma proves one useful property of DDH-quadruples. As a corollary, it follows that, for an arbitrary distribution of private keys x's such that nevertheless they cannot be recovered from their public keys, a linking tag in the form $x\mathbf{H_{point}}(xG)$ is statistically indistinguishable from a linking tag in the form $x^{-1}\mathbf{H_{point}}(xG)$, and both are indistinguishable from the independently and uniformly distributed randomness.

Thus, it follows from the lemma that the linking tag $x\mathbf{H_{point}}(xG)$ which is used, e.g., in the LSAG and CLSAG schemes [22, 15] is indistinguishable from the linking tag $x^{-1}\mathbf{H_{point}}(xG)$ used in the **MRL2SLnkSig** signature, provided that x is unknown.

Lemma 14 (UnevenDDH)**:**
Under the DDH assumption, for a generator G, for the independently and uniformly distributed random scalars a, c, for any distribution of nonzero scalar x such that no x can be efficiently computed from a stream of xG's, the following three assertions hold

A) quadruple (G, xG, aG, xaG) is statistically indistinguishable from quadruple (G, xG, aG, cG),

B) quadruple $(G, xG, aG, x^{-1}aG)$ is statistically indistinguishable from quadruple (G, xG, aG, cG),

C) quadruple $(G, xG, aG, x^{-1}aG)$ is statistically indistinguishable from quadruple (G, xG, aG, xaG).

Proof. Suppose, the assertion A) doesn't hold. This means that there exist an adversary $\mathcal{A}$ and a distribution $\mathcal{D}$ such that $\mathcal{A}$ has statistical advantage $\epsilon(\boldsymbol{\lambda}) > 0$ in distinguishing (G, xG, aG, xaG) from (G, xG, aG, cG), as long as x has distribution $\mathcal{D}$. Without loss of generality, let's assume that $\mathcal{A}$ returns 0 when it detects a quadruple of the first type, and 1 when of the second type.

Let's consider the following game. Challenger C generates on its own an internal stream of triplets (x, P_0, P_1) such that $P_0 = xG$, $P_1 = x^{-1}G$, and x is distributed by $\mathcal{D}$. For each triplet, C constructs two LSAG signatures [22], $S_0 = (I_0, \dots)$ and $S_1 = (I_1, \dots)$, over the ring $R = \{P_0, P_1\}$ such that the key P_0 is the actual signer for S_0, the key P_1 is the actual signer for S_1, with the key images $I_0 = x\mathbf{H_{point}}(P_0)$ and $I_1 = x^{-1}\mathbf{H_{point}}(P_1)$, respectively. Then, C flips a coin and writes one of the S_0, S_1 signatures along with the ring R into its output stream.

Next, in this game, master $\mathcal{M}$ listens to the C's output stream and reads the pairs (S, R) from it. For each pair, without knowing which of the two signatures is S, the master $\mathcal{M}$ takes P_0 from R, takes I from S, and composes the quadruple $(G, P_0, \mathbf{H_{point}}(P_0), I)$. Then, $\mathcal{M}$ asks $\mathcal{A}$ to distinguish the quadruple $(G, P_0, \mathbf{H_{point}}(P_0), I)$.

Let's look at what this quadruple is from $\mathcal{A}$'s point of view. In the case if $S = S_0$, the quadruple has the form (G, xG, aG, xaG), where a is the independent uniform randomness produced by $\mathbf{H_{point}}(P_0)$. Otherwise, if $S = S_1$, the quadruple has the form (G, xG, aG, cG), where $c = x^{-1}b$, and a, b are the independent uniform randomnesses produced by $\mathbf{H_{point}}(P_0), \mathbf{H_{point}}(P_1)$, respectively. Thus, a, c are independently and uniformly distributed in this case. So, $\mathcal{A}$ has on its input one of the quadruples it is designed for, and, using its advantage, $\mathcal{A}$ distinguishes them.

