Efficient One-out-of-Many Proofs Using Gray Codes

Cargodog
cargodog@protonmail.com

October 8, 2020

Abstract

One-out-of-Many proofs provide an efficient construction for a zero-knowledge proof of membership of a secret committed value u belonging to a public list $\mathcal{L} = \{\lambda_1, ..., \lambda_N\}$. With logarithmic sized proofs and very efficient batch verification, they are widely used in zero-knowledge proof systems that require the use of very large anonymity sets. Batch verification is highly efficient, due to the ability share the computation of one large multi-exponentiation in the verification of independent proofs sharing the same anonymity set. While this allows additional proofs to be verified with only a few additional exponentiations each, each additional proof still requires N evaluations of polynomials of scalar coefficients. As a result, the number of proofs in a batch grows, the verification cost becomes dominated by the scalar multiplication operations necessary to evaluate each polynomial, instead of exponentiations.

We propose a modified construction of One-out-of-Many proofs, which leverages a Gray code sequence of integers to reduce the computational effort necessary to evaluate each scalar polynomial, and thus greatly improves batch verification performance. The Gray code sequence of integers guarantees that the Hamming distance between each subsequent integer is exactly 1. Given this, we show a technique to iteratively evaluate each subsequent polynomial by canceling out one term from the previous polynomial and factoring in the corresponding new term for the next polynomial. Such iterative evaluation reduces the number of scalar multiplications by a factor of $m/2$. Finally, we give concrete performance comparisons, demonstrating the improved efficiency of our new construction.

1 Introduction

Zero-knowledge membership proofs and ring signature schemes enable cryptographic systems to provide anonymity to participants, by hiding the identity of the prover within a set of decoy identities, known as the anonymity set. When implemented correctly, an outside observer will be unable to guess which identity belongs to the prover, with better probability than a random guess. This has applications in privacy preserving blockchain technologies [10][6], voting systems [3], and even communications protocols [2]. In such schemes, there is a direct relationship between the prover's anonymity and the size of the anonymity set. Generally speaking, a larger anonymity set provides greater anonymity. Unfortunately, larger anonymity sets increase proof size and complexity, causing system designers to choose an appropriate trade-off between performance and anonymity.

One-out-of-Many Proofs [5] provide a very efficient construction of membership proofs. Proof size only grows logarithmically with the size of the anonymity set, and proofs with overlapping anonymity sets may be verified in batch with only minimal extra computation for each additional proof in the batch. [1] describes a more general n-ary form this scheme, allowing the implementer to choose n and m, where the anonymity set size is $N = n^m$, and thus select a configuration that provides a desirable trade-off between anonymity and performance. For these reasons, [5] has been adopted in several higher level protocols that require strong anonymity [1][9][8][6].

1.1 Our contribution

Our work describes a slight modification to [5], which facilitates even more efficient generation and verification of proofs. As described in [5], the generation and verification of each proof consists of two distinct components. First, each proof includes a set of committed bits, and a proof that these bits are a valid n-ary

representation of an integer. Second, each proof contains the aggregate of low order terms of a polynomial, which when combined with a large polynomial evaluation over the anonymity set, cancel out all terms except the prover's member blinded by some offset. This second component is vastly more computationally demanding, requiring the computation of N exponents, and performing N exponentiations of each exponent with the its associated member in the anonymity set. Each exponent is computed as the evaluation of an n-ary polynomial, requiring m scalar multiplications in each computation. Our key innovation is the use of Gray codes [4][7] to facilitate efficient iterative computation of each coefficient, reducing the computation to a single scalar batch inversion and only two multiplications per coefficient, regardless of m. This results in approximately a factor of $m/2$ reduction in scalar multiplications.

While this performance benefits all cases for both the prover and the verifier, the most substantial performance gains are attained in batch verification. Since each proof verified in a batch may share the same exponentiation computation over the anonymity set, additional proofs require no additional exponentiations over the anonymity set. Each additional proof does, however, add N additional polynomials to be evaluated to compute each exponent, as well as the computations necessary to verify each bit commitment. Given this, as batch size increases, the dominant computational expense in verification is quickly dominated by the scalar arithmetic necessary to compute each exponent. In this work, we show that for even modest batch sizes, using our technique for iterative computation of each exponent offers significant performance benefits.