Going back to the master $\mathcal{M}$, after getting the answer 0 from $\mathcal{A}$ it states that S is S_0, whereas getting the answer 1 it states that S is S_1. Thus, $\mathcal{M}$ correctly detects the actual signer with a probability greater than a random coin flip. However, this is not possible due to the Signer Ambiguity property of the LSAG scheme proved in [22]. Therefore, the supposition is incorrect. We have proved the assertion A) holds.

The case of B) is proved in the same way. The challenger C remains the same. Master $\mathcal{M}$ now composes the quadruple $(G, P_0, \mathbf{H_{point}}(P_1), I)$ and, getting the answer 0 from $\mathcal{A}$ states that S is S_1, otherwise S is S_0. From $\mathcal{A}$'s point of view, in this case, if $S = S_0$, the quadruple has the form (G, xG, aG, cG), where $c = xb$, and a, b are the independent uniform randomnesses produced by $\mathbf{H_{point}}(P_1), \mathbf{H_{point}}(P_0)$, respectively. Thus, a, c are independently and uniformly distributed. Otherwise, if $S = S_1$, the quadruple has the form $(G, xG, aG, x^{-1}aG)$, where a is the randomness produced by $\mathbf{H_{point}}(P_1)$. So, $\mathcal{A}$ has on its input one of the quadruples it is designed for, and, using its advantage, $\mathcal{A}$ distinguishes them which, again, falls into a contradiction to LSAG's Signer Ambiguity.

The assertions A) and B) together imply that the assertion C) holds, since the independent uniform randomness is indistinguishable from itself. Thus, we have proved that all the three assertions A), B), C) hold. □

Note, when getting the contradiction by appealing to LSAG's Signer Anonymity in the above proof, we implicitly use the fact that no x can be efficiently computed from a stream of xG's which holds by the lemma's premise. Otherwise, if x can be known from xG's, then $\mathcal{M}$ is able to trivially distinguish the quadruples with x.

Corollary 1 (of UnevenDDH lemma)**:**
*The **MRL2SLnkSig** signature can be simulated using an independently and uniformly distributed random I in place of the key image $x^{-1}\boldsymbol{H_{point}}(P)$. By the simulation is meant producing an indistinguishable from the honest one signature without knowing private key, with the challenges known in advance.*

Proof. When such an I is put in place of the key image, it cannot be distinguished from the real key image by the UnevenDDH lemma's statement B). At the same time, to make indistinguishable the rest of the signature, which

comprises a **MRL2SPoM** proof, it suffices to use the **MRL2SPoM** simulator from 8.2.2. Thus, the simulated signature is indistinguishable from the real one. □

Corollary 2 (of UnevenDDH lemma)**:**
*The **MRL2SLnkSig** signature can be simulated using $I = x\boldsymbol{H_{point}}(P)$ instead of the key image $I = x^{-1}\boldsymbol{H_{point}}(P)$. By the simulation is meant producing an indistinguishable from the honest one signature without knowing private key, with the challenges known in advance.*

Proof. By the UnevenDDH lemma's statement A) the simulated key image $I = x\mathbf{H}_{\mathbf{point}}(P)$ is indistinguishable from the independently and uniformly distributed randomness. Hence, by the UnevenDDH lemma Corollary 1, the **MRL2SLnkSig** signature can be simulated this way. □

9.3.3 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 $L = 1$.

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 proved in 9.2.2, implies correctness. It is trivially seen from the design of the scheme core protocol (Table 8) that **MRL2SLnkSig** is perfectly correct for $L = 1$.

The **MRL2SLnkSig** scheme is unforgeable w.r.t. insider corruption. Since the proof of this is exactly the same as the proof of unforgeability w.r.t. insider corruption for the ring signature in [17], we do not repeat it here, providing only the reference. To transfer the proof from [17] we model the $\mathbf{H}_{\mathbf{scalar}}$ function as a random oracle. Instead of the zero-knowledge property of the (not-linkable) ring signature in [17], we use the simulated signature indistinguishability asserted by the Corollary 1 of the UnevenDDH lemma.