2 Preliminaries

2.1 Public parameters

Let $\mathbb{G}$ be a cyclic group in which the discrete logarithm problem is hard, and let $\mathbb{F}$ be its scalar field. Let $N = n^m$ be a positive integer, where $m > 1$.

2.2 Notation

For integers or field elements i, j, the Kronecker delta function $\delta(i, j)$ evaluates to 1 if $i = j$ and 0 otherwise, where the output is taken to be in the appropriate set. We sometimes use index subscript notation of the form i_j to indicate the jth digit of i, where such a decomposition of i is taken base n with padded length m:

$$\sum_{j=0}^{m-1} i_j n^j = i$$

2.3 Gray codes

Gray codes are described in [4] as an m digit binary code, in which each subsequent integer differs from its predecessor by only one bit. In other words, each integer in the code has a Hamming distance of 1, from its immediate predecessor and immediate successor in the sequence. For example, the three bit binary (i.e. $n = 2, m = 3$) Gray code sequence is given below:

$$\{000, 001, 011, 010, 110, 111, 101, 100\}$$

While [4] describes the code as a binary sequence, [7] describes a generalized form, which extends to n-ary codes. Importantly, for any n, each code will maintain the property that subsequent integers have a Hamming distance of 1. For example, the two bit ternary (i.e. $n = 3, m = 2$) Gray code sequence is given below:

$$\{00, 01, 02, 12, 11, 10, 20, 21, 22\}$$

We define a function $gray_n(k)$, which maps an n-ary integer k to an n-ary Gray code.

3 Gray coded One-out-of-Many protocol

3.1 Modified protocol

We modify the construction from [1], such that the set indices are represented as Gray codes. The original construction consists of two Σ-protocols to prove two key relations. We do not modify either relation, however we modify the Σ-protocol for the second relation, to use Gray coded set indices. The first Σ-protocol proves that a commitment B has an opening consisting of sequences of bits, where in each sequence there is exactly one 1. More precisely, the first is a Σ-protocol $(\mathcal{G}_{crs}, \mathcal{P}_1, \mathcal{V}_1)$ for the relation

$$\mathcal{R}_1 = \left\{ (B, (b_{0,0}, ..., n_{m-1,n-1}; r)) \mid (\forall i, j : b_{j,i} \in \{0,1\}) \wedge (\forall j : \sum_{i=0}^{n-1} b_{j,i} = 1) \wedge (B = Com_{ck}(b_{0,0}, ..., b_{m-1,n-1}; r)) \right\}$$

This proves that $b_{j,i}(1 - b_{j,i}) = 0$ for all i, j, and also that $\Sigma_{i=1}^{n} b_{j,i} - 1$ for all j. We do not modify this Σ-protocol in this work, but reproduce it in Fig. 1 for convenience.

$\mathcal{P}_1(ck, B, (b_{0,0}, ..., b_{m-1,n-1}, r))$		$\mathcal{V}_1(ck, B)$
$r_A, r_C, r_D, a_{j,1}, ..., a_{j,n-1} \leftarrow \mathbb{Z}_q$		
$\forall j : a_{j,0} := -\sum_{i=1}^{n-1} a_{j,i}$		
$A := Com_{ck}(a_{0,0}, ..., a_{m-1,n-1}; r_A)$	$\xrightarrow{A,C,D}$	
$C := Com_{ck}(\{a_{j,i}(1 - 2b_{j,i})\}_{j,i=0}^{m-1,n-1}; r_C)$		Accept if and only if
$D := Com_{ck}(-a_{0,0}^2, ..., -a_{m-1,n-1}^2; r_D)$		$A, B, C, D \in \mathbb{G}$
	$\xleftarrow{x \leftarrow \{0,1\}^\lambda}$	$f_{0,1}, ..., f_{m-1,n-1}, z_A, z_C \in \mathbb{Z}_q$
$\forall j, i : f_{j,i} := b_{j,i}x + a_{j,i}$		$\forall j : f_{j,0} := x - \sum_{i=1}^{n-1} f_{j,i}$
$z_A := rx + r_A$	$\xrightarrow{f_{0,1}, f_{1,1}, ..., f_{m-1,n-1}, z_A, z_C}$	$B^x A = Com_{ck}(f_{0,0}, ..., f_{m-1,n-1}; z_A)$
$z_C := r_C x + r_D$		$C^x D = Com_{ck}(\{f_{j,i}(x - f_{j,i})\}_{j,i=0}^{m-1,n-1}; z_C)$

Figure 1: Σ-protocol for relation $\mathcal{R}_1$.

The second Σ-protocol proves that the prover has knowledge of one out of N commitments $c_0, ..., c_{N-1}$ being a commitment to 0. More precisely, the Σ-protocol ensures the relation

$$\mathcal{R}_2 = \left\{ ((\{c_i\}_{i=0}^{N-1}), (l, r)) \mid (\forall i, c_i \in \mathbb{C}_{ck}) \wedge (l \in \{0, ..., N-1\}) \wedge (c_l = Com_{ck}(0; r)) \right\}$$

In the original work, the committed bits represent the index, l, of the prover's member in the set. In our modified construction, however, the bits instead represent the Gray code of the prover's index, $gray_n(l)$. While this difference in encoding requires adaptation to its associated Σ-protocol, relation $\mathcal{R}_2$ holds without modification. Fig. 2 gives the adapted Σ-protocol for $\mathcal{R}_2$.

Since we can pad the list with copies of the last ciphertext (at little extra cost in the protocol), we may assume $N = n^m$. The idea behind this Σ-protocol is to prove knowledge of an index l for which the product $\Pi_{i=0}^{N-1} c_i^{\delta_{l,i}}$ is a commitment to 0, where as usual $\delta_{l,i} = 1$ when $i = l$ and $\delta_{l,i} = 0$ otherwise. We have $\delta_{l,i} = \Pi_{j=0}^{m-1} \delta_{l_j, i_j}$, using our n-ary decomposition notation for l and i respectively.

The prover first computes the list of Gray codes g_i for every i in $\{0, ..., N-1\}$. Note, that Gray codes map into the same field as their source, so each Gray code will map to some other value in $\{0, ..., N-1\}$. This means that Gray codes can be substituted into the original protocol for set indices, and the net effect is just a reordering of the elements of the set. Since the set elements are always aggregated either by sum or product, the ordering of elements does not matter.

Next, the prover commits to m sequences of n bits $(\delta_{g_{l_j},0}, ..., \delta_{g_{l_j},n-1})$. It runs the Σ-protocol in Fig. 1 to prove that the commitment is well-formed. On receiving a challenge x, the prover discloses the elements $f_{j,i} = \delta_{g_{l_j},i} x + a_{j,i}$ as in Fig. 1. Observe that for every i in $\{0, ..., N-1\}$, the product $\Pi_{j=0}^{m-1} f_{j,i_j}$ is the evaluation at x of the polynomial $p_i(x) = \Pi_{j=0}^{m-1}(\delta_{g_{l_j},i} x + a_{j,i})$. For $0 \leq i \leq N-1$, we have

$$p_i(x) = \prod_{j=0}^{m-1} \delta_{g_{l_j}, g_{i_j}} x + \sum_{k=0}^{m-1} p_{i,k} x^k = \delta_{g_{l_j},i} x^m + \sum_{k=0}^{m-1} p_{i,k} x^k \tag{1}$$

for some coefficients $p_{i,k}$ depending on g_l (which depends on l) and $a_{j,i}$. Note that $p_{i,k}$ can be computed by the prover independently of x, and that $p_{g_l}(x)$ is the only degree m polynomial amongst $p_0(x), ..., p_{N-1}(x)$. Furthermore, since the Hamming distance between subsequent Gray code elements, g_i, is always 1, we see that each successive polynomial differs by exactly one term. This allows us to iteratively compute each polynomial, by dividing out a single term from the previous polynomial and multiplying in a single term from the next polynomial. This is the fundamental advantage of our work, which enables much more efficient batch verification. This is described further in Section 3.2.