The **MRL2SLnkSig** scheme is EU_CMA/CPA. Since the proof of this is exactly the same as the proof of EU_CMA/CPA for the LSAG signature in [22], we do not repeat it here, providing only the reference. The proof in [22] simulates the $\mathbf{H}_{\mathbf{point}}$ function by randomly picking r and returning rG and, hence, it requires the linking tag to have the form $I = x\mathbf{H}_{\mathbf{point}}(P)$ such that it can be later replaced by $I = rP$. By the Corollary 2 of the UnevenDDH lemma, if the **MRL2SLnkSig** signature linking tag $I = x^{-1}\mathbf{H}_{\mathbf{point}}(P)$ is replaced by $I = rP$ during the simulation, then the simulated signature remains indistinguishable from the honest one. Thus, the proof of EU_CMA/CPA from [22] applies here. □

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 I together with the public key ring R. For any uncorrupted signing public key $X_i \in R$, by the UnevenDDH lemma's statement B) the corresponding key image $I = x_i^{-1}\mathbf{H}_{\mathbf{point}}(X_i)$ is distributed independently and uniformly at random. 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 they bring no additional information. Therefore, $\mathcal{A}$ wins only with the probability of a random coin flip, and hence, 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 for the corresponding signature in [23] also applies to **MRL2SLnkSig**. For both of the signature provided in [23] and ours, the underlying protocols (underlying proving systems) are sHVZK. The difference in the linking tags doesn't change the proof, so we refer to [23] as the source of the proof and don't repeat it here.

The same is for non-frameability. The non-frameability proof in [23] applies to **MRL2SLnkSig** as well, so we don't repeat it here. Moreover, since the proof relies only on the scheme witness extraction property, namely, on

special soundness in [23], 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. □

9.3.4 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 comprising four procedures:

- KeyGen() $\rightarrow (x, X)$: Generates key pairs, the same as for GLRS.
- L is a threshold.
- Sign$(\vec{x}, m, R) \rightarrow \mathfrak{S}$: Generates 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 the signature $\mathfrak{S}$ on a message m with respect to the ring R. Returns 0 if the signature is rejected, and 1 if accepted. The signature $\mathfrak{S}$ is accepted iff it was created with a call to Sign$(\vec{x}, m, R)$.
- Link$(\mathfrak{S}, \mathfrak{S}') \rightarrow \{0, 1\}$: Determines if the 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.

The **MRL2SLnkSig** signature is considered GLTRS by appropriately renaming its 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 [15, 27].

Theorem 4 (Informal)**:**
*The **MRL2SLnkSig** signature scheme, for $L \geq 1$, considered as GLTRS, has the following properties: perfect correctness, unforgeability w.r.t. insider corruption and EU_CMA/CPA, anonymity and anonymity w.r.t. CPA, linkability, non-frameability, and non-frameability w.r.t. CPA.*

Proof. For the proof of this theorem, we present only the idea behind it. It is rather easy, so given the already proved case $L = 1$ we will not go much into details.

MRL2SLnkSig is perfectly correct for $L \geq 1$, this is trivially seen from the implementation (Table 8). Regarding the two types of anonymity, they both hold for the following reason. For any ring R, for any acceptable instance $\mathfrak{S} = [(I^p, \sigma^p)]_{p=1}^L$ of the **MRL2SLnkSig** signature over R, its part $[\sigma^p)]_{p=1}^L$ is an instance of the **MRL2SPoM** proof which is proved sHVZK in Section 8.2.2. Thus, the part $[\sigma^p)]_{p=1}^L$ can be excluded from the consideration as not revealing even a bit of information. The remaining data $([I^p)]_{p=1}^L, R)$, since I^p's related to uncorrupted signing keys are by the UnevenDDH lemma's statement B) indistingushable from white noise, reveals no information other than what the corrupted keys might reveal themselves.