From these coefficients and some random noise values ρ_k, the prover computes ciphertexts $G_k = \prod_{i=0}^{N-1} c_i i^{p_{i,k}} \cdot Com_{ck}(0; \rho_k)$ and includes them in the initial message. these ciphertexts are then used to cancel out the low degree terms in (1). Namely, if c_{g_l} is a commitment to 0, the following product is a commitment to 0 for any x:

$$\prod_{i=0}^{N-1} c_i^{\prod_{j=0}^{m-1} f_{j,g_{i_j}}} \cdot \prod_{k=0}^{m-1} G_k^{-x^k} = \left(\prod_{i=0}^{N-1} c_i^{\delta_{g_{l_j}, g_{i_j}}} \right)^{x^m}$$

$\underline{\mathcal{P}_2(ck, (c_0, ..., c_{N-1}), (l, r))}$		$\underline{\mathcal{V}_2(ck, (c_0, ..., c_{N-1}))}$
$r_B, \rho_k \leftarrow \mathbb{Z}_q, g_l := gray_n(l)$		
$B := Com_{ck}(\delta_{g_{l_{0,0}}}, ..., \delta_{g_{l_{m-1,n-1}}}, r_B)$		
$(A, C, D) \leftarrow \mathcal{P}_1(ck, B, (\{\delta_{g_{l_j}, i}\}_{j,i=0}^{m-1,n-1}; r_B))$		
For $k = 0, ..., m-1$	$\xrightarrow{A,B,C,D,\{G_k\}_{k=0}^{m-1}}$	Accept if and only if
$G_k = \prod_{i=0}^{m-1} c_i^{p_{i,k}} \cdot Com_{ck}(0; \rho_k)$		$A, B, C, D, G_0, ..., G_{m-1} \in \mathbb{G}$
using $p_{i,k}$ from (1)		$f_{0,1}, ..., f_{m-1,n-1}, z_A, z_C, z \in \mathbb{Z}_q$
	$\xleftarrow{x \leftarrow \{0,1\}^\lambda}$	$\mathcal{V}_1(ck, B, x, A, B, C, \{f_{j,i}\}_{j=0,i=1}^{m-1,n-1}, z_A, z_C) = 1$
$(f_{0,1}, ..., f_{m-1,n-1}, z_A, z_C) \leftarrow \mathcal{P}_1(x)$		$\forall j : f_{j,0} := x - \sum_{i=1}^{n-1} f_{j,i}$
$z := rx^m - \sum_{k=0}^{m-1} \rho_k x^k$	$\xrightarrow{f_{0,1}, f_{1,1}, ..., f_{m-1,n-1}, z_A, z_C, z}$	$\forall i \in \{0, ..., N-1\} : g_i := gray_n(i)$
		$\prod_{i=0}^{N-1} c_i^{\prod_{j=1}^{m} f_{j,g_{i_j}}} \cdot \prod_{k=0}^{m-1} G_k^{-x^k} = Com_{ck}(0; z)$

Figure 2: Σ-protocol for a list containing a commitment to 0.

3.2 Efficient computation of exponents

The reason we wished to modify the original construction with Gray coded indices, is to facilitate more efficient computation. Gray code's have a nice property, that subsequent codes in the sequence have a Hamming distance of exactly 1. Namely, each subsequent Gray code will differ from its predecessor by only one n-ary digit. Since each exponent is a polynomial evaluation of committed bits, this means each subsequent exponent may be computed by dividing out the relevant bit from its predecessor and multiplying in the corresponding new bit. This performance benefit comes at no cost to proof size.

Before elaborating, its useful to first understand the dominant computational costs of the original work. For large N, when using the Pedersen commitment scheme in an elliptic curve based group, the prover's and verifier's cost is dominated by a large multi-exponentiation of N group elements $\prod_{i=0}^{N-1} c_i^{\prod_{j=0}^{m-1} f_{j,i_j}}$. Each exponent is computed as a polynomial evaluation of committed bits. Each polynomial requires m scalar multiplications, for a total of $N * m$ scalar multiplications. Since these multiplications are relatively fast compared to curve group operations, they are often dismissed as only a minor impact to the overall computational complexity. While this may be true in the case of a single proof, the situation changes for batch verification of many proofs. Due to the efficient construction of [5], additional proofs that share the same set, may be verified in batch without incurring any more exponentiations over the set. When verifying p proofs in batch, each proof exponent may be computed and summed together, so that still only one exponentiation is required per element in the set. Namely, the set exponentiations may still be computed as a single multi-exponentiation over N group elements. Scalar operations, however, increase linearly with each proof in the batch. For a batch of p proofs, the number of scalar operations increases to $p * N * m$ scalar multiplications and $N * (p-1)$ additions to aggregate each exponent. Its easy to see that even for

a small batch, scalar operations may quickly exceed exponentiations in terms computational demand. The exact point at which this trade-off occurs depends on the relative performance of group operations vs scalar operations in your curve library, as well as hardware and software configurations of the verifier's setup.

In our modified work, adapted to use Gray coded indices, prover and verifier cost is dominated now by $\prod_{i=0}^{N-1} c_i^{\prod_{j=0}^{m-1} f_{j,g_{i_j}}}$, where g_i is the Gray code of the set index i. As described previously, the polynomials of subsequent exponents will differ by exactly one term. When computed iteratively, the incremental cost of computing each new exponent is only a single scalar inversion and 2 multiplications. Batch inversion techniques enable us to pre-compute all inversions for approximately the cost of a single inversion, reducing the incremental cost of each polynomial to just 2 multiplications. For a batch of p proofs, the total verification cost is dominated by the same multi-exponentiation over N group elements, but now only requires $p * m + 2p * (N - 1)$ scalar multiplications and $N * (p - 1)$ additions to compute the aggregated exponents. This is an improvement by approximately a factor of $m/2$ scalar multiplications. For large batches, this can result in significant performance savings. We give concrete performance comparisons in Section 5.

4 Security

In the original proof scheme described in [5], each member of the set is raised to an exponent, which is computed by evaluating a polynomial of terms representing the digits of that member's index within the set. The result of each exponentiation is then summed into one term, we call S. More precisely, given a set of commitments C_k, in which the prover knows the opening of C_l, S is computed as:

$$S = \sum_{k=0}^{N} C_k^{e(k)}$$

where, given the prover's committed bits $f_{j,i}$, the exponent $e(k)$ is computed as follows:

$$e(k) = \prod_{j=0}^{m-1} f_{j,k_j}$$

Our work modifies only this term, slightl, such that each exponent is computed not by its index k, but rather the Gray encoding of k. Given a function $gray_n(k)$ which maps an integer k to its corresponding Gray code, we can define the resulting aggregate term as:

$$S' = \sum_{k=0}^{N} C_k^{e(gray_n(k))}$$

We know from [4], that a Gray code sequence is merely a 1 to 1 mapping of the traditional integer sequence. Since a Gray code maps one to one from $k \rightarrow gray_n(k)$, it follows that the exponents are identically mapped from $e(k) \rightarrow e(gray_n(k))$. Subsequently, its trivial to see that S' can be equivalently be computed by remapping the members in the set, instead of the exponents. This comes with a small caveat; the prover's commitment in the remapped set will be at position $gray(l)$ instead of l as before. Now, S' may be rewritten as:

$$S' = \sum_{k=0}^{N} C_{gray_n(k)}^{e(k)} = \sum_{k=0}^{N} C_k'^{\,e(k)}$$

From here, it is easy to see that this construction is identical to the original, except that each member in the anonymity set has been mapped from $C_k \rightarrow C_k'$. Since the original work [5], does not depend on the selection or arrangement of members within the anonymity set, the security properties and their associated security proofs all hold without the need to adapt them to our modified construction.

5 Performance comparison

The performance benefit of using Gray coded indices depends on your selection of n, m, verification batch size p, as well as the relative performance of scalar multiplication vs exponentiation.The proof and verification cost for the original protocol are given in Table 1.

	Exponentiations	Multiplications
Prover	$(3 + 3nm) + (1 + nm + mn^m + m)$	$(2nm + 2) + (m^2n^m + m + 1)$
Verifier	$(4 + 2nm) + (n^m + m + 1)$	$(nm) + (mn^m)$

Table 1: Computational cost for prover and verifier in the original One-out-of-Many protocol.

Using the batch verification strategy of [6], additional verifications may be computed without requiring n^m exponentiations over the set for each proof in the batch. Due to this, additional proofs add many more scalar multiplications than exponentiations. As a result, batch verification cost becomes dominated by scalar multiplications as batch size increases. This is even more-so the case for large 'm'. This updated computational costs, accounting for batch verification of a batch of p proofs, are given in Table 4.

	Exponentiations	Multiplications
Verifier	$p(4 + 2nm) + (n^m + mp + 1)$	$(nmp) + (mn^mp)$

Table 2: Computational cost for batch verification (batch size p) in the original One-out-of-Many protocol.