MRL2SLnkSig is unforgeable w.r.t. insider corruption and also is EU_CMA/CPA for $L \geq 1$, since everything the PPT algorithms borrowed from [22, 17] do and get in the proof of Theorem 1 in 9.3.3 is scalable to $L-$dimensions. Namely, as $\mathfrak{S} = [(I^p, \sigma^p)]_{p=1}^L$ is a direct sum of a polynomial number L of the independent parts $(I^p, \sigma^p) \in \mathfrak{S}$, for $L \geq 1$, L instances of the case $L = 1$ unforgeability proofs can be conducted in parallel.

Linkability, non-frameability, and non-frameability w.r.t. CPA also alive for **MRL2SLnkSig** in $L-$dimensions, because if one of them were 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 in 9.3.3.

□

9.4 MRL2SLNKSIG SIGNATURE AND SOME RECENTLY PROPOSED LOG-SIZE SCHEMES

Although we created **MRL2SLnkSig** mainly to show the applicability of the Lin2-Xor lemma, we can try to compare its performance with some recent log-size signatures. We refer to the work [23], where proof sizes and verification complexities for two of the recently proposed top-performative schemes are shown in Table 1 [23].

A direct performance comparison of **MRL2SLnkSig** to the schemes analyzed in [23] seems hard due to the following two reasons:

- The linkable signature schemes analyzed in [23] 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 [23] operates with the linking tags of the form $x^{-1}U$, where U is a predefined independent generator.

Anyway, we can look at the asymptotes. Assuming an $\mathbf{H_{point}}$ call is about ten times faster than an exponentiation, and thus $\mathbf{H_{point}}$ takes about the same time as an exponentiation calculated using the multi-exponent for a roughly 2^{10} element ring, we can see that, for instance, for big N's and $L = 1$, our signature verification time asymptote is $\boldsymbol{mexp}(3N) + N\mathbf{H_{pt}}$, which is not far from the RingCT 3.0's asymptote $\boldsymbol{mexp}(4N)$. Triptych asymptote $\boldsymbol{mexp}(2N)$ is about 2 times faster than both ours and RingCT 3.0.

As for the thresholds $L > 1$, RingCT 3.0 provides the best asymptotic time $O(\log_2 N + L)$, whereas our signature is only $O(L \log_2 N)$.

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.

It is worth mentioning that we use linking tag in the form $x^{-1}\mathbf{H_{point}}(xG)$, for which we proved indistinguishability from the most time-tested and secure linking tag $x\mathbf{H_{point}}(xG)$, used e.g. in [22, 28]. This form of linking tag is absolutely insensitive to a distribution of public keys and allows our signature to be anonymous even if the keys are deliberately malformed.

10 POSSIBLE EXTENSIONS

The **MRL2SPoM** membership proof and **MRL2SLnkSig** signature are schemes that are achievable using the Lin2-Xor lemma. The possible extentions could include an optimization of **MRL2SLnkSig** as well as creating other arguments of knowledge using the methods of 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 that the method of the Lin2-Xor lemma itself can be further developed.

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] Feng Bao, Robert H. Deng, and HuaFei Zhu. "Variations of Diffie-Hellman Problem". In: *Information and Communications Security*. Ed. by Sihan Qing, Dieter Gollmann, and Jianying Zhou. Springer Berlin Heidelberg, 2003, pp. 301–312.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] Ivan Damgård. *On Σ-protocols*. CPT 2010, v.2. https://cs.au.dk/~ivan/Sigma.pdf. 2010.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] S Goldwasser, S Micali, and C Rackoff. “The Knowledge Complexity of Interactive Proof-Systems”. In: *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*. New York, NY, USA: Association for Computing Machinery, 1985, pp. 291–304.

[15] 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.

[16] 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.

[17] 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.

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

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

[20] 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.

[21] 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.

[22] 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.

[23] 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.

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

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

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

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

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

[29] 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.