Notice that the dominant term in the exponentiations is n^m, and that this term does not grow with p. This is the primary benefit of batch verification; the dominant source of exponentiations does not grow with the batch size, and can thus the average verification cost for each proof decreases with batch size. Unfortunately, the number of multiplications does not share the same advantage, and grows quickly with batch size. Our work addresses this, by allowing more efficient iterative computation of scalar exponents. As described in Section 3.2, we compute the set's exponent terms iteratively, and evaluate each polynomial by a single scalar inversion and two scalar multiplications applied to the previous polynomial. Since all scalar terms are known to the verifier in advance, the inversions may be batched together for approximately the cost of just a single inversion. Table 3 shows the computational cost for our work, using iterative computation of exponents.

	Exponentiations	Multiplications
Prover	$(3 + 3nm) + (1 + nm + mn^m + m)$	$(2nm + 2) + (2m(n^m - 1) + 2m + 1)$
Verifier	$(4 + 2nm) + (n^m + m + 1)$	$(nm) + (m + 2(n^m - 1))$

Table 3: Computational cost for prover and verifier in our protoco.

Notice, the dominant term of scalar multiplications is reduced from m^2n^m to $2m(n^m - 1) + m$ and from mn^m to $m + 2(n^m - 1)$ for the prover and verifier respectively. This is a factor of approximately $m/2$ on the dominant term of scalar multiplications. The batch verification costs for our work are givin in Table 3. From here its clear to see, that the benefit of this iterative computation of polynomials also grows with p; the dominant term of multiplications has been reduced from mn^mp to $(m + 2(n^m - 1))p$, a reduction by approximately a factor of $pm/2$.

	Exponentiations	Multiplications
Verifier	$p(4 + 2nm) + (n^m + mp + 1)$	$(nmp) + (m + 2(n^m - 1))p$

Table 4: Computational cost for batch verification (batch size p) in our protocol.

As stated previously, the real performance benefit of this technique depends on the relative cost of computing exponentiations vs scalar multiplications. This depends heavily on the nature of your cyclic

group, software configurations, and hardware setup. As such, giving concrete numbers for performance gain would not be meaningful. In our tests though, a proof setup using the Curve25519 elliptic curve group on a modest laptop, observed greater than 60% reduction in overall batch verification time for a batch of 100 proofs with $n = 2$ and $m = 2$.

6 Conclusion

The original construction of [5] provides a very efficient zero knowledge membership scheme. This efficiency is even greater when considering batch verification, because the dominant cost of exponentiations can be shared and amortized for each additional proof in a batch. Unfortunately, as we have shown, the scalar multiplications necessary to evaluate each polynomial, still grow linearly with the batch size. Due to this, at a certain point depending on m and batch size p, the cost of performing scalar multiplications exceeds the cost of computing exponentiations. Our modified construction using Gray coded indices, allows for much more efficient computation of each exponent, leading to significant performance gains in batch verification.

References

[1] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Essam Ghadafi, Jens Groth, and Christophe Petit. Short accountable ring signatures based on DDH. In *European Symposium on Research in Computer Security*, pages 243–265. Springer, 2015.

[2] Nikita Borisov, Ian Goldberg, and Eric Brewer. Off-the-record communication, or, why not to use pgp. In *In Proceedings of the 2004 ACM workshop on Privacy in the electronic society, WPES '04*, pages 77–84. ACM, 2004.

[3] Koutarou Suzuki Eiichiro Fujisaki. Traceabl06. https://eprint.iacr.org/2006/389.

[4] Frank Gray. Pulse code communication, 1950. Bell Telephone Laboratories, Incorporated. US Patent 2,632,058.

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

[6] Aram Jivanyan. Lelantus: Towards confidentiality and anonymity of blockchain transactions from standard assumptions. Cryptology ePrint Archive, Report 2019/373, 2019. https://eprint.iacr.org/2019/373.

[7] Dah jyh Guan. (scientific note) generalized gray codes with applications, 1998.

[8] Sarang Noether. Arcturus: efficient proofs for confidential transactions. Cryptology ePrint Archive, Report 2020/312, 2020. https://eprint.iacr.org/2020/312.

[9] Sarang Noether and Brandon Goodell. Triptych: logarithmic-sized linkable ring signatures with applications. Cryptology ePrint Archive, Report 2020/018, 2020. https://eprint.iacr.org/2020/018.

[10] Shen Noether, Adam Mackenzie, et al. Ring confidential transactions. *Ledger*, 1:1–18, 2016